CAD Algorithms Dealing with Process and Temperature Effects in Digital Integrated Circuits

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Hushrav Mogal

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Kia Bazargan, Adviser

January 2010

© Hushrav Mogal January 2010

Acknowledgments

First and foremost, I would like to thank my adviser, Prof. Kia Bazargan, without the guidance and support of whom, none of this would be possible. He has encouraged me to think independently, something I will carry forward for the rest of my career. It has been a pleasure and I am ever indebted to him for all his contributions over the four years of my doctoral study.

I would like to thank Prof. Sachin Sapatnekar, for many reasons. My interest in statistical analysis arose out of his teaching the subject and over the past two years of my study, I gained invaluable insights by working closely with him. But most of all, for being an inspiration and someone I can lookup to in the years to come. It has been an honor working with him.

I would also like to thank Prof. Chris Kim and Prof. Antonia Zhai, for reviewing my thesis and giving me valuable feedback.

I would like to thank Yong Zhan and Tianpei Zhang, for their support as my office colleagues. I would also like to thank my friends Vijay, Sivakumaran and Ronita, for the kindness, affection and encouragement they have shown me throughout the years.

Lastly, I would like to thank my parents and family for this journey I have come through. You have taught me to believe in myself and I will be ever grateful for all you have done throughout my life.

Abstract

The aggressive scaling trend of the semiconductor industry to improve integrated circuit performance manifests itself as process, voltage and temperature (PVT) variations which can negatively impact design yield. The aim of this work is to deal with process (P) and temperature (T) effects and to develop software CAD analysis and optimization tools to mitigate their effects on digital integrated circuit performance.

In the first part of this thesis, we aim to develop an algorithm to compute the criticality of gates in a circuit with underlying process variations. The timing criticality of a gate indicates its impact on the overall timing performance of a circuit. We propose to use graph-based techniques to linearly traverse the timing graph of a digital circuit to obtain its criticality information. Such information can be useful to a designer or optimization tool in making decisions regarding gate sizing to improve the circuit performance. Our methodology must not only improve the speed of computation but also the accuracy with which we obtain the criticality values of gates in the circuit.

In the second part of this thesis, we propose to deal with temperature effects in the presence of increased scaling of devices. The sub-threshold leakage power of a digital chip, which is the wasted power in a digital circuit without doing any useful work, is exponentially dependent on the operating temperature of the chip. We propose to use techniques to exploit this dependence to reduce the sub-threshold leakage power. By rearranging the physical placement we can affect the temperature distribution of various blocks on a digital chip, thereby also affecting the total sub-threshold leakage power. We aim to develop a physical floorplanning tool to alleviate the temperature and sub-threshold leakage power by taking into account their interdependence.

This work proposes to use task migration (TM) as a methodology to deal with increasing sub-threshold leakage power in future technology nodes. The main idea is to replicate certain high-power blocks in the design, and migrate tasks at regular intervals from one part

of the chip to another, thereby reducing the power density and temperature of the design. We aim to develop a CAD optimization framework using floorplanning to read in a circuit description and produce a physical floorplan layout of the TM-aware design. This involves the selection of the design blocks to replicate, followed by the judicious placement of the blocks and finally the selection of an appropriate migration interval taking into account its negative impact on circuit performance.

The traditional semiconductor process technology consists of a single layer of silicon on which various devices like transistors and diodes are fabricated along with several layers of metal. Besides the problems outlined above, increasing device density is forcing larger die footprints. As a result, designers are facing increased congestion of routing wires, limiting the amount of performance benefit with scaling. Three-dimensional or vertical integration technology offers a promising alternative, in which multiple layers of silicon with their associated metal layers are stacked on top of each other. Field-programmable devices are particularly suited to such a technology due to the regular layout of logic and routing elements on the die. As the final part of this thesis, we examine the benefits of vertical integration applied to field programmable logic devices.

Contents

List of Tables x

List of Figures xii

List of Algorithms xviii

Chapter 1 Introduction 1

1.1 Dealing with Process Variation Effects 1

1.2 Dealing with Temperature Effects 2

1.3 Three-Dimensional Integration Technology 5

Chapter 2 Dealing with Process Variations 7

2.1 Static Timing Analysis 8

2.1.1 Computation of Arrival Times 8

2.1.2 Computation of Required Times and Slack 10

2.1.3 Critical Path 12

2.1.4 Timing Graph 12

2.1.5 Corner Based STA 14
2.2 Statistical Static Timing Analysis 15
2.2.1 Monte Carlo Analysis 16
2.2.2 Correlation Model 16
2.2.3 Modeling Gate Delays 17
2.2.4 The Statistical Summation and Maximum Operations 20
2.2.4.1 The Statistical Summation Operation 21
2.2.4.2 The Statistical Maximum Operation 21
2.2.4.3 The Tightness Probability 22
2.3 Statistical Criticality Computation 23
2.3.1 Problem Description 24
2.3.2 Background 26
2.3.3 Definitions 28
2.4 Fast and Accurate Criticality Computation under Process Variations 31
2.4.1 Cutset Computation 32
2.4.2 BSC: Basic SC Algorithm 33
2.4.3 Linear Time Book-keeping 34
2.4.4 Zone Computation 35
2.4.5 ZSC: Zone Based SC Algorithm 41
2.4.6 Errors in ZSC 43

2.4.6.1 The abc Problem 43

2.4.6.2 Local and Global Errors 44

2.4.7 Clustering Based Statistical Criticality Computation 47

2.4.7.1 ${}^{n}C_2$ Cutset Pruning 48

2.4.7.2 Clustering Based Cutset Pruning and Ordering 49

2.4.7.3 CPSC: Clustering Based SC Algorithm 54

2.4.8 Reducing Errors 57

2.4.8.1 LS: Localized Sampling 57

2.4.8.2 Timing Graph Reduction 58

2.4.8.3 Spatially Uncorrelated Independent Parameter Variations 60

2.4.9 Results 62

2.5 Conclusion 67

Chapter 3 Reduction of On-Chip Temperature via Static Thermal-Aware Floorplanning to Control Sub-threshold Leakage 69

3.1 Introduction 69

3.2 Power Dissipation in a Digital Design 70

3.2.1 Dynamic Power Dissipation 70

3.2.2 Subthreshold Leakage Power Dissipation 72

3.3 Thermal Modeling in a Digital Design 74

3.3.1 Finite Difference Thermal Model 74

3.3.2 HotSpot Thermal Modeler 78

3.4 Floorplanning 80

3.4.1 Sequence Pair Representation 81

3.4.2 Parquet Floorplanner 83

3.5 Microarchitecture Subthreshold Leakage Reduction Via Static Floorplanning 85

3.6 The Flow of Our Method 88

3.6.1 Architecture Profiling and Thermal Modeling 89

3.6.2 Performance Modeling 91

3.6.3 Floorplanning 93

3.6.4 Evaluation 94

3.7 Floorplanning Details 94

3.7.1 Transient Formulation Independent of Leakage Power Model 96

3.7.2 Simple Formulation Dependent on Leakage Power Model 98

3.7.3 Heuristic Moves 100

3.7.4 Whitespace Modeling 100

3.8 Results 103

3.8.1 Fidelity of Our Formulation 103

3.8.2 Leakage Savings 104

3.8.3 Maximum Temperature Reduction 105

3.8.4 Whitespace Considerations 107

3.8.5 Runtimes 107

3.9 Conclusion 108

Chapter 4 Thermal-Aware Floorplanning for Task Migration Enabled Active Sub-threshold Leakage Reduction. 109

4.1 Introduction 110

4.2 Preliminaries 114

4.2.1 Thermal Model 115

4.2.2 Leakage Model 115

4.3 Motivation 116

4.4 Floorplanning for Task Migration 119

4.4.1 Problem Statement 119

4.4.2 Candidate Replica Block Selection using TM Leakage Sensitivity . 120

4.4.3 TM-Aware Floorplan Patching 123

4.4.3.1 Deadspace Selection 123

4.4.3.2 Deadspace Allocation Via Bipartite Matching 124

4.4.3.3 Deadspace Substitution 127

4.4.4 Patching the Example Floorplan 128

4.4.5 Bringing it Together: Floorplanning 129

4.5 Experiments Setup and Results 131

4.5.1 SOC Benchmarks 131

4.5.2 Results 132

4.6 Conclusion 134

Chapter 5 CAD and Architecture Exploration of Three-Dimensional FPGAs 136

5.1 Introduction 136

5.2 Overview of TPR 138

5.3 SA-TPR: Simulated Annealing Based 3D Placement 139

5.4 Hybrid-TPR: Combined Partitioning and Simulated Annealing Based 3D Placement 142

5.5 Simulation Results 143

5.5.1 3D Architectures 143

5.5.2 Delay Results of TPR and SA-TPR Algorithms 144

5.5.3 Wire-length Results of TPR and SA-TPR Algorithms 145

5.5.4 Experiments using Hybrid-TPR 147

5.6 Conclusion 148

Chapter 6 Conclusion 150

Bibliography 152

List of Tables

2.1 Comparison of Monte Carlo and MAX_θ for the *abc* Problem 43

2.2 The ISCAS89 benchmarks with number of gates, N_G, and independent sources of variation, N_ζ. The effect of TGR on circuit depth, L, and maximum cutset size, η is also shown. 60

2.3 Criticality run-times and errors for various benchmarks; $\varepsilon = 5\%$ and $N_{ls} = 1000$ ZSC - Zone based criticality, ${}^nC_2$ - pairwise pruning scheme, CPSC - clustering based pruning scheme, TGR - timing graph reduction, LS - localized sampling 63

3.1 Microarchitecture configuration 90

3.2 Runtimes for different floorplanning schemes 108

4.1 Architectural and power characteristics of blocks shown in Figure 4.2. . . . 117

4.2 Leakage power evaluation for the TM configurations in Figure 4.2. 118

4.3 The SOC benchmarks. N_C, N_M and N_T denote the count of combinational, memory and total blocks in the design. P_{den} denotes the average power density of a module in the design. 132

4.4 Performance results for TM-aware floorplanning with respect to the floorplan area, leakage power, maximum chip temperature and half-perimeter wirelength. N_{TM} denotes the number of replica modules for the TM-aware floorplan. 133

5.1 Simulated circuits: statistics, vertical channel width (VCW) and run-time. . 148

5.2 Average of delay, wire-length (WL), horizontal channel width (HCW) and routing area for the *Sing-Seg* architecture. 148

5.3 Average of delay, wire-length (WL), horizontal channel width (HCW) and routing area for the *Multi-Seg* architecture. 149

List of Figures

1.1 Projected trend for transistor sub-threshold leakage current. 3

2.1 A simple circuit to illustrate Static Timing Analysis (STA). The three lightly shaded rectangles depict the levels or stages of the circuit. The interconnects have a delay of 0.5 and the gate delays are indicated by numbers in gray. The three vertically aligned numbers at the gate outputs and end-point (D input of $FF3$), represent the arrival time, required time and slack respectively. 9

2.2 Timing graph of the circuit in Figure 2.1. The shaded portions represent the region of the timing graph used to compute the arrival and required times for gate $U3$. 13

2.3 The grid-based spatial correlation model. Gates in the same grid square are tightly correlated whereas gates in distant grid squares are uncorrelated. . . 18

2.4 A simple circuit to illustrate the notion of criticality. The criticality of the gates is marked in gray. Representative delay distributions at the two output ports are also shown. 25

2.5 Example timing graph, G with cutsets, $\Sigma(1) - \Sigma(4)$. The shaded portions are nodes and edges used to compute the arrival and required time of edge e_{en}. The complementary path delay of e_{en} is the statistical maximum delay of all paths not passing through it, i.e., $e^{\sigma}_{en'} = MAX_{\sigma}(e^{\sigma}_{em}, e^{\sigma}_{bm}, e^{\sigma}_{fg}, e^{\sigma}_{ap})$. . . 28

2.6 Illustration of forward and reverse linear book-keeping data structures, Υ_F and Υ_R, to compute the complementary path delay of e_3 in cutset, $\Sigma(2) = \{e_{ap}, e_{fg}, e_{bm}, e_{em}, e_{en}\}$, from Fig. 2.5, with edges relabeled e_1, e_2, e_3, e_4, e_5 respectively. 34

2.7 An example timing graph, G, with mc-edges, $\Sigma_e = \{e_1, e_2, \ldots, e_6\}$. Edges e_x and e_y are not mc-edges. 36

2.8 Fig. 2.8(a) shows the mc-edges, $\Sigma_e = \{e_1, \ldots, e_6\}$, of the timing graph in Fig. 2.7, represented as half-open intervals. Fig. 2.8(b) shows the corresponding interval graph representation, G_e, with zones, $Z_1 = \{e_1, e_2, e_3, e_4\}$ and $Z_2 = \{e_5, e_6\}$, identified as mutually exclusive maximal cliques. 37

2.9 A pictorial depiction (not to scale) of the *abc* example with random variables a, b and c with one PC, p. 44

2.10 Illustration of the clustering based pruning procedure of Algorithm 5. Crosses indicate dominant objects and dots indicate non-dominant objects. The clustering distance is the local criticality (τ_{ai}) of an edge (i) from its cluster center (a). 52

2.11 Illustration of the timing graph reduction procedure. Nodes a, b, e and f are eliminated in the forward reduction phase. Node d is eliminated in the reverse reduction phase. 59

2.12 A reconvergent structure from one of the ISCAS89 benchmarks with a high criticality path indicated using bold lines. Arrival time correlations of fanouts in cutset Σ, denoted r_i, due to variation in oxide thickness, Δt_{ox}, of gate G_{11}, cause structural correlations in reconvergent fanouts like G_{41}. . . 61

2.13 Runtime of criticality computation as a fraction of SSTA runtime. The two cases shown are with and without the zone-based algorithm to compute the statistical maximum of mc-edges crossing a level in the timing graph. . . . 64

2.14 Trade-off showing number of LS samples, N_{ls}, vs the overall criticality computation runtime and maximum percentage error (with respect to a Monte Carlo simulation with 10000 samples), for the s38417 ISCAS89 benchmark. Runtimes are normalized to the case with $N_{ls} = 50$ and the error is normalized to the case with $N_{ls} = 10000$. 65

2.15 Trade-off showing pruning threshold, ε, vs the overall criticality computation runtime and maximum percentage error (with respect to a Monte Carlo simulation with 10000 samples), averaged over the 7 largest ISCAS89 benchmarks. The runtimes and error are normalized to the case with $\varepsilon = 1\%$. The number of samples used in LS, $N_{ls} = 1000$. 66

2.16 Comparison of runtime ratio and difference in maximum criticality percentage error between our implementation of the approach in [65] and the clustering based approach, referenced to a Monte Carlo simulation of 10000 samples. The number of samples used in LS, $N_{ls} = 1000$, and the pruning threshold, $\varepsilon = 5\%$. 67

3.1 Illustration of a grid of discretized elements in two dimensions used in the Finite Difference Method for thermal modeling. 75

3.2 Illustration of the HotSpot **block** and **grid** thermal models, on a toy chip layout with 5 blocks. Only the active device layer is shown. 79

3.3 Parquet uses the concept of slack from static timing analysis to optimize floorplan area. The slack of block A in the x and y dimensions is shown in the figure. 84

3.4 Overall flow of our leakage reduction and evaluation methodology. 88

3.5 Compacted floorplan for the core of the Alpha 21264 microarchitecture. The L2 cache surrounding the core is not shown. 90

3.6 Busses modeled for performance evaluation. The fetch unit is considered part of IL1. 92

3.7 Illustration of the details of the floorplanning process. Figure 3.7(b) shows how we break the thermal loop of Figure 3.7(a) using our own leakage computation technique. 95

3.8 Execution of the sweep-line on a toy example. Disjoint dotted lines are whitespace intervals showing the advance of the sweep-line at event points. A, B, C are blocks and 1-8 are whitespaces filled in. Two event points between 3.8(c) and 3.8(d) are not shown. 102

3.9 Normalized fidelity plots for Equation 3.26 ‘transient’, Equation 3.29 ‘estimated’ and accurate transient simulations ‘accurate’. The leakage values are normalized to a chosen base floorplan and are numbered in increasing value of their ‘accurate’ leakage for better visualization. 103

3.10 Normalized graphs of the leakage power and CPI in comparison with a floorplan optimized without leakage called Base Case. Y-axis begins at 0.75. 104

3.11 Normalized graphs of leakage power with maximum temperature and transient formulation as objective functions compared to a base floorplan without leakage optimization. Y-axis begins at 0.75. 105

3.12 Normalized graphs showing the impact of WS on leakage reduction. Bar 1, 2 and 3 are leakage with no WS modeling, 10% WS and 15% WS modeling during floorplan optimization respectively. Y-axis begins at 0.75. 106

4.1 Task migration illustrating the activity time-line of the primary, and shadow (or replica) set of blocks in a design. 113

4.2 An example illustrating the motivation for TM via floorplanning. Figure 4.2(a) shows design blocks $A \ldots G$, and S_1, S_2, S_3 represent the three main deadspace regions in the floorplan. Figures 4.2(b) to 4.2(f) show different configurations of the primary and replica blocks (denoted with a '$_R$' suffix) for TM. 116

4.3 Illustration of deadspace allocation using bipartite matching. 126

4.4 Execution of TM-aware floorplan patching on the base floorplan in Fig. 4.4(a). Fig. 4.4(b) shows the selection of a candidate TM block C (in red) for TM, followed by the selection of candidate deadspace locations DS_1, DS_2, DS_3 (in green) in Fig. 4.4(c), matching of C with DS_2 (in pink) in Fig. 4.4(d) and finally insertion of TM replica block C_R to obtain the patched floorplan in Fig. 4.4(e). Fig. 4.4(f) shows the case of floorplan patching with two candidate TM blocks selected. 129

4.5 Overall flow of our TM-aware floorplanning approach. 130

4.6 Chip temperature contours for floorplans without and with TM. The replica module is labeled $SOC8_1$. 134

5.1 Flow diagram of TPR: 3D Placement and routing tool is similar to VPR's flow diagram. 138

5.2 Flow diagram of SA-TPR: The partitioning based placement is replaced with a simulated annealing engine. 140

5.3 Two possible placement scenarios of the same net e with different values of S_Z and N_Z. 141

5.4 Flow diagram of Hybrid-TPR: The partitioning based initial placement is followed by a simulated annealing based layer refinement. 142

5.5 Illustration of the 3D FPGA architecture. 143

5.6 Variation of delay as reported after detailed routing. 144

5.7 Variation of delay as reported after detailed routing. 146

5.8 A 3D switch box. The third dimension adds vertical tracks which require 5 connections. 147

List of Algorithms

1 BSC $(G(V, E))$ // G = circuit timing graph 33

2 $Z =$ ComputeZones (Σ_e)

// Σ_e = mc-edges in the timing graph organized as per levels

// Z = list of mutually exclusive zones, $\{Z_1, \ldots, Z_K\}$ 39

3 ZSC $(G(V, E))$ // G = circuit timing graph 42

4 nC2 (Σ, ε)

// $\Sigma =$ cutset of edges; $\varepsilon =$ pruning threshold 49

5 $K =$ KCenterPrune (Σ, ε, S)

// $\Sigma =$ cutset of edges; $\varepsilon =$ pruning threshold;

// $S =$ maximum cluster size; $K =$ # clusters 50

6 σ = CreateNewCluster (Ω)

// $\Omega =$ set of clusters; $\sigma =$ new cluster 51

7 CPSC $(G(V, E), \varepsilon)$

// G = circuit timing graph; ε = pruning threshold 55

8 LS (Σ, Ψ, R)

// Σ = cutset; $\Psi_p = N_{ls}$ x k array of i.i.d. gaussian samples

// $R = N_{ls}$ x $|\Sigma|$ array of i.i.d. gaussian samples 57

9 $ComputeWhitespace(Blocks, N, W, H)$ 101

10 Φ_R = TMCandidates $(\Phi_P, \mathcal{F}_\mathcal{P}, \rho)$ // Φ_P = set of primary blocks in base floorplan $\mathcal{F}_\mathcal{P}$; ρ = allowable area budget; Φ_R = set of candidate replica blocks partaking in TM 122
11 DS_R = DSSelection $(\mathcal{F}_\mathcal{P}, \Phi_R, \epsilon)$ // $\mathcal{F}_\mathcal{P}$ = base floorplan; Φ_R = set of candidate replica blocks partaking in TM; ϵ = deadspace threshold; DS_R = subset of deadspaces in floorplan $\mathcal{F}_\mathcal{P}$ that are candidates for accommodating replicas Φ_R 124
12 M = DSMatching $(\mathcal{F}_\mathcal{P}, \Phi_R, DS_R)$ // $\mathcal{F}_\mathcal{P}$ = base floorplan; Φ_R = set of candidate replica blocks partaking in TM; DS_R = selected deadspaces in floorplan $\mathcal{F}_\mathcal{P}$; M = matching between blocks in Φ_R and DS_R 127
13 $\mathcal{F}_\mathcal{R}$ = DSSubstitution $(\mathcal{F}_\mathcal{P}, \Phi_R, M)$ // $\mathcal{F}_\mathcal{P}$ = base floorplan; Φ_R = set of candidate replica blocks partaking in TM; M = matching between blocks in Φ_R and floorplan deadspaces; $\mathcal{F}_\mathcal{R}$ = patched floorplan with TM blocks . 128

Chapter 1

Introduction

Very large scale integration (VLSI) Compute-Aided Design (CAD) deals with developing algorithms and methodologies integrated into tools used to analyze and improve the performance of an integrated circuit (IC) design. Over the recent years, the primary focus of Electronic Design Automation (EDA) companies has been to design tools to cope with the semiconductor industry trend of scaling devices (each new generation scaled device is referred to as a *technology node*) as outlined in the International Technology Roadmap for Semiconductors [55]. Although such a scaling improves the performance, i.e., speed, of digital circuits, it also manifests itself as process, voltage and temperature (PVT) variations which negatively impact the performance of the IC. The aim of this thesis is to deal with process (P) and temperature (T) effects and to develop CAD optimization techniques to mitigate their effects. This chapter very briefly outlines how this is accomplished.

1.1 Dealing with Process Variation Effects

Process variations refer to variations in the device or interconnect parameters of a semiconductor chip that occur due to artifacts in the fabrication process. Although present in previous technology generations, they have become more significant due to the scaling of

transistors and interconnects. For instance the industry roadmap specifies a target variation in gate length, L, of 12%. However, they acknowledge the fact that such a target will be difficult to meet in the coming years. Furthermore, we can no longer predict gate delay. Variations such as these make circuit performance metrics less predictable. Hence, we need a paradigm shift in our approach to deal with such variations in the design. Statistical analysis is one such technique.

In traditional timing optimization methods, each gate is assumed to have a deterministic delay. Performing timing analysis on the timing graph of a circuit gives us a set of gates which impact the circuit delay the most (these are called *critical gates* and they lie on the path with the maximum delay, called the *critical path*). By selectively sizing these gates we can improve the critical path delay and meet the timing specification. However, under process variations, each gate now has a nondeterministic or random delay. In such an environment, no one path is critical in every die of the circuit design. The first part of this thesis deals with computing those gates in a circuit which can impact the delay the most. This work aims to:

- compute the statistical criticality of gates with large process variations.
- improve the accuracy and runtime of the criticality computation procedure using a clustering-based approach to eliminate candidates which contribute the least to the critical path delay.
- use a clustering-based framework to improve the accuracy of the statistical MAX operation.

1.2 Dealing with Temperature Effects

Figure 1.1 shows the ITRS projected off-state sub-threshold leakage current with scaling of technology nodes. As can be seen, the subthreshold leakage current (hereby also referred to

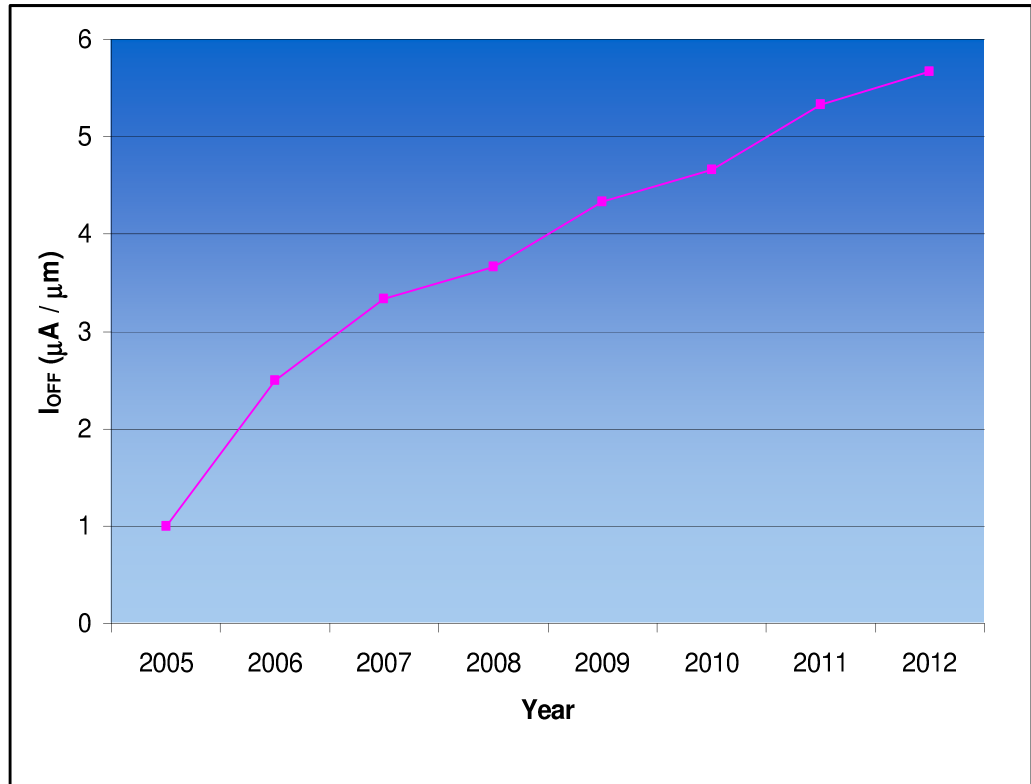


Figure 1.1: Projected trend for transistor sub-threshold leakage current.

as leakage in the rest of this document) is increasing with each technology generation. This is mainly due to threshold voltage scaling to increase the transistor on-current drive in order to improve the speed of circuit operation. However, since the sub-threshold leakage current depends exponentially on threshold voltage, this increases the circuit power dissipation. In addition, device scaling is no longer accompanied by an equally proportional scaling of the supply voltage [55], which has slowed down considerably over the last few years. Device scaling also brings about increased device density. All these factors result in an increase in the power density, which leads to higher on-chip temperatures.

We also know that the sub-threshold leakage current (and therefore power) is exponentially dependent on temperature. Therefore, an increase in temperature increases the leakage power which in turn leads to higher temperatures creating a positive feedback loop. Hence,

any optimization procedure dealing with temperature effects must take this feedback effect into account. We aim to deal with the problem of increasing on-chip temperatures in a high performance microarchitecture by taking into account the exponential dependence of leakage. This work explores the use of microarchitectural techniques to deal with temperature hotspots in the design. The first part of this work uses floorplanning to rearrange the blocks in a design so as to directly impact the sub-threshold leakage. Towards this end, this work:

- develops models which capture the exponential dependence of leakage on temperature.
- uses the models in a floorplanning optimization algorithm to reduce the overall leakage of a microarchitecture design.

Although microarchitecture floorplanning techniques are effective in reducing the leakage power of a design, they cannot adapt to the operating environment. The second part of this work revisits a technique called task migration (TM) whereby high activity blocks in a microarchitecture are replicated so as to reduce the active power density. This in turn leads to a reduction in temperature and therefore leakage power of a design. Such a technique is similar to a dynamic thermal management (DTM) scheme used to deal with high on-chip temperatures. We however propose to use this as a scheme to reduce the active sub-threshold leakage power consumption by exploiting the exponential dependence of leakage on temperature. Towards this end, this work:

- is the first to define the task migration (TM) problem in the floorplanning context to reduce the active sub-threshold leakage
- proposes a floorplanning framework which uses TM as a means of reducing sub-threshold leakage in system-on-chips.

1.3 Three-Dimensional Integration Technology

Three-dimensional (3D) device fabrication holds promise in dealing with the challenges that arise from device scaling. To accommodate the ever-increasing transistor count brought about by scaling, this design paradigm stacks multiple dies on top of each other instead of increasing the die area of a chip. With each technology generation and increasing frequency of operation, the same length of wire has a much greater impact on performance. As a direct consequence of vertical integration, interconnect lengths reduce thereby improving performance. The last part of this work deals with 3D IC technology applied to Field Programmable Gate Arrays (FPGAs). This work aims to:

1. develop placement algorithms to augment the Three Dimensional Place and Route (TPR) package targeted towards FPGAs. This work is part of a much larger flow of CAD tools for 3D integration [5].
2. use our algorithms to explore different 3D FPGA architectures and evaluate the potential benefits of 3D integration.
3. provide academia with these tools to facilitate advances in 3D FPGA research.

The rest of this thesis explains our main goals in greater detail, providing the methodologies used in achieving them. Chapter 2 describes an algorithm to compute the statistical criticality of gates in a circuit in linear time. We also describe a clustering based pruning algorithm to deal with errors in the criticality computation procedure with applications to improving the accuracy of the statistical MAX operation. Chapter 3 describes a floorplanning algorithm to reduce the sub-threshold current of a microarchitecture. Chapter 4 explains the task migration paradigm and its use in reducing the on-chip temperature profile. It also describes a floorplanning framework which uses TM to reduce the leakage power consumption of a design by exploiting its dependence on temperature. Finally, Chapter 5

describes two placement algorithms and presents studies on two 3D FPGA architectures to evalutate their benefits compared to the existing 2D FPGAs.

Chapter 2

Dealing with Process Variations

With ever shrinking device geometries, process variations play an increased role in determining the delay of a digital circuit. Under such variations, a gate may lie on the critical path of a manufactured die with a certain probability, called the criticality probability.

This chapter deals with the variations in device and interconnect parameters on a semiconductor chip due to fabrication process variations. Although such variations have existed in older technology nodes, they have a more pronounced effect in the sub-nanometer regime. This is because semiconductor devices and interconnects have aggressively scaled with each technology node and the magnitude of fabrication process variations have more or less remained the same. As a result, such process parameter variations can render the timing delay of a circuit as unpredictable [46], making it difficult to sign-off against chip failure with the existing conventional approaches to static timing analysis (STA).

This chapter begins with a brief introduction to static timing analysis. We then overview statistical timing analysis, a new paradigm to deal with variations in a circuit. We then motivate the need for criticality computation in a circuit. This is followed by a survey of the current techniques used to compute the criticality of gates in a circuit. The rest of the chapter explains in detail our approach and presents some results depicting its usefulness.

2.1 Static Timing Analysis

Static timing analysis is a method to verify if a circuit meets its timing requirements. The name static is used to denote the fact that the verification is performed without using input vectors, i.e., without any input stimulus. The main reason for the popularity of STA is its capacity to handle very large designs, which otherwise cannot be verified by Spice-like circuit simulation tools. Figure 2.1 shows a simple circuit to illustrate the technique of STA. The circuit consists of three flip-flops (FFs) and four gates ($U1$, $U2$, $U3$ and $U4$) with one path between $FF1$ and $FF3$ and two paths between $FF2$ and $FF3$. Each gate and the wire connecting them enables the propagation of a digital signal with some delay from its input to its output. The basic goal of STA is to compute the slowest timing path in the circuit, or in other words, the path which constrains the operating speed of the circuit. This can be done either by enumerating all the paths in the circuit, an approach called path-based analysis, and picking the slowest path or by propagating the signal as a wavefront from all the inputs to all the outputs simultaneously, an approach called a block-based analysis. Although the path-based analysis technique is accurate because it concentrates on a single path, it is much slower than the block-based technique since a digital circuit potentially contains an exponential number of paths. Since the main focus of this work relates to block-based approaches, we briefly describe the block-based STA technique. Conceptually, the basic goal is to compute three numbers called the arrival time, required time and slack at each terminal of the circuit, denoted by a triplet of green, blue and red in the figure. We next describe how each of these quantities are computed.

2.1.1 Computation of Arrival Times

Let us assume that each wire connecting the pins of gates in the circuit has a delay of 0.5 units. Additionally, the delays from each input pin of a gate to its output is shown in grey. For example, the delay of gate $U1$ is 1.4 units. The connections between terminals of gates and FFs in the circuit are traditionally known as timing arcs. To compute arrival

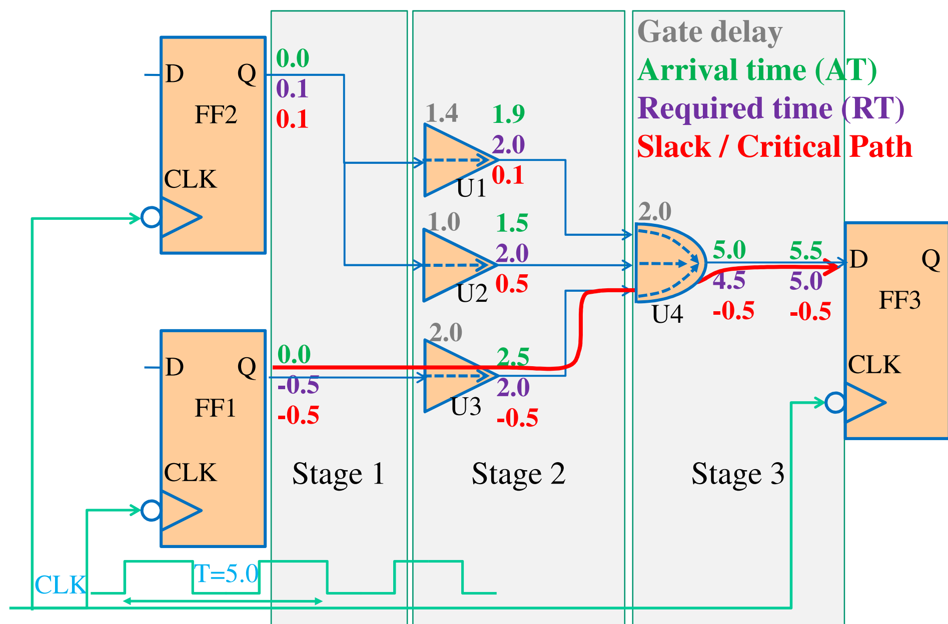


Figure 2.1: A simple circuit to illustrate Static Timing Analysis (STA). The three lightly shaded rectangles depict the levels or stages of the circuit. The interconnects have a delay of 0.5 and the gate delays are indicated by numbers in gray. The three vertically aligned numbers at the gate outputs and endpoint (D input of $FF3$), represent the arrival time, required time and slack respectively.

times (denoted as AT), we propagate the timing information along the timing arcs from the inputs of the circuit (here the Q pin of $FF1$ and $FF2$) to the outputs (here the D pin of $FF3$). Two basic operations are performed during an STA.

SUM The first is a sum operation from the output of a gate along gate timing arcs of one stage to the input of gates of the following stage. For example, propagating the signal from the output of gate $U2$ to the input of gate $U4$, we obtain a signal arrival time of $1.9 + 0.5 = 2.4$.

MAX The second operation is a maximum operation at the output of a multiple input gate. Since the goal is to compute the latest arriving signal which could violate the timing requirements, we take the maximum of the arrival times of signals through different paths which converge at the output of a gate. For example, at the output of gate $U4$, the arrival time of the path originating from flip-flop $FF1$ is 5.0 and the arrival times of the paths originating from $FF2$ are 4.0 and 4.4. Therefore, at the output of $U4$ we compute the arrival time as 5.0.

Finally, at the D input of $FF3$ the total maximum arrival time is computed as 5.5 units. Typically, a synchronous circuit is designed to operate at a desired speed dictated by the maximum frequency of the clock signal feeding the CLK inputs of the flip-flops. This is referred to as a maximum timing constraint, T. In our example, the circuit must meet a timing of $T = 5$ units, i.e., the data signal from any path starting from the output of an FF leading to the input of an FF should arrive no later than 5 units with respect to the clock. In our example therefore, the circuit does not meet the timing requirements.

2.1.2 Computation of Required Times and Slack

Although the arrival time at the circuit output tells the designer if timing requirements are met, it would be of no use in particular if she is not able to target portions of the design

which are responsible for the slow circuit behavior. This is where the notion of slack helps.

The slack at any pin or terminal in the circuit quantitatively tells the designer the amount of time units the signal can possibly be delayed from its current arrival without violating the timing of the circuit. For example, the slack at the output of gate $U1$ is 0.1 which means that the signal can arrive at most 0.1 units later than its current arrival time without violating the timing of the circuit. This can easily be seen if the delay of gate $U1$ were 1.6 instead of 1.4. Moreover, a negative slack implies that the signal arrives too late to meet the circuit timing. Therefore the signal at the output of $U3$, which has a slack of -0.5, must actually be sped up by 0.5 units to meet the timing constraint. This would be the case if, for example, the delay of $U3$ were 1.5 units instead of its current value of 2.0 units.

To compute the slack at each pin in the circuit we compute what are called required times (denoted as RT). This is done by backward propagating the signal from the output of the circuit to its inputs, beginning with the timing constraint. At the circuit output we know that the signal should arrive no later than the required 5.0 units (the timing constraint, T). This is used to compute the required time at the output of $U3$ as 4.5 units. For example, the required time at the output pin of gate $U2$ is computed as 2.0 units, implying that for the circuit to meet timing requirements, the signal should arrive no later than 2.0 units at the pin. For a gate with multiple fanouts, the required time at the output pin is computed as the minimum of the required times of all paths converging at the pin. So, the RT at the output Q of $FF2$ is computed as the minimum of 0.5 and 0.1.

The slack at a pin can now be computed by computing the difference of the arrival and required times at a pin. For example, the slack at the output pin of $U3$ is computed as $2.0 - 2.5 = -0.5$.

2.1.3 Critical Path

The concept of the critical path helps the designer deal with the circuit at the path level and identify a single path with the worst delay or arrival time in the circuit. In other words, this is the path that truly constrains the speed of the circuit. The critical path delay, as its name implies, refers to the delay of the critical path. In Figure 2.1 this is highlighted in red and is the path from the output of $FF1$ to the input of $FF3$ with a delay of 5.5 units. To identify the critical path, we begin with the output and trace backwards the pins with the lowest slack. This is logical since the slack at a pin gives a quantitative measure of the worst of all paths passing through it. If the circuit exactly meets the timing requirements, the slack along all pins of the critical path would be zero. In a circuit however, there can be more than one path which does not meet timing requirements, i.e., paths with slack less than zero. These are referred to as sub-critical paths.

Designers typically try to first speed up the critical path in the circuit (using techniques like buffer insertion and gate sizing) to meet timing requirements. In modern designs which are highly optimized, there are hundreds of paths which are close to critical, creating the so called slack wall, in reference to the histogram of the slack of paths in the design.

2.1.4 Timing Graph

We have described a method to compute the maximum delay of a circuit. Typically this is accomplished by using a timing graph, defined below.

Definition 2.1.1 (Timing Graph). A timing graph $G(V, E)$ of a circuit is a directed acyclic graph with V nodes representing gate terminals and E edges representing connections between the terminals. Primary inputs and outputs are connected, respectively, to a virtual source node, v_s, and a virtual sink node, v_t.

The addition of nodes v_s and v_t is done to simplify the analysis, since now any path in

the timing graph must pass through these nodes. Figure 2.2 shows the timing graph representation of the example circuit in Figure 2.1. The shaded portion depicts the region of the timing graph used to calculate the arrival and required times of the node $U3$, denoted, AT_{U3} and RT_{U3} respectively.

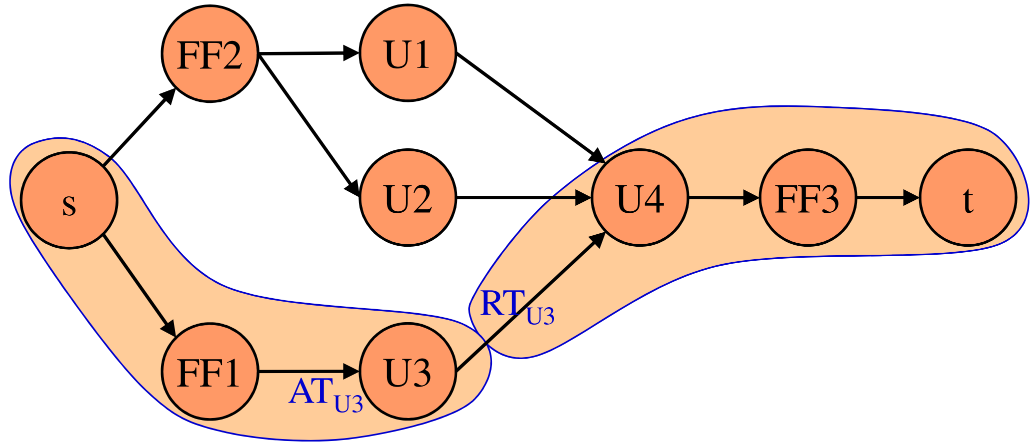


Figure 2.2: Timing graph of the circuit in Figure 2.1. The shaded portions represent the region of the timing graph used to compute the arrival and required times for gate $U3$.

Every node and edge in the graph G represents the timing arcs in the circuit. As described previously, to compute the timing of the circuit, we begin from the virtual source node, v_s, and in a breadth first traversal, perform a SUM operation along the nodes and edges. At the input of nodes with a fanin greater than one, a MAX operation is performed. This forward traversal computes the arrival time of all the nodes and edges in G. A similar traversal is performed in the reverse direction starting from the virtual sink node, v_t, to compute the required time of all the nodes and edges in G. Finally, the slack at each node is computed by subtracting the arrival time from the required time. With respect to the timing graph we define the arrival and required time as follows.

Definition 2.1.2 (Arrival Time). The arrival time AT at an edge/node in timing graph, G,

is the maximum delay from any primary input to the edge/node.

Definition 2.1.3 (Required Time). The required time RT at an edge/node in timing graph, G, is the maximum delay from any primary output to the edge/node.

2.1.5 Corner Based STA

Designers test their chips by performing STA at what are referred to as corners. These reflect the various operating conditions under which the chip may operate. Typically, traditional STA uses three corners, namely nominal, fast and slow. These as their names indicate are the mode in which the chip typically operates, the mode in which the chip is at its fastest, and the mode in which the chip is at its slowest. Satisfaction of chip timing in one corner is not a guarantee of the chip working reliably in a different corner. The assurance that the chip meets its timing requirements under all three modes is called timing sign-off and is usually the last step of the digital design flow before tape-out. Such an analysis may require three separate STA runs, to verify the chip in each of the corners.

Note that the above description of STA is a very simple one. At the very least, the flip-flops have internal timing arcs which refer to the set-up and hold times. The MAX operation is used to test the set-up requirements of the chip. A similar check for the hold-time requirements is done by performing an STA in which the MAX operation is replaced by a MIN operation to find the fastest path through the circuit. In a real design there are many other issues which need to be taken into consideration, for instance, false paths, multi-cycle paths and cross talk effects between wires to name just a few. Nevertheless, the basic technique of block-based STA is still the same, with the computation of the slack, arrival and required times using the SUM and MAX operations. For a more in depth and detailed description of timing analysis, please refer to [53].

2.2 Statistical Static Timing Analysis

With the coming of the sub-nanometer era, device scaling brings with it a host of issues to resolve. A crucial artifact of device fabrication is process parameter variations. Although variations in the device geometries have existed for the past several technology nodes, the magnitude of these variations has remained more or less the same, thereby becoming increasingly significant in light of device scaling. For example, the variation in the gate length of a transistor implies a non-deterministic delay for the gate. The non-deterministic behavior refers to the fact that the same transistor fabricated on a different chip would have a different delay because its size may be different. The ITRS [55] has set a goal of 12% variation in gate length, which by their own admission is very aggressive. Besides gate length variations, there are many other types of variations that come into play, some of them being the thickness of metal wires forming the interconnects, the dielectric thickness, the oxide thickness of the transistor and the variations in the dopant concentration. All these variations render the delay of the circuit as unpredictable [46], making the regular STA based timing sign-off techniques ineffective in assuring against chip failure. The ITRS [55] predicts that in the next few years, such variations will cause the sign-off delay to vary by as much as 18%.

Such variations cause an explosion in the number of corners which need to be verified, which is already becoming impractical due to the exponential number of STA runs needed. Moreover, such a corner based approach is inherently pessimistic since not all chips are operating at their absolute corners. Over the recent years, CAD tools have integrated variability aspects in the design flow to deal with process variations. As an immediate consequence of this variability, the delay of the circuit cannot be characterized by a single deterministic number but a probabilistic distribution instead. Statistical static timing analysis (SSTA) is a technique used to predict the circuit delay reliably in the face of process parameter variations, by computing an accurate representation of the circuit delay distribution. We briefly

describe the SSTA flow, which includes the correlation model used to capture the process parameter variations.

2.2.1 Monte Carlo Analysis

Monte Carlo analysis is a relatively simple technique to mathematically model a process with statistical variation. In brief, the system is sampled at every point in the variation space (the inputs to the system are statistical quantities) and simulated for the outcome. By aggregating the response of the system at several sample points, we can obtain an accurate overall response of the system. For our application, timing analysis, with statistically varying process parameters, which are the input to the system, a Monte Carlo analysis would proceed by sampling each process parameter according to its variation model (if multiple process parameters are correlated, the sampling should take that into account), modeling the gate and interconnect delays at this sample point, and performing a regular Static Timing Analysis (STA) to obtain a deterministic response of the system. By repeating this process at several sample points, the output of system, which is the timing of the circuit, is represented as a set of points, which characterizes the distribution of the circuit delay. Although simple in nature, the accuracy of the method depends on the number of samples chosen and typically is inversely proportional to the square root of the number of samples. For reasonable accuracy, typically at least 10000 Monte Carlo simulations need to be performed, which is very expensive. Therefore, smarter techniques are needed to accurately reproduce the circuit delay distribution in a reasonable amount of time.

2.2.2 Correlation Model

As discussed above, the manufacturing process introduces variations in device parameters. The variations on a chip manifest themselves as inter-die and intra-die variations. Inter-die variations, as its name implies, are variations that affect different dies, or in other words, a transistor at a particular location has a different critical dimension on different dies. Such

variations are also referred to as global variations, since they equally affect all the devices on a single die. Intra-die variation refers to variations within a die, or in other words, transistors have a different critical dimension depending on which region of the chip they lie in. Such variations are also referred to as local variations since they affect the devices of the same die differently. Variations are also categorized according to their mathematical modeling. Random variations characterize parameters such as gate oxide thickness, t_{ox}, and dopant fluctuations, N_A, which are essentially random in nature. On the other hand, parameters such as gate length, L_g, are referred to as spatially correlated because the correlation between the variations of different devices can be captured by a model which takes into account their spatial locations on the chip. To model spatially correlated parameters, we use the spatial correlation model in [15], illustrated in Figure 2.3. Briefly, the chip is divided into a uniform, $N \times N$, grid in which gates in a grid square are assumed to have perfect correlation and gates in far-away grid squares have weak correlations. A covariance matrix of size equal to the number of grid squares is obtained for each modeled parameter, such as gate length, L_g. The model includes the global (inter-die) component of variation common to each entry of the matrix. In our work, we model the length and width of gates, and the thickness, width and inter-layer dielectric thickness for interconnects. It is assumed that zero-correlations exist between different types of parameters, for example the length and width of a gate. In addition, we also model the gate oxide thickness, tox, assuming independence between the different gates on the chip. All process parameters are assumed to have a Gaussian distribution as in [15, 60].

2.2.3 Modeling Gate Delays

Every gate delay, d_i, is expressed using a first order Taylor expansion of its device parameters. For example a gate which depends on the gate length parameter, L_{gi}, given by the relation $f(L)$, is expressed as shown in Equation 2.1. This is arguably adequate to obtain an accurate representation of the statistical gate delay if the magnitude of the variations is

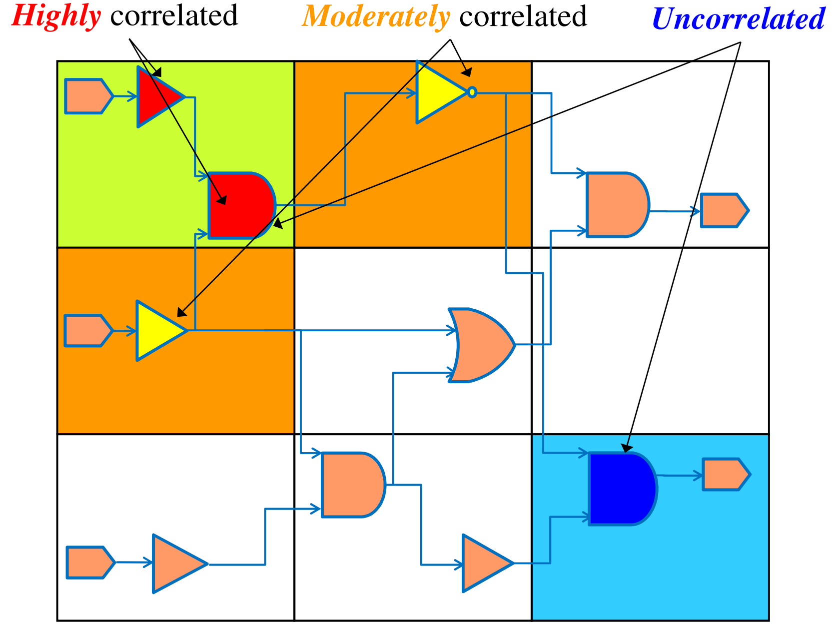


Figure 2.3: The grid-based spatial correlation model. Gates in the same grid square are tightly correlated whereas gates in distant grid squares are uncorrelated.

small.

$$\begin{aligned} d_i &= f(L_{g1}) \\ d_i &= d_0\Big|_{L_0} + \frac{\partial f}{\partial L_{gi}} \cdot \Delta L_{gi} \end{aligned} \tag{2.1}$$

In the above equation, L_0 is the mean of the gate length L_{g1}, and ΔL_{gi} is the variation in the gate length L_{gi}. As described previously, each process parameter is also expressed by an $N \times N$ correlation matrix, describing the correlation of the parameter on various regions of the chip. For example the gate length variation L_g can be expressed as shown below.

$$\begin{bmatrix} \Delta L_{g11} & L_{g12} & \dots & L_{g1N} \\ \Delta L_{g21} & L_{g22} & \dots & L_{g2N} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta L_{gN1} & L_{gN2} & \dots & L_{gNN} \end{bmatrix}$$

While performing timing analysis, the various gate delays interact with each other, and it is essential that we keep track of the correlations between the delays of various gates. As a point in illustration, consider the sum operation along a long chain of inverters. Since the gate delays of these inverters are correlated, we need to keep track of the relationship between them to accurately predict the delay distribution at the output. In order to simplify this task, a commonly used technique is that of Principal Component Analysis (PCA), wherein a set of correlated Gaussian random variables (such as the gate length variations, ΔL_g) are linearly transformed into a set of independent and identically distributed (i.i.d.) unit normal random variables by an orthogonal transformation. Therefore, each of the N gate length parameter variations, ΔL_{gi}, can be expressed in terms of N (or fewer) i.i.d. Gaussian random variables p_i, as shown below.

$$\Delta L_{gi} = s_{i1} \cdot p_1 + s_{i2} \cdot p_2 + \ldots + s_{in} \cdot p_n \tag{2.2}$$

In the above equation, p_1, p_2, ..., p_n are i.i.d. unit normal Gaussian random variables, $\mathcal{N}(0, 1)$, and are called the principal components (PCs). The sensitivity of the variation in gate length, L_{gi} to p_j is given by s_{ij}.

Substituting the PCs back into the Taylor expansion of the gate delay d_i from Equation 2.1 gives what is referred to as a first order canonical delay model, shown below.

$$d_i = \mu_i + \sum_{j=1}^{j=n} s_{ij} \cdot p_j \tag{2.3}$$

In the above equation, s_{ij} represents the sensitivity of gate delay, d_i, to the jth principal component, p_j. To include the dependence of the gate delay on purely random variations we get the following expression below.

$$d_i = \mu_i + \sum_{j=1}^{j=n} s_{ij} \cdot p_j + \zeta_i \cdot r \tag{2.4}$$

The last term captures the independent parameter variations in the circuit (for example the oxide thickness, t_{ox}). A similar expansion is computed for the gate delay dependence on other process parameters of interest and for the interconnects in the circuit. It must be noted, that this technique requires that the process parameters be Gaussian in nature, which is a reasonable assumption for most parameter variations. The gate and interconnect delays represented as canonical forms in Equation 2.4 are therefore modeled as Gaussian or normal random variables. For more details, readers are referred to [15].

2.2.4 The Statistical Summation and Maximum Operations

As we described in Section 2.1, every block-based timing analysis tool uses two basic mathematical operations, the statistical summation, SUM_σ, and maximum, MAX_σ. With statistical gate and interconnect delays expressed in the canonical form of Equation 2.4, these need to be implemented carefully, making sure that correlations between the various elements of the circuit are retained to obtain a reasonable representation of the circuit output delay distribution. This section defines these operators which are used in state of the art statistical timing analysis tools.

2.2.4.1 The Statistical Summation Operation

Below we define the statistical summation operation used in our work. A key to simplifying statistical timing analysis is to maintain the invariant that that all statistical quantities, such as delays, arrival and required times and slacks, maintain their respective canonical forms. It will be apparent in the following discussion, as to how the Principal Component Analysis technique and the accompanying canonical representation in Equation 2.4 helps us simplify the operators.

Definition 2.2.1 (SUM_σ). The statistical summation, SUM_σ, of two normal random variables, e_i and e_j in canonical form is shown below.

$$SUM_\sigma(e_i, e_j) \quad = \quad \mu_i + \mu_j \quad + \quad \sum_{k=1}^{k=n} (s_{ik} + s_{jk}) \cdot p_j \quad + \quad \zeta_{ij} \cdot r \tag{2.5}$$

where, ζ_{ij} is given by $\zeta_{ij} = \sqrt{{\zeta_i}^2 + {\zeta_j}^2}$. The basic idea is to sum up the sensitivities of the individual random variables to their respective principal components.

2.2.4.2 The Statistical Maximum Operation

To obtain the statistical maximum MAX_σ, of two normal random variables e_i and e_j in canonical form is not so straightforward. This is because even though e_i and e_j are Gaussian, their statistical maximum, $MAX_\sigma(e_i, e_j)$ is not. However, as described above, to maintain the invariant that all statistical quantities are expressed in their canonical form, a commonly used technique (see the work in [15, 38, 60]) is to use a formulation by Clark [19], which computes the maximum of a set of correlated Gaussian random variables as another Gaussian random variable. We denote this linear statistical maximum approximation (the actual maximum operator is non-linear) by MAX_θ defined below.

Definition 2.2.2 (MAX_θ). The statistical maximum, MAX_θ, of two normal random variables, e_i and e_j, in canonical form (see Equation 2.4) using Clark's formulation [19], is

given by $c = MAX_\theta(e_i, e_j)$, where ϕ is the normalized Gaussian probability density function (pdf), $\mathcal{N}(0,1)$, θ is defined in Equation 2.8 and

$$\begin{aligned}
\mu_c &= \tau_{ij} \cdot \mu_i + \tau_{ji} \cdot \mu_j + \theta \cdot \phi\left(\tfrac{\mu_i - \mu_j}{\theta}\right) \\
\sigma_c^2 &= \tau_{ij} \cdot (\sigma_i^2 + \mu_i^2) + \tau_{ji} \cdot (\sigma_j^2 + \mu_j^2) \\
&\quad + (\mu_i + \mu_j) \cdot \theta \cdot \phi\left(\tfrac{\mu_i - \mu_j}{\theta}\right) - \mu_c^2 \\
s_{ck} &= \tau_{ij} \cdot s_{ik} + \tau_{ji} \cdot s_{jk} \\
\zeta_c &= \sqrt{(\tau_{ij} \cdot \zeta_i)^2 + (\tau_{ji} \cdot \zeta_j)^2}
\end{aligned} \tag{2.6}$$

In the above equations the mean and standard deviation of random variable e_i is given by μ_i and σ_i respectively. The correlation between variables e_i and e_j is given by ρ_{ij}. The mean and standard deviation of $c = MAX_\theta(e_i, e_j)$ is given by μ_c and σ_c respectively and s_{ck} is the weight of principal component, p_k, in the canonical representation of c. A weighting factor is applied to s_{ck} and the random component ζ_c to equate the variance of c to σ_c^2.

2.2.4.3 The Tightness Probability

One quantity deliberately left out from Equation 2.6 for the statistical maximum of two random variables e_i and e_j is τ_{ij}, since it deserves special attention. The factor τ_{ij} gives us a weighting factor for each of e_i and e_j in the linear approximation of the maximum operator MAX_θ. This is what is commonly referred to as the tightness probability [60] in the literature. It gives us the probability that random variable e_i takes on a value greater than random variable e_j, and is given below,

$$\tau_{ij} = \Phi\left(\tfrac{\mu_i - \mu_j}{\theta}\right) \tag{2.7}$$

where Φ is the cumulative distribution function (cdf) of a unit normal random variable, $\mathcal{N}(0,1)$ and

$$\theta = \sqrt{\sigma_i^2 + \sigma_j^2 - 2 \cdot \rho_{ij} \cdot \sigma_i \cdot \sigma_j} \tag{2.8}$$

The rest of the symbols have their meanings in Section2.2.4.2. Clearly, the complementary relation given below holds.

$$\begin{aligned} \tau_{ji} &= \Phi(\tfrac{\mu_j - \mu_i}{\theta}) \\ &= 1.0 - \tau_{ij} \end{aligned} \tag{2.9}$$

For the rest of the discussion it is assumed that the time taken to compute τ_{ij} is similar to that taken to compute $MAX_\theta(e_i, e_j)$.

Although the discussion in this section references generic random variables e_i and e_j, for the purposes of SSTA these can represent quantities such as the gate and interconnect delays, path delays, arrival times, required times and slack in the timing graph.

2.3 Statistical Criticality Computation

This section discusses our main contribution, that of computing the statistical criticality of gates in a digital circuit under process variations. Our technique works by linearly traversing the edges in the timing graph of a digital circuit and dividing it into "zones". We investigate the sources of error in using tightness probabilities for criticality computation with Clark's statistical maximum formulation. The errors are dealt with using a new clustering-based pruning algorithm which greatly reduces the size of circuit-level cutsets improving both accuracy and runtime over the current state of the art. On large benchmark circuits, our clustering algorithm gives about a $250X$ speedup compared to a pairwise pruning strategy with similar accuracy in results. Coupled with a localized sampling technique, errors are reduced to around 5% of Monte Carlo simulations with large speedups in runtime.

We first describe the problem of criticality computation in a digital circuit followed by a brief but important survey of the past works in this field. The rest of the chapter discusses our technique to compute the criticality of gates in a timing graph along with some results to back our approach.

2.3.1 Problem Description

We first begin by motivating the need to compute the criticality under process variations. Consider the circuit shown in Figure 2.4. Under process variations, as described in Section 2.2, each gate and interconnect in the circuit has a variable delay. As a result, the delay of the circuit is not deterministic but rather expressed as a probability distribution. Designers typically want to find the critical path (see Section 2.1.3) in a circuit and size the gates on this path to reduce the circuit delay and improve performance. However, with process variations, no one path dominates the delay of the circuit across all chips, since every path in the circuit has a certain probability of being critical. This idea not only applies to paths, but to individual circuit elements as well, for example gates in the circuit. As seen in Figure 2.4, each path marked in red can be critical depending on where the die lies in the region of probability space. In fact, theoretically, under process variations, every path in the circuit can be critical with a certain probability. The gates in the circuit are marked with a number between 0.0 and 1.0, denoting their **criticality probability**. This is the probability that the gate lies on a critical path, or in other words, it denotes the probability that a critical path in the circuit passes through the gate.

Definition 2.3.1 (Problem Definition). Given a digital circuit manufactured under an uncertain environment due to process variations, find the probability that a circuit element (gate or interconnect) lies on the critical path in the circuit. In other words, we need to compute the probability that a critical path in a manufactured die, passes through the circuit element. For a deterministic scenario, the critical path (assuming the circuit consists of only one such path) and all the elements that lie on it, has a criticality of 1.0. The other

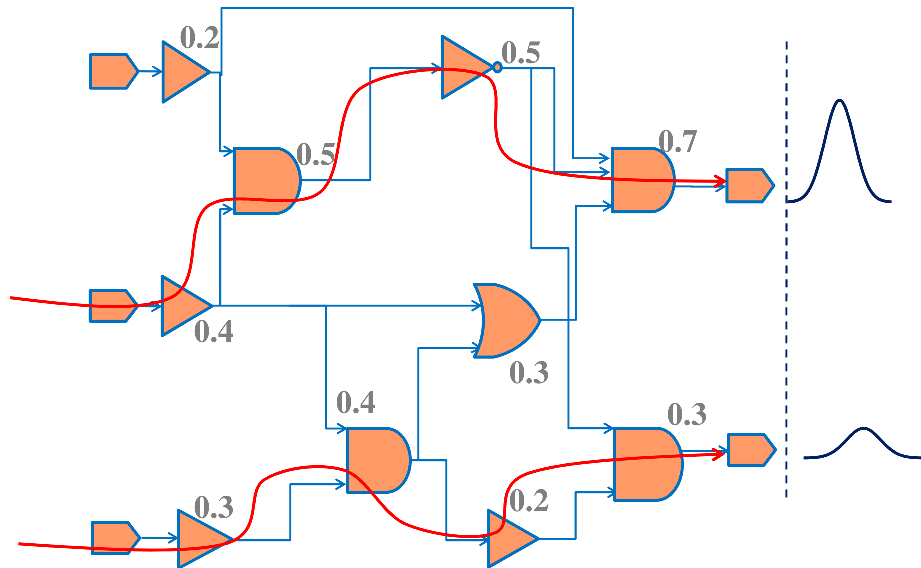


Figure 2.4: A simple circuit to illustrate the notion of criticality. The criticality of the gates is marked in gray. Representative delay distributions at the two output ports are also shown.

gates and interconnects in the circuit have a criticality of 0.0. As in previous works like [65] and [38], for the notion of probability to hold, we make the assumption that the probability of any two paths having the same delay in a circuit under process variations is zero.

The criticality probability of a gate can also be thought of as the sum of the criticality probabilities of all the paths passing through it. It is interesting to note that adding the criticalities of the gates that line up vertically produces 1.0. Briefly, this describes what is called a cutset, more formally defined in Section 2.3.3. By visually inspecting the vertically aligned gates, we note that all paths in the circuit pass through them. Therefore the summation of their criticality probabilities, which is the sum of the criticality probabilities of all paths in the circuit is 1.0.

The rest of this chapter describes our method to obtain these numbers. The notion of probability can help a circuit designer make informed decisions as to the relative importance of different gates in a design. In Section 2.3.2 we describe the previous approaches to com-

pute the criticality of gates and interconnects followed by definitions used throughout the rest of the chapter. In Sections 2.4.1-2.4.5 we describe our zone-based approach to compute the criticality using cutsets followed by the description of a fundamental drawback of using Clark's approximation for the statistical maximum operation in Section 2.4.6 and our clustering-based pruning approach which is used to overcome it, in Section 2.4.7. In 2.4.8 we discuss a few techniques to reduce the errors in our criticality compuation procedures to further improve accuracy. We finally present some results in Section 2.4.9 and conclude in Section 2.5.

2.3.2 Background

The authors in [60] propose the concept of a path criticality, which is the probability that a path in the manufactured chip is critical. This concept is also extended to edge (node) criticalities in the timing graph of a circuit, i.e., the probability that a path passing through the edge (node) is critical. To this end, works like [18, 38] and [65] attempt to compute the criticality probability of edges in a timing graph, using a canonical first order delay model.

The work in [60] is one of the earliest attempts to compute criticalities, wherein the authors perform a reverse traversal of the timing graph, multiplying criticality probabilities of nodes with local criticalities of edges. An incorrect assumption is the independence of edge criticalities despite structural and spatial correlations in the circuit. Subsequently, the work in [38] defined the statistical sensitivity matrix of an edge in the timing graph with respect to the circuit delay. The authors compute the sensitivity matrix of any edge with respect to the circuit output by using the chain rule in the backward propagation of the timing graph. Due to the matrix multiplications involved, the complexity of their approach, although linear in circuit size, could be potentially cubic in the number of principal components if the matrices are not sparse.

An alternative approach in [18] perturbs gate delays to compute their effects on the cir-

cuit output delay. The complexity of computation is reduced using the notion of a cutset belonging to a node in the timing graph. A cutset is a minimal set of edges, the removal of which divides the timing graph into two disconnected components. A key property of cutsets is that the statistical maximum of the sum of arrival and required times across all the edges of a cutset gives the circuit delay distribution. A gate sizing operation on the source node of a cutset affects only the arrival time of some of its edges. This is then used to incrementally update the circuit delay *pdf* which allows effective computation of the circuit yield gradient to gate sizing. Their approach however, is potentially quadratic in the size of the timing graph.

The cutset based idea is further extended in [65], wherein the authors propose an algorithm to compute the criticalities of edges by linearly traversing the timing graph. The criticality of an edge in a cutset is computed using a balanced binary partition tree. Edges recurring in multiple cutsets are kept track of in an array-based structure while performing the timing graph traversal. In [62] the authors compute critical regions of nodes and arcs in a timing graph which are regions in the process space where the node or arc lies on the critical path. Since the number of such regions is potentially exponential they use a pruning technique to reduce the time complexity.

We make the following contributions. First, similar to [65], we propose an algorithm to compute the criticality probability of edges (nodes) in a timing graph using the notion of cutsets. Edges crossing multiple cutsets are dealt with using a zone based approach, similar to [67], in which old computations are reused to the greatest possible extent. Second, we investigate the effect of independent random variations on criticality computation and devise a simple scheme to keep track of structural correlations due to such variations. Third, and more importantly, we examine the sources of error in criticality computations due to Clark's [19] formulation and propose a clustering based pruning algorithm to effectively eliminate a large number of non-competing edges in cutsets with several thousand edges.

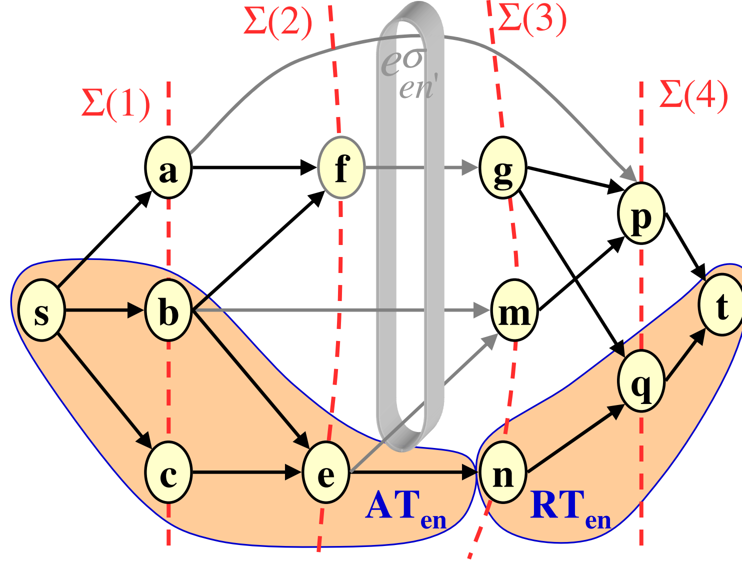


Figure 2.5: Example timing graph, G with cutsets, $\Sigma(1) - \Sigma(4)$. The shaded portions are nodes and edges used to compute the arrival and required time of edge e_{en}. The complementary path delay of e_{en} is the statistical maximum delay of all paths not passing through it, i.e., $e^{\sigma}_{en'} = MAX_{\sigma}(e^{\sigma}_{em}, e^{\sigma}_{bm}, e^{\sigma}_{fg}, e^{\sigma}_{ap})$.

The proposed clustering scheme can also help order the statistical MAX operations in a set, a source of significant error as shown in [56]. Localized sampling on the pruned cutset further reduces errors in edge criticalities to within 5% of Monte Carlo simulations, with large speedups in run-time when compared to a pairwise pruning strategy.

2.3.3 Definitions

This section defines the quantities used throughout the rest of this chapter. It must be noted that the definitions are in the context of statistical quantities. For the sake of clarity, if a particular deterministic quantity is defined as x, its statistical counterpart is denoted as x^{σ}. Figure 2.5 illustrates the timing graph of the circuit in Figure 2.4, which is used to explain some of the definitions.

Definition 2.3.2 (**Cutset**)**.** A cutset Σ is a set of edges/nodes in a timing graph G (see Definition 2.1.1), such that **every** path from the virtual source vertex, v_s, to the virtual sink vertex, v_t, passes through one and only one member of Σ.

As shown in Figure 2.5, the timing graph consists of four cutsets, denoted $\Sigma(1) - \Sigma(4)$. A single cutset can be composed of a combination of nodes (gates) and edges (connections between gate terminals). For example, cutset $\Sigma(2)$ consists of nodes, $\{n_f, n_e\}$, and edges, $\{e_{ap}, e_{bm}\}$. In our work, nodes are replaced by their fanout edges. Therefore the cutset $\Sigma(2) = \{e_{ap}, e_{fg}, e_{bm}, e_{em}, e_{en}\}$. Clearly, if every path in the circuit passes through a member of the cutset, the sum of the criticality probability of all members of a cutset equals 1.0. Herein lies the essence of many criticality computation algorithms, like [18] and [65]. Our criticality computation algorithm is also based on the notion of cutsets.

Definition 2.3.3 (Path Delay). The path delay of an edge/node, e_i in G, hereon, also referred to as edge/node delay, is denoted by e_i^σ and defined as the sum of its arrival and required times (see Definitions 2.1.2 and 2.1.3) and is a distribution with a pdf. Note that the arrival and required times are themselves distributions.

$$e_i^\sigma = AT_i^\sigma + RT_i^\sigma \tag{2.10}$$

In other words, the path delay of an edge/node represents the statistical maximum delay of all paths passing through it. Each path delay e_i^σ is represented in the canonical form of Equation 2.4.

Definition 2.3.4 (Complementary Path Delay). Given a cutset, Σ, in a timing graph, G, the complementary path delay, $e_{i'}^\sigma$, of an edge/node $e_i \in \Sigma \subset G$, is defined as

$$e_{i'}^\sigma = MAX_\sigma(e_j^\sigma, \forall e_j : e_j \in \Sigma, e_j \neq e_i) \tag{2.11}$$

where MAX_σ denotes the statistical maximum operator (see Section 2.2.4), which returns the statistical maximum of a set of random variables. As described in Section 2.2.4.2 we use Clark's maximum operation, MAX_θ to approximate the maximum. We consider $e_{i'}$ as a fictional edge with path delay $e_{i'}^\sigma$.

The complementary path delay of an edge e_i represents the statistical maximum delay of all paths not passing through it. All the paths in the timing graph from the source vertex v_s to the sink vertex v_t must pass through either e_i or the fictional edge $e_{i'}$ whose delay is $e^{\sigma}_{i'}$. Therefore $MAX_{\sigma}(e^{\sigma}_i, e^{\sigma}_{i'})$ represents the statistical circuit delay. Figure 2.5 shows the complementary path delay of edge e_{en} represented as $e^{\sigma}_{en'}$.

Definition 2.3.5 (Statistical Critical Path). As mentioned in Section 2.1.3, a critical path of a circuit implemented on a silicon die is the path which determines the maximum circuit delay. With process variations, different paths can be critical on different dies. Therefore, under a probabilistic scenario, every path of a circuit timing graph, G, has a certain probability of being the critical path. This probability is called the criticality probability of the path.

Definition 2.3.6 (Local Criticality). The local criticality, τ_{ij}, of edge e_i with respect to e_j is defined as the probability that at least one path of timing graph G passing through edge e_i takes on a value greater than or equal to any path passing through edge e_j, over all manufactured dies. The local criticality is given by the tightness probability in [60] described in Section 2.2.4.3 and computed as shown in Equation 2.7.

Local criticality τ_{ij} can be thought of as the **degree of domination** of edge e_i over e_j. It is easy to see that $\tau_{ji} = 1.0 - \tau_{ij}$.

Definition 2.3.7 (Global Criticality). The global criticality T_i (also referred to as criticality hereon) of edge e_i in cutset Σ is the probability that it has maximum delay among all the edges in the cutset, i.e.,

$$\begin{aligned} T_i &= \Pr(e^{\sigma}_i \geq e^{\sigma}_j) \quad \{\, \forall\, e_j \in \Sigma, \quad e_j \neq e_i \,\} \\ &= \Pr(e^{\sigma}_i \geq e^{\sigma}_{i'}) \quad \text{(see Def. 2.3.4)} \\ &= \tau_{ii'} \qquad\qquad\quad \text{(from Eq. 2.7)} \end{aligned} \tag{2.12}$$

In other words, T_i represents the probability that at least one path passing through e_i has a delay greater than any other path not passing through it, over all the manufactured dies. T_i is also referred to as the **criticality probability** of e_i.

Physically, the local criticality of an edge e_i in a cutset corresponds to a comparison of its path delay with respect to another edge of the cutset whereas the global criticality of an edge corresponds to a comparison of its path delay with respect to all other edges in the cutset. It follows that the global criticality of $e_i \in \Sigma$ cannot be greater than its local criticality with respect to any other edge in Σ, i.e.,

$$T_i \leq \tau_{ij} \quad \{\forall\, e_j \in \Sigma, \quad e_j \neq e_i\} \tag{2.13}$$

As mentioned before, the global criticality can guide a designer in the optimization of the circuit. Gates with high global criticality are candidates for up-sizing to improve the delay over all manufactured dies. The rest of this chapter describes our technique of computing global criticalities of gates and interconnects in the design using the notion of cutsets.

2.4 Fast and Accurate Criticality Computation under Process Variations

We mention at the outset that our idea to compute edge criticalities in a timing graph, $G(V, E)$, is similar to [65], in which the notion of cutsets is used. Although both algorithms asymptotically take time linear in the number of edges in G, we use a zone-based approach. Section 2.4.1 illustrates the cutset computation procedure, followed by outlining a simple statistical criticality algorithm, BSC, with runtime complexity quadratic in the number of edges in G, in Section 2.4.2. The runtime complexity is reduced in Section 2.4.3 using linear book-keeping data structures. The authors in [65] use a binary tree like data structure for this purpose. Sections 2.4.4 and 2.4.5 detail our zone-based approach, which reduces the

criticality computation runtime complexity to linear in the number of edges in G. In [65], an array based structure is used for this purpose. The primary advantage of our method is that the cutsets can be processed in any order, i.e., to compute the criticality of an edge in a cutset we need not compute the edge criticalities in any of its predecessor or successor cutsets.

2.4.1 Cutset Computation

This section describes the computation of cutsets on the timing graph illustrated in Figure 2.5. We topologically order the nodes in G from the virtual source, v_s, to virtual sink, v_t, node, followed by grouping them according to levels in G, such that nodes with a level lower than l are predecessors and nodes with a level greater than l are successors of nodes at level l, denoted $\Sigma_n(l)$. For instance, $\Sigma_n(2) = \{n_e, n_f\}$. Edges crossing l are denoted by $\Sigma_e(l)$ and these are called mc-edges.

Definition 2.4.1 (mc-edge). An mc-edge is an edge with end-level at least two greater than its start-level.

Thus, with our level enumerated cutsets, these are edges which cross over at least one cutset. In Fig. 2.5, e_{ap} and $e_{bm} \in \Sigma_e(2)$ are a set of mc-edges crossing level 2. However, edges like e_{fg} and e_{en} are not mc-edges since they start at level 2 and end at level 3 with no cross over. Consider the set of nodes and edges given by,

$$\Sigma(l) \quad = \quad \Sigma_n(l) \cup \Sigma_e(l) \tag{2.14}$$

By Definition 2.3.2, $\Sigma(l)$ forms a cutset since every v_s to v_t path in G must pass through at least one member of $\Sigma(l)$ and its elements are disjoint. Our aim is to compute the criticalities of all edges in G, using Definition 2.3.7. The topological level-enumerated cutsets are necessary and sufficient because they cover all the nodes and edges in G and no

Algorithm 1 BSC ($G(V, E)$) // G = circuit timing graph

1: Perform a forward and reverse SSTA on G
2: Topologically order G and find its cutsets Σ
3: **for all** cutsets $\Sigma \in G$ **do**
4:     **for all** edges $e_i \in \Sigma$ **do**
5:         $e^{\sigma}_{i'} = MAX_{\sigma}(e^{\sigma}_{j}, \ \forall e_j \in \Sigma, \ e_j \neq e_i)$ // see Definiiton 2.3.4
6:         $T_i = \tau_{ii'}$ // see Definition 2.3.7
7:     **end for**
8: **end for**

cutset is fully contained in another cutset. The number of such cutsets equals the number of levels L in G. To compute the criticality of all edges in G, we substitute nodes in Σ_n with their fanout edges. For instance, at level 2, we obtain the cutset of edges $\Sigma(2) = \{e_{ap}, e_{fg}, e_{bm}, e_{em}, e_{en}\}$.

2.4.2 BSC: Basic SC Algorithm

The simplistic approach called BSC, shown in Algorithm 1, computes global criticalities of all edges in timing graph G. Step 1 performs a forward and reverse SSTA to compute path delays (see Definition 2.3.3) of all edges (nodes) in G followed by topologically ordering G into levels to compute cutsets, Σ. Steps 3-8 compute the criticality of an edge in Σ by first computing its complementary path delay and then using Equation 2.12.

Due to the presence of mc-edges, each cutset, Σ, potentially contains $O(E)$ edges, where E is the number of edges in the timing graph. Moreover, since Step 5 computes the complementary path delay of an edge in time linear in the size of Σ, this step takes quadratic time, $O(E^2)$ over all edges in Σ. Over all cutsets in G, Algorithm 1 therefore has a complexity of $O(L \cdot E^2)$. Sections 2.4.3 and 2.4.4 discuss methods to reduce the time complexity of the basic approach.

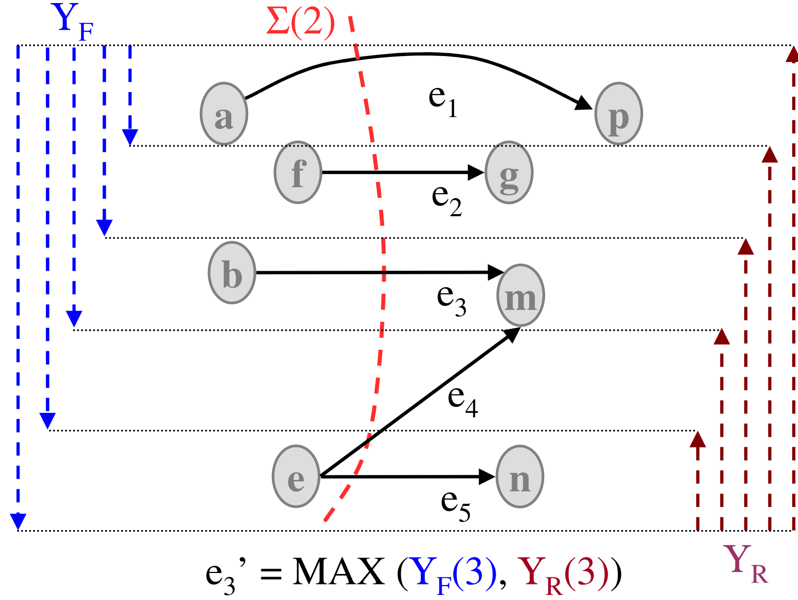


Figure 2.6: Illustration of forward and reverse linear book-keeping data structures, Υ_F and Υ_R, to compute the complementary path delay of e_3 in cutset, $\Sigma(2) = \{e_{ap}, e_{fg}, e_{bm}, e_{em}, e_{en}\}$, from Fig. 2.5, with edges relabeled e_1, e_2, e_3, e_4, e_5 respectively.

2.4.3 Linear Time Book-keeping

From the previous discussion, computing the complementary path delay of all edges in a cutset takes quadratic time. Fig. 2.6 shows a cutset, $\Sigma = \{e_1, e_2, \ldots, e_5\}$. To compute the complementary path delay of edges e_1 and e_2, we compute

$$\begin{aligned} e^{\sigma}_{1'} &= MAX_{\sigma}(e^{\sigma}_2, e^{\sigma}_3, e^{\sigma}_4, e^{\sigma}_5) \\ e^{\sigma}_{2'} &= MAX_{\sigma}(e^{\sigma}_1, e^{\sigma}_3, e^{\sigma}_4, e^{\sigma}_5) \end{aligned} \tag{2.15}$$

Clearly, $MAX_{\sigma}(e^{\sigma}_3, e^{\sigma}_4, e^{\sigma}_5)$ is a common term in Eq. 2.15 above. To speed up the computation of the complementary path delay of edges in a cutset, our book-keeping ordered lists aim to keep track of this common information.

Definition 2.4.2 (Ordered Lists). Given an arbitrary set $\Sigma = \{e_1, e_2, \ldots, e_n\}$ of n random variables, we define forward and reverse ordered lists, denoted Υ_F and Υ_R respectively, as

$$\Upsilon_F(i) = MAX_\sigma(e_1^\sigma, \ldots, e_i^\sigma) \tag{2.16}$$

$$\Upsilon_R(i) = MAX_\sigma(e_i^\sigma, \ldots, e_n^\sigma) \tag{2.17}$$

The global criticality of an edge $e_i \in \Sigma$ (Definition 2.3.7) can now be computed as

$$\begin{aligned} T_i &= \Pr\Big(e_i^\sigma \geq MAX_\sigma(e_1^\sigma, \ldots, e_{i-1}^\sigma, e_{i+1}^\sigma, \ldots, e_n^\sigma)\Big) \\ &= \Pr\Big(e_i^\sigma \geq MAX_\sigma(\Upsilon_F(i-1), \Upsilon_R(i+1)\Big) \end{aligned} \tag{2.18}$$

Computation of Υ_F and Υ_R takes $2n$ MAX_σ operations. Equation 2.18 takes two MAX_σ operations for a total of $4n$ MAX_σ operations over all edges in Σ. The ordered lists help compute criticalities of cutset edges in $O(n)$ time, i.e., linear in the size of the cutset. Figure 2.6 illustrates this computation for edge e_3.

2.4.4 Zone Computation

Using Υ_F and Υ_R, Steps 5-6 of Algorithm 1 now take $O(E)$ as compared to $O(E^2)$ time. Typical circuits however contain many mc-edges (Def. 2.4.1), as a result of which every cutset potentially has $O(E)$ edges. Over all L cutsets, we could take $O(L \cdot E)$ time, still a considerable slowdown.

To see why we can do better, the example timing graph G in Fig. 2.7, depicts mc-edges e_1, e_2, e_3, e_4, e_5 and e_6. Consider traversing G to compute the criticality of its edges using Algorithm 1. To compute the criticality of an edge, say e_x, at level 1, we compute its complementary path delay, $e_{x'}^\sigma$, which includes $MAX_\sigma(e_1^\sigma, e_2^\sigma)$. At level 2, for $e_{y'}^\sigma$, we compute $MAX_\sigma(e_1^\sigma, e_2^\sigma, e_3^\sigma, e_4^\sigma)$. Clearly, the only information needed at level 2 with regard to edges e_1 and e_2 is $MAX_\sigma(e_1^\sigma, e_2^\sigma)$, already computed at level 1. Algorithm 1 redundantly recomputes this information at level 2, thus accounting for the multiplicative L factor in the

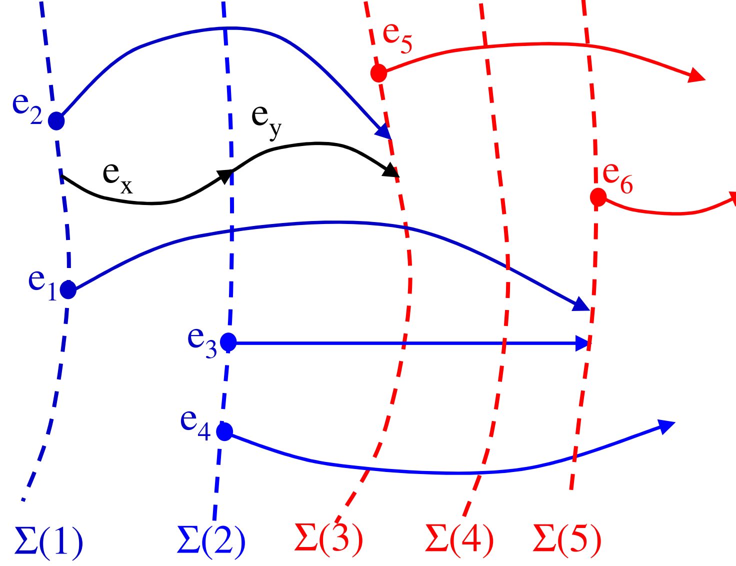


Figure 2.7: An example timing graph, G, with mc-edges, $\Sigma_e = \{e_1, e_2, \ldots, e_6\}$. Edges e_x and e_y are not mc-edges.

computation cost. The basic idea of zones is to abstract out the maximum of the mc-edges thereby reusing information to the greatest possible extent.

Let us reconsider traversing the timing graph in Fig. 2.7. At level 1 we would like to forward accumulate the maximum of mc-edges e_1 and e_2, used to compute the criticality of edges at higher levels. We thus enter an accumulation phase beginning at level 1, to obtain $Z_{1F} = \{e_1, e_2\}$ and $Z^{\sigma}_{1F}(1) = MAX_{\sigma}(e^{\sigma}_1, e^{\sigma}_2)$, useful at level 2 to find the maximum of the mc-edges crossing it (to compute $e^{\sigma}_{y'}$). At level 2 we accumulate edges e_3 and e_4 to obtain $Z_{1F} = \{e_1, e_2, e_3, e_4\}$ and $Z^{\sigma}_{1F}(2) = MAX_{\sigma}(Z^{\sigma}_{1F}(1), e^{\sigma}_3, e^{\sigma}_4)$. The indices of Z^{σ}_{1F}, denoted z_{1F}, are time points recording the order in which mc-edges accumulate in Z_{1F}. At level 3 however, we can no longer accumulate e_5 in Z_{1F}, since e_2 has reached its end level and does not contribute to the accumulated maximum in Z^{σ}_{1F}. Z_{1F} is thus a maximal set representing all edges accumulated in phase 1. Note that at this point, Z^{σ}_{1F} is not useful to us.

We now begin a reverse accumulation phase, to compute the order in which edges belonging to Z_{1F} leave timing graph, G. Once again, this is precomputed by a traversal of G as,

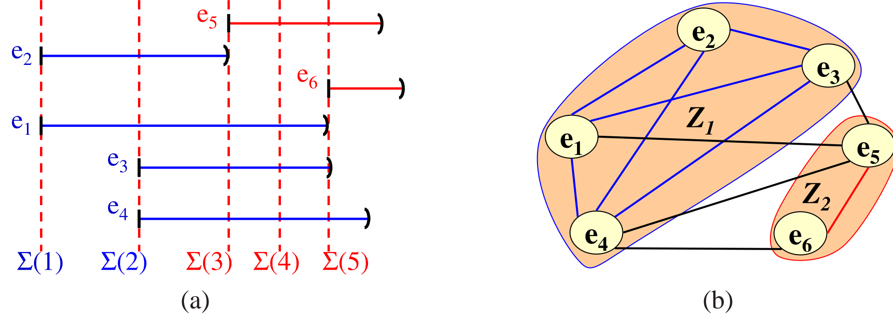


Figure 2.8: Fig. 2.8(a) shows the mc-edges, $\Sigma_e = \{e_1, \ldots, e_6\}$, of the timing graph in Fig. 2.7, represented as half-open intervals. Fig. 2.8(b) shows the corresponding interval graph representation, G_e, with zones, $Z_1 = \{e_1, e_2, e_3, e_4\}$ and $Z_2 = \{e_5, e_6\}$, identified as mutually exclusive maximal cliques.

$Z_{1R} = \{e_1, e_3, e_4\}$ and $Z^{\sigma}_{1R}(1) = MAX_{\sigma}(e^{\sigma}_1, e^{\sigma}_3, e^{\sigma}_4)$ at time point 1. Similarly at time point 2, $Z_{1R} = \{e_4\}$ and $Z^{\sigma}_{1R}(2) = e^{\sigma}_4$. The indices of Z^{σ}_{1R}, denoted z_{1R}, are time points that record the order in which mc-edges leave Z_{1R}. We concurrently start a new accumulation phase beginning with edge e_5 as $Z_{2F} = \{e_5\}$ and $Z^{\sigma}_{2F}(1) = e^{\sigma}_5$. The maximum of mc-edges crossing levels greater than 3 (for example at level 4), can now be computed using $MAX_{\sigma}(Z^{\sigma}_{1R}(1), Z^{\sigma}_{2F}(1))$.

Definition 2.4.3 (**Zone**)**.** A zone Z_i is a set of mc-edges with the end-level of any edge higher than the start-level of all edges in Z_i, i.e., edges enter a zone before any edge exits it.

From the above description, at a particular level, l, of the timing graph, the different mc-edges that cross it can be active in different zones. At level l, the contribution, Z^{σ}_{iMAX}, of mc-edges belonging to zone Z_i, is given respectively by the appropriately indexed entry of Z^{σ}_{iF} or Z^{σ}_{iR}, depending on the forward or reverse accumulation phase of the zone. The statistical maximum of mc-edges crossing level l, denoted Z^{σ}_{MAX}, is obtained by computing the maximum of the contributions of each zone, Z^{σ}_{iMAX}, over all the zones.

Formally, mc-edges represent half-open intervals, from their source to sink level, as in Fig. 2.8(a), denoted Σ_e, with the corresponding interval graph representation shown in

Fig. 2.8(b), denoted G_e. The interval graph is a one-to-one representation of intervals to vertices, with two vertices connected by an edge if and only if their corresponding intervals overlap [24]. In what follows, the term interval is used interchangeably with edge.

By Definition 2.4.3, a zone is any set of overlapping intervals. In the interval graph representation, a zone is a clique (not necessarily maximal). Hence, like [67], we aim to compute the cliques in the interval graph. Since an mc-edge belongs to a single zone, the cliques must be mutually exclusive. Fig. 2.8(b) shows one set of mutually exclusive cliques which forms the zones. We begin with edge e_2 and greedily compute the maximal clique $\{e_1, e_2, e_3, e_4\}$ to form zone Z_1. Next, with e_5 we get zone Z_2 with maximal clique $\{e_5, e_6\}$.

In the worst case, the number of zones, K, in a timing graph with L levels is $O(L)$. The idea is to minimize K so as to reduce the number of MAX_σ operations over all zones to compute Z^{σ}_{MAX}. The Algorithm described in [28] computes a minimum clique covering of an interval graph, G_e. A simplicial vertex of G_e is defined as follows.

Definition 2.4.4 (Simplicial edge). A simplicial vertex, v^s, of an interval graph, G_e, is a vertex, all of whose neighbors form a clique with v^s [24]. The interval e^s in the corresponding interval representation, Σ_e, of G_e is called a simplicial edge.

It is easy to verify that an interval with minimum end-level is a simplicial edge in Σ_e. In fact, the neighbors of v^s form a clique which is maximal. In Figure 2.8, e_2 is a simplicial edge.

Algorithm 2, like [28], finds a minimum size clique covering of the interval graph, G_e, by repeatedly finding a simplicial edge in Σ_e, and removing all the edges overlapping it, i.e., it repeatedly computes mutually exclusive maximal cliques in G_e. These cliques form the zones in our criticality computation algorithm. However, unlike [28], a separate step to sort the intervals according to their end points is not needed, because of the topological ordering of the timing graph described in Section 2.4.1. For the example interval representation in

Algorithm 2 Z = ComputeZones (Σ_e)
// Σ_e = mc-edges in the timing graph organized as per levels
// Z = list of mutually exclusive zones, $\{Z_1, \ldots, Z_K\}$

1: $K = 1$, $Z = \{\ \}$ // initialize the list of zones
2: $Z_K = \left\{Z_{KF} = \{\ \}, Z_{KR} = \{\ \}\right\}, z_{KF} = z_{KR} = 0$
   // Z_{KF} (Z_{KR}) records the history of edges entering
   // (leaving) zone Z_K, indexed by pointer z_{KF} (z_{KR})
3: **for all** levels $l \in G$ **do**
4:   **for all** $e_j \in \Sigma_e$ with end level l **do**
5:     Z_j = zone of e_j
6:     Insert e_j into Z_{jR}, ++z_{jR} // e_j exits zone Z_j
7:     **if** $Z_j == Z_K$ **then** // $e_j^s = e_j$ is the first to exit Z_j
8:       ++K // create new zone
9:       $Z_K = \left\{Z_{KF} = \{\}, Z_{KR} = \{\}\right\}, z_{KF} = z_{KR} = 0$
10:       Insert Z_K into Z
11:     **end if**
12:   **end for**
13:   **for all** $e_i \in \Sigma_e$ with start level l **do**
14:     Set zone of e_i to Z_K
15:     Insert e_i into Z_{KF}, ++z_{KF} // e_i enters zone Z_i
16:   **end for**
17: **end for**
   // Compute book-keeping lists for all zones
18: **for all** $Z_i \in Z$ **do**
19:   Compute Υ_{iF} (Υ_{iR}) for Z_{iF} (Z_{iR}) // Equation 2.16 (2.17)
20: **end for**
21: **return** Z

Fig. 2.8, we compute $e_2 \in \Sigma_e$ as the first simplicial edge, with zone $Z_1 = \{e_1, e_2, e_3, e_4\}$, followed by $e_5 \in \{\Sigma_e - Z_1\}$, as the second simplicial edge, with zone $Z_2 = \{e_5, e_6\}$.

Algorithm 2 computes zones linearly traversing timing graph, G, from the virtual source to virtual sink node, identifying mc-edges and reporting the mutually exclusive maximal cliques by keeping track of when an edge enters (Steps 13-16) and leaves (Steps 4-12) G. The zones are computed as $Z = \{Z_1 \cup Z_2 \cup \ldots \cup Z_K\}$, and the claim is that K is the minimum number of zones (cliques) needed to cover Σ_e (the interval graph, G_e). The following property proves this claim.

Property 2.4.1. In Algorithm 2, the first edge, e_j^s, to exit its zone, Z_j, is a simplicial edge of the intervals corresponding to $\Sigma_e - \{Z_1 \cup Z_2 \cup \ldots \cup Z_{j-1}\}$.

Proof. Step 7 computes the first edge, e_j^s, to exit its zone, Z_j. Edge e_j^s has minimum end-level, l_j, in $\Sigma_e - \{Z_1 \cup Z_2 \cup \ldots \cup Z_{j-1}\}$ and therefore is a simplicial edge in $\Sigma_e - \{Z_1 \cup Z_2 \cup \ldots \cup Z_{j-1}\}$. If this was not the case, consider another edge, e_n, with end-level, $l_n < l_j$, $e_n \notin \{Z_1 \cup Z_2 \cup \ldots \cup Z_{j-1}\}$. Clearly, $e_n \notin Z_k$, $k > j$, because edges are assigned to zones in sequence implying its start-level (and thereby its end-level, l_n) must be greater than l_j. Therefore $e_n \in Z_j$, which is again a contradiction since it implies $l_n \geq l_j$.

□

Using Property 2.4.1, Algorithm 2 like [28] repeatedly finds simplicial edges in Σ_e to compute a minimum size clique cover of its corresponding interval graph representation, G_e.

Lists Z_{iF} and Z_{iR} record the history of mc-edges entering and exiting zone Z_i, indexed by pointers z_{iF} and z_{iR} respectively (Steps 15 and 6). For the example in Fig. 2.8(a), the lists for zones Z_1 and Z_2 are computed as,

$$
\begin{aligned}
Z_{1F} &= \{e_1, e_2, e_3, e_4\} \quad Z_{2F} = \{e_5, e_6\} \\
Z_{1R} &= \{e_4, e_3, e_1, e_2\} \quad Z_{2R} = \{e_6, e_5\}
\end{aligned}
\tag{2.19}
$$

For each zone Z_i, Step 19 computes the forward (Υ_{iF}) and reverse (Υ_{iR}) book-keeping lists, for Z_{iF} and Z_{iR} respectively.

In terms of computational complexity, Steps 3-17 of Algorithm 2 process each edge in Σ_e twice, first at its start-level and then at its end-level. Step 19 computes the forward and reverse book-keeping lists for the mutually exclusive zones in $O(|Z_1|) + O(|Z_2|) + \ldots + O(|Z_K| = O(|\Sigma_e|)$ time, where $|Z_i|$ is the number of edges in each zone and K is the minimum number of zones to cover the mc-edge interval representation, Σ_e, of the timing graph. Overall, the runtime of Algorithm 2 is $O(|\Sigma_e|)$, i.e., linear in the number of mc-edges.

It should be noted that as shown in [28], this approach is optimal, i.e., the lower bound on the computational complexity of computing a minimum clique cover is $\Omega(|\Sigma_e| \log |\Sigma_e|)$, if the mc-edges are not sorted by their end points.

2.4.5 ZSC: Zone Based SC Algorithm

Our zone-based criticality computation technique, ZSC, is shown in Algorithm 3. Step 4 computes zones in timing graph G, in time linear in the size of Σ_e, the set of mc-edges. We then forward traverse G from v_s to v_t. Steps 6-13 update the forward and reverse history pointers of each zone, to compute the contribution, Z^{σ}_{iMAX}, of the mc-edges belonging to zone, Z_i, in constant time. Steps 14-17 compute Z_{MAX}, a fictional edge representing the statistical maximum, Z^{σ}_{MAX} of mc-edges crossing a particular level, depending on the contribution, Z^{σ}_{iMAX}, to each zone Z_i. Since we have on the order of $O(L)$ number of zones, over all levels of G, this step takes $O(L^2)$ time. Finally, using the book-keeping ordered lists from Section 2.4.3, we compute global criticalities of edges in cutset Σ, in

Algorithm 3 ZSC ($G(V, E)$) // G = circuit timing graph

1: Perform a forward and reverse SSTA on G
2: Topologically order G and find its cutsets Σ
3: Compute Σ_e, the set of mc-edges
4: Z = ComputeZones(Σ_e) // see Section 2.4.1
5: **for all** levels $l \in G$ **do**
6:     **for all** $e_j \in \Sigma_e$ with end-level l **do**
7:         ++z_{jR} // e_j exits zone Z_j, update reverse pointer
8:         $Z_{jMAX} = \Upsilon_{jR}(z_{jR})$
9:     **end for**
10:     **for all** $e_i \in \Sigma_e$ with start-level l **do**
11:         ++z_{iF} // e_i enters zone Z_i, update forward pointer
12:         $Z_{iMAX} = \Upsilon_{iF}(z_{iF})$
13:     **end for**
14:     $Z^{\sigma}_{MAX} = -\infty$ // maximum over all active zones
15:     **for all** $Z_k \in Z$ **do**
16:         $Z^{\sigma}_{MAX} = MAX_{\sigma}(Z^{\sigma}_{MAX}, Z^{\sigma}_{kMAX})$
17:     **end for**
18:     Create fictional edge Z_{MAX} with path delay Z^{σ}_{MAX}
19:     $\Sigma = \{$Fanout edges of the nodes in $\Sigma_n(l)\} \cup Z_{MAX}$
        // see Section 2.4.1
20:     Create Υ_F and Υ_R for Σ // see Section 2.4.3
21:     Compute $T_i \;\forall\; e_i \in \Sigma$ // see Definition 2.3.7
22: **end for**

time linear in the number of edges in Σ. The overall runtime of the ZSC algorithm is therefore $O(E + L^2)$, which for a reasonably sized practical circuit is $O(E)$.

In summary, like [65], the zone-based approach computes the criticality of edges in timing graph $G(V, E)$, with a linear runtime complexity $O(E)$. Although both algorithms asymptotically take $O(L)$ time to compute the MAX_{σ} of mc-edges crossing a level, Algorithm 2 computes a minimum clique cover and helps to reduce the total number of MAX_{σ} operations computed in Steps 14-17 over all the cutsets. More importantly, our algorithm can compute the criticality of edges in a cutset, independently of other cutsets.

Table 2.1: Comparison of Monte Carlo and MAX_θ for the *abc* Problem

Method	T_a	T_b	T_c
MC	0.923	0.000	0.077
Clark	0.356	0.297	0.079
% Error δ	56.7%	29.7%	0.2%

2.4.6 Errors in ZSC

We ran Algorithm 3 on a subset of the ISCAS89 benchmarks to compute the global criticalities of all edges in the timing graph, G. We compared our implementation with a Monte Carlo (MC) simulation of 10000 samples and noted the absolute maximum difference in the criticalities of edges (denoted δ hereon). The difference was larger than 50% (for example, an edge reported by MC as 80% critical was reported by Algorithm 3 as 30% critical). In the following sections, we illustrate the sources of these errors with 3 random variables in a simple example we call the *abc* problem.

2.4.6.1 The abc Problem

As an illustration of these errors, consider a cutset Σ with random variables a, b and c, each with independent principal components (PCs) p_1 and p_2 (where p_i is a unit normal Gaussian, $\mathcal{N}(0,1)$), shown below,

$$\begin{aligned} a &= 4.000 + 0.5000 \cdot p_1 + 0.5000 \cdot p_2 \\ b &= 3.999 + 0.4999 \cdot p_1 + 0.5001 \cdot p_2 \\ c &= 3.800 + 0.6001 \cdot p_1 + 0.3999 \cdot p_2 \end{aligned} \tag{2.20}$$

It can be observed that a and b are nearly identical highly correlated random variables, and for any sample value of the $p_i's$, $a \geq b$ (high correlation coupled with the difference in means ensures that $\Pr(b \geq a) \approx 0.0$, or in other words a completely dominates b).

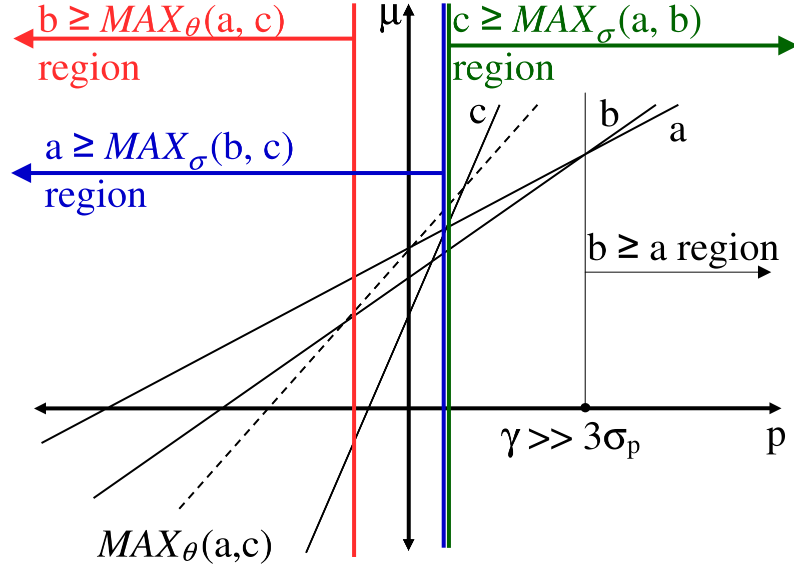


Figure 2.9: A pictorial depiction (not to scale) of the *abc* example with random variables a, b and c with one PC, p.

We ran a MC simulation with 100000 samples to determine the global criticalities of a, b and c. Table 2.1 shows a comparison with Clark's formulation, MAX_θ (see Def 2.2.2). The columns T_i, $i \in \{a, b, c\}$, depict the global criticality of variable i. As seen in the last row of Table 2.1, errors of 57% in the global criticality of a and 30% in that of b were observed.

2.4.6.2 Local and Global Errors

For a better illustration of the *abc* problem, Fig. 2.9 depicts the scenario of Equation 2.20, using just one principal component (PC), p. We make the following observations.

1. The local criticality of b with respect to a, i.e., $\tau_{ba} \approx 0$. This is indicated by a large value of $\gamma \gg 3\sigma_p$ (the region where $b \geq a$). Moreover, Clark's tightness probability formulation from Equation 2.7 also gives $\tau_{ba} \approx 0$.

2. Global criticality of b, $T_b \approx 0$. This is evident in Figure 2.9 where regions $a \geq MAX_\theta(b, c)$ and $c \geq MAX_\theta(a, b)$ cover the entire probability space.

Observations 1 and 2 are consistent with Equation 2.13. Now consider computing the global criticality of b, using the cutset approach. We first compute its complementary path delay $b' = MAX_{\sigma}(a, c)$. It follows from Def. 2.3.7 that $T_b = \Pr(b \geq MAX_{\sigma}(a, c))$. With Clark's formulation MAX_{θ}, for the statistical MAX_{σ}, we get $T_b = \Pr(b \geq MAX_{\theta}(a, c)) = 0.297$.

Intuitively, for this scenario, Clark's formulation is accurate with respect to local criticality t_{ba} of b, but it overestimates its global criticality T_b, and is inconsistent with Eq. 2.13.

Definition 2.4.5 (**Local Errors**). With respect to Clark's formulation, edge e_i in cutset Σ is said to have **local errors** iff there exists some edge $e_j \in \Sigma$ with respect to which its local criticality is less than its global criticality, i.e.,

$$\{\,\exists\, e_j \;\in\; \Sigma, \;\; e_j \neq e_i\,\} \;\; : \;\; \tau_{ij} \;\; < \;\; T_i \tag{2.21}$$

In other words, Equation 2.13 does not hold. The difference, $T_i - \tau_{ij}$, gives the numerical value of the **local error** of e_i with respect to e_j. By definition, local errors always overestimate the criticality of an edge in Σ. In our toy example, b exhibits local errors of magnitude 0.297, with respect to a. Local errors were found to propagate in the ZSC algorithm, where variables (edges) like b that should not have been critical, were found to have a significant criticality.

It must be pointed out that as was shown in [56], the order of variables plays an important role due to Clark's MAX_{θ} approximation. For the *abc* problem however, ordering variables (a and c) in the MAX_{θ} operation will not eliminate local errors in b. Local errors are an artifact of the manner in which we compute global criticalities.

Local errors only present a part of the picture with respect to the overall errors seen in criticality computation. This is because of the inherent inconsistencies in using Clark's formulation, MAX_{θ}, to approximate the maximum of a set of Gaussian random variables

as another Gaussian. Recall that Clark's method approximates the statistical maximum as a weighted linear summation. The works in [56] and [68] for instance, give a detailed analysis of the errors involved in such an approximation.

Definition 2.4.6 (Global Errors). With respect to Clark's formulation, edge e_i in a cutset Σ is said to have **global errors**, iff its computed criticality T_i differs from its true criticality and the edge does not exhibit local errors, i.e.,

$$T_i \leq \tau_{ij} \{ \forall e_j \in \Sigma, e_j \neq e_i \} \text{ and } T_i \text{ in error.} \tag{2.22}$$

Global errors cause erroneous values of the global criticality of an edge, e_i, in a cutset, due to the inaccuracies in the computation of its complementary path delay, $e^{\sigma}_{i'}$, using Clark's approximation. For the *abc* example, T_a is underestimated by 0.567. Note that the value of T_a is consistent with Equation 2.13, since both $\tau_{ab} = 1.0$ and $\tau_{ac} = 0.921$ are greater than $T_a = 0.356$. It is not a straightforward task to determine if an edge e_i of cutset Σ is in global error. However, we give a technique in Section 2.4.8.1 to overcome global errors that may arise in a cutset.

Two observations motivate the need for our pruning based criticality algorithm, described in Section 2.4.7. First, with respect to local errors in variable b, if we choose to "ignore" variable c and compute the criticality of b directly with respect to a, we get, $T_b = \tau_{ba} = 0.0$, a better result, since a almost completely dominates b. Second, with respect to global errors in variable a, if we choose to "ignore" variable b (due to its high dominance by variable a) and compute the criticality of a directly with respect to c, we get, $T_a = \tau_{ac} = 0.921$, a better result, since the computation of $MAX_{\theta}(b, c)$ (and hence the inaccuracy involved in it) is avoided.

In summary, although local and global errors result from Clark's MAX_{θ} linear approximation, local errors are an artifact of the manner in which we compute global criticalities

of edges in a cutset whereas global errors are more fundamental, arising due to the inherent approximation of MAX_θ. The next section addresses local errors in the criticality computation of a cutset of edges using a clustering based pruning scheme.

2.4.7 Clustering Based Statistical Criticality Computation

We begin this section by defining edges with respect to their relation in a set Σ. Although these definitions hold for any set of edges, for the purpose of this work, sets in general refer to cutsets of a timing graph.

Definition 2.4.7 (**Dominant Edge**). An edge, e_i, in set, Σ, is **dominant** iff its local criticality with respect to all other edges in Σ is above a threshold ε, i.e.,

$$\tau_{ij} \;\; > \;\; \varepsilon \;\; \{\, \forall\, e_j \,\in\, \Sigma, \;\;\; e_j \neq e_i \,\} \tag{2.23}$$

Definition 2.4.8 (**Non-dominant Edge**). An edge e_i in Σ is said to be **non-dominant** iff there exists some edge e_j for which its local criticality is below a threshold ε, i.e.,

$$\{\, \exists\, e_j \,\in\, \Sigma, \;\;\; e_j \neq e_i \,\} \;\; : \;\; \tau_{ij} \;\; \leq \;\; \varepsilon \tag{2.24}$$

In this case edge e_j is said to **dominate** edge e_i. An important property of non-dominant edges is that if e_i is non dominant in Σ with respect to e_j, then e_i is also non-dominant with respect to the statistical maximum of any combination of edges in Σ including edge e_j. This follows directly from the definition of a maximum operator. This can also be described by the equation given below.

$$\begin{aligned} & e_i \;\; <^\sigma \;\; e_j \\ \Rightarrow \;\; & e_i \;\; <^\sigma \;\; MAX_\sigma(e_j, \{e_{j1} \cup e_{j2} \ldots \cup e_{jn}\}) \end{aligned} \tag{2.25}$$

where $e_{j1} \in \Sigma$. In the above equation $<^{\sigma}$ denotes a statistical dominator operator. In other words, if non-dominant edge $e_i <^{\sigma} e_j$, i.e., e_i is dominated by some other edge e_j in cutset Σ, then it is dominated by $MAX_{\sigma}(e_i^{\sigma}, e_k^{\sigma})$, where e_k is another edge in the cutset.

Definition 2.4.9 (Mutually-dominant Edges). A set, Σ, of edges are said to be **mutually dominant** iff each edge in Σ is dominant, i.e.,

$$\tau_{ij} \;\; > \;\; \varepsilon \;\; \{ \forall \, e_i, e_j \; \in \; \Sigma, \;\; e_j \neq e_i \, \} \tag{2.26}$$

As seen in the previous section, non-dominant edges (like b in Figure 2.9) in a cutset exhibit local errors. Moreover, they also contribute to global errors of other edges in the cutset (like a in Figure 2.9). To avoid the bulk of these errors, we propose to prune the cutset, eliminating its non-dominant edges from injecting errors in the computation of global criticality.

Pruning is justified by Equation 2.13, wherein eliminating edge e_i with local criticality lower than a sufficiently small threshold value ε does not hurt global criticality computations because $T_i \leq \varepsilon$. The benefits are accentuated in cutsets with dominant edges that have large global criticalities, since the sum of global criticalities across a cutset must equal 1.0 (implying that many edges have very small local criticalities).

However, not every edge with global criticality below ε can be eliminated by pruning, particularly if its local criticality is greater than ε. Such edges cause global errors in the cutset.

2.4.7.1 $^{n}C_2$ Cutset Pruning

A straightforward approach to prune a cutset would be to perform a pairwise comparison of edges, eliminating those that have a local criticality less than a predefined threshold ε. The main drawback of this approach is its quadratic runtime complexity of $O(n^2)$, due to $^{n}C_2$

Algorithm 4 nC2 (Σ, ε)
// Σ = cutset of edges; ε = pruning threshold

```
1: for all i ∈ Σ do
2:    for all j = i + 1 ∈ Σ do
3:       if τ_ij < ε then // see Definition 2.3.6
4:          Mark e_i = pruned
5:       else if τ_ji < ε then
6:          Mark e_j = pruned
7:       end if
8:    end for
9: end for
10: Remove all pruned objects from Σ
```

local criticality computations, where n is the number of edges in the cutset. Clearly, this can be prohibitive even for moderately large circuits with cutsets on the order of several thousand edges. We show this procedure in Algorithm 4 which prunes a cutset Σ with respect to pruning threshold ε.

2.4.7.2 Clustering Based Cutset Pruning and Ordering

To overcome the quadratic runtime complexity overhead of the aforementioned ${}^{n}C_2$ approach, we present a new clustering based pruning technique which uses the K-center clustering algorithm of [26], commonly used in image quantization applications [64].

The basic idea is to prune the non-dominating edges from the cutset to return a set of mutually dominant edges. Throughout the execution of the algorithm, a dominant edge, selected from the current set of edges, Σ, is used to prune out non-dominant edges from Σ. Clustering facilitates the selection of dominant edges. The variables used in the algorithm are:

σ: A cluster containing at least one object.

κ: Each cluster σ contains a center, κ.

$d_{i\kappa}$: Distance of an object, i, from its cluster center, κ, is its local criticality, $\tau_{i\kappa}$, with

Algorithm 5 K = KCenterPrune (Σ, ε, S)
// Σ = cutset of edges; ε = pruning threshold;
// S = maximum cluster size; K = # clusters

1: $\Omega = \{ \}$ // set of clusters
2: $\sigma = \{ \}$ // initialize the 1st cluster
3: $K = 0$ // total number of clusters present in Ω
4: seed $\chi \in \Sigma$ = object (or edge) with maximum mean μ
5: Insert χ as the center of cluster σ
6: **for all** $i \in \Sigma$ **do**
7: **if** $\tau_{i\chi} > \varepsilon$ **then** // see Definition 2.3.6
8: Insert i in σ // object i not dominated by χ
9: **end if**
10: **if** $\tau_{\chi i} \leq \varepsilon$ **then**
11: Mark χ = pruned // object χ dominated by i
12: **end if**
13: **end for**
14: Compute radius, r_σ and distal element, R_σ of σ
15: Insert cluster σ into Ω; $++K$
16: **while** (maximum size of a cluster in $\Omega > S$) **do**
17: σ = CreateNewCluster (Ω)
18: Insert new cluster σ into Ω; $++K$
19: **end while**
20: Insert all un-pruned objects of Ω in Σ
21: **return** K

respect to κ.

r_σ: Radius of cluster σ, is the distance of the object farthest from center κ, i.e., $r_\sigma = max(d_{i\kappa})\ \forall i \in \sigma$.

R_σ: A distal object of cluster σ is an object with maximal distance from κ, i.e., $R_\sigma = j\ :\ d_{j\kappa} = r_\sigma$. In case of multiple distal elements we choose one arbitrarily.

Algorithm 5 describes the procedure. We first choose the seed χ, as the object with maximum mean in cutset, Σ (Steps 4-5). Next, Steps 7-9 prune Σ with respect to seed, χ, also marking χ as pruned if its local criticality with respect to any other object in Σ is less than ε (Steps 10-12). Steps 16-19 iteratively compute new clusters from existing ones (Algorithm 6) until no cluster has size exceeding S. Step 21 returns the remaining un-pruned

---

Algorithm 6 σ = CreateNewCluster (Ω)
// Ω = set of clusters; σ = new cluster

1: $\sigma = \{\ \}$ // initialize new cluster σ
2: m = cluster with maximum radius in Ω
3: $\chi = R_m$ // distal element of cluster m
4: Insert χ as center of newly created cluster σ
   // Prune Ω with respect to χ
5: **for all** $j \in \Omega, j \neq \kappa, \kappa =$ center of a cluster in Ω **do**
6:   **if** $\tau_{j\chi} < \varepsilon$ **then** // χ dominates j (see Definition 2.3.6)
7:     Delete j from Ω // prune j
8:   **else if** $\tau_{j\chi} < d_{j\kappa}$ **then** // χ dominates j more than κ
9:     Remove j from its current cluster, insert j in σ
10:   **end if**
11: **end for**
12: **if** $\exists\, j \in \Omega\ :\ (1 - \tau_{j\chi}) \leq \varepsilon$ **then** // j dominates χ
13:   Mark χ = pruned
14: **end if**
15: Compute r_σ and R_σ for σ and all existing clusters in Ω
16: **return** σ

---

objects in Σ.

In Algorithm 6, the distal element, χ, of the cluster, m, with maximum radius is chosen as the center of a newly created cluster, σ (Steps 1-4). Intuitively, χ is the object upon which its center has the lowest degree of domination (Definition 2.3.6) and hence a good candidate to facilitate the pruning of other edges in the cutset. Therefore it is chosen as the center of the new cluster. Step 7 uses χ to prune objects j (with local criticality with respect to χ less than ε) from their respective clusters. If χ has a higher degree of domination over j compared to its current center κ, j is removed from its current cluster and inserted into new cluster σ (Steps 8-10). Intuitively, a greater degree of domination between two edges results in smaller global errors in MAX_θ. If the newly added cluster center χ, is dominated, it is marked pruned (Step 13). We return the newly created cluster σ after adjusting the radius and the distal element of all currently existing clusters in Ω (Steps 15-16).

Fig. 2.10 illustrates the execution of Algorithm 5 on a cutset of 9 objects labeled a-i with pruning threshold $\varepsilon = 0.05$, taken from one of the ISCAS89 benchmarks (s9234). The

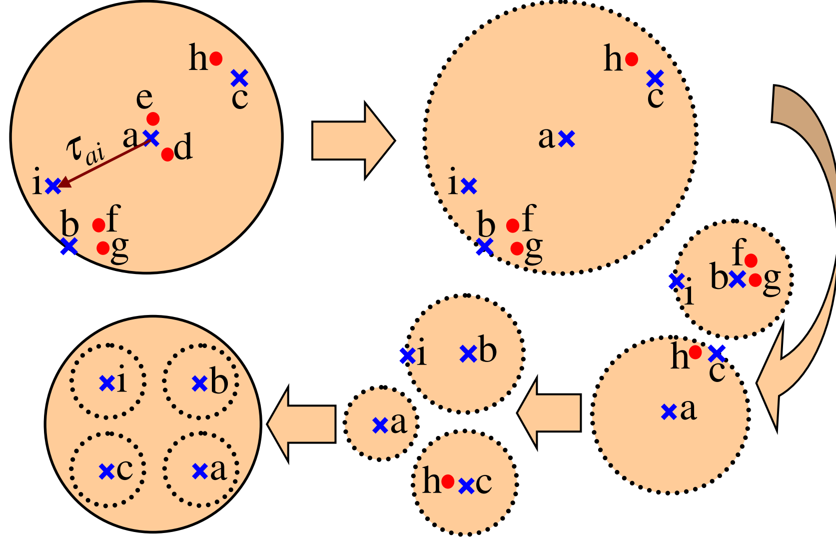


Figure 2.10: Illustration of the clustering based pruning procedure of Algorithm 5. Crosses indicate dominant objects and dots indicate non-dominant objects. The clustering distance is the local criticality (τ_{ai}) of an edge (i) from its cluster center (a).

relevant local criticalities of the objects are, $\tau_{ba} = 0.19$, $\tau_{ca} = 0.18$, $\tau_{da} = 0.01$, $\tau_{ea} = 0.0$, $\tau_{fa} = 0.17$, $\tau_{ga} = 0.17$, $\tau_{ha} = 0.17$, $\tau_{ia} = 0.17$, $\tau_{fb} = 0.02$, $\tau_{gb} = 0.02$, $\tau_{ib} = 0.06$ and $\tau_{hc} = 0.03$. Initially, a is chosen as the center of the 1st cluster, pruning out objects d and e. Next, b, a distal element of cluster 1 becomes the center of cluster 2, pruning out objects f and g. Also, since $\tau_{ib} < \tau_{ia}$, object i is absorbed into cluster 2. Next c, the distal element of cluster 1, the cluster with maximum radius, is chosen as the center of cluster 3, pruning object h. Finally, object i becomes the center of cluster 4 and the algorithm returns mutually dominant objects a, b, c and i. The algorithm has the following properties.

Property 2.4.2. At any iteration, all objects in Ω (excluding cluster centers marked pruned) are dominant with respect to all existing cluster centers.

Proof. To avoid being pruned, objects must be dominant with respect to seed χ, which is also the center of the 1st cluster (Step 8 of Algorithm 5). Moreover, every object j is compared with all newly added cluster centers in Line 7 of Algorithm 6. Clearly, any object j must be dominant with respect to these centers to avoid being pruned. Moreover,

Lines 11 of Algorithm 5 and 13 of Algorithm 6 compare every cluster center with every object for dominance. Although not immediately removed from Ω, centers are marked pruned if they are non-dominant with respect to other cluster objects.

□

Property 2.4.3. With $S = 1$, KCenterPrune(Σ, ε, 1) returns a set of mutually dominant edges (see Definition2.4.9) in Σ.

Proof. When $S = 1$, each cluster in Ω contains only one object, its cluster center. From Property 2.4.2 above we know that these are either marked pruned or are dominant with respect to other cluster centers. It follows from step 21 of Algorithm 5 (which returns all un-pruned objects of Ω), Σ contains mutually dominant objects.

□

Property 2.4.4. For any cluster $\sigma \in \Omega$, its center, χ, has a **higher degree of domination** over its members than any other cluster center κ, i.e.,

$$\tau_{\chi j} \;\; > \;\; \tau_{\kappa j} \;\; \{\forall\, j \, \in \, \sigma, \, \kappa \, \in \, \Omega, \;\; \kappa \neq \chi\} \tag{2.27}$$

Proof. This is evident from Steps 8-10 of Algorithm 6. Each object in Ω is compared with the new cluster center χ. The condition $\tau_{j\chi} < d_{j\kappa}$ is equivalent to $\tau_{\chi j} > \tau_{\kappa j}$, i.e., the new cluster center, χ, has a higher degree of domination over object j than its cluster center, κ.

□

Property 2.4.5. For a cutset Σ of size n and K clusters returned, KCenterPrune takes $O(nK)$ time.

Proof. A single run of Algorithm 6 compares every object in Σ with center χ of the new cluster σ, taking $O(n)$ time. Since each iteration in Algorithm 5 returns a new cluster, with K clusters returned, the overall runtime is $O(nK)$.

□

2.4.7.3 CPSC: Clustering Based SC Algorithm

Algorithm 7 derives mainly from Algorithm 3 combined with Algorithm 5 to compute the statistical criticality (SC). The main difference is Steps 3-15 (differing from Steps 6-13 of Algorithm 3), which update the zone information, accounting for pruned edges in cutsets from previous levels. Unlike Algorithm 3, we only compute the contribution of an mc-edge, e_i, to its zone, Z_i, if it is un-pruned in previous levels. Therefore we do not need forward book-keeping data structure Υ_{iF}, to compute Z^{σ}_{iMAX}, the maximum of mc-edges belonging to Z_i, crossing the current level. Instead, Z^{σ}_{iMAX} is computed online, in Steps 12-15. Due to pruning, the computed reverse book-keeping data structure, Υ_{jR}, of a zone Z_j, may be invalid. On encountering the first edge leaving this zone, we recompute Υ_{jR}, removing all pruned edges from it (Steps 4-6). This is allowed because all edges enter a zone (and therefore it is known if they have been pruned) before any edge exits it.

Step 17 derives a set of mutually dominant edges from cutset Σ, facilitated using Property 2.4.3. Step 18 orders cutset Σ, facilitated by Property 2.4.4. There can be many different orderings when performing the statistical maximum of edges in the cutset [56]. Property 2.4.4 proves that a cluster center has a higher degree of domination over its members than any other cluster center. Therefore, in the order of edges returned, an edge is closer to its most dominating center (as opposed to the case in which a purely random order were chosen). The intuition is that a greater degree of domination between two edges would result in smaller errors in the MAX_{θ} operation, as shown in [68]. Algorithm 5 stops execution when the maximum cluster size equals S. If S were set to a large number, like the

Algorithm 7 CPSC $(G(V, E), \varepsilon)$
// G = circuit timing graph; ε = pruning threshold

```
1:  Algorithm 3, Steps 1-4 to obtain a cutset Σ of edges
2:  for all levels l ∈ G do
3:     for all e_j ∈ Σ_e with end-level l do
4:        if e_j is the first edge to exit zone Z_j then
5:           Remove pruned edges from Z_jR; Recompute Υ_jR
6:        end if
7:        if e_j is un-pruned then
8:           ++z_jR // e_j exits zone Z_j, update reverse pointer
9:           Z_jMAX = Υ_jR(z_jR)
10:       end if
11:    end for
12:    for all un-pruned e_i ∈ Σ_e with start-level l − 1 do
13:       Z_i = zone of e_i
14:       Z^σ_iMAX = MAX_σ(Z^σ_iMAX, e^σ_i)
15:    end for
16:    Algorithm 3, Steps 14-19 to compute cutset Σ
17:    K = KCenterPrune (Σ, ε, 1) // pruning Σ
18:    KCenterPrune (Σ, ε, S) // ordering Σ
19:    Algorithm 3, Steps 20-22 to compute the global criticality of all edges in the pruned
       Σ
20: end for
```

size of the cutset, the algorithm would exit without any clustering iterations and a random ordering would result. For our experiments, we heuristically chose a cluster size S equal to the square root of the number of edges in the cutset, to balance out the number of edges in each cluster and help to reduce the runtime of the ordering step by performing a fewer number of iterations. Our framework is also flexible enough to accommodate other error metrics like [56] or the skewness. Such an ordering cannot be obtained with the ${}^{n}C_2$ pruning strategy of Section 2.4.7.1. Section 2.4.8.1, discussed later, uses a sampling technique which obviates the need for the ordering step. Property 2.4.5 ensures that in a cutset with n edges having a small number of dominant edges, K ($K \ll n$), Algorithm 5 runs in $O(n)$ or linear time.

In summary, our clustering based algorithm eliminates non-dominant edges from the cutset so as to reduce errors (due to Clark's maximum operation, MAX_θ) in the global criticality computation of the dominant edges. Ideally, computing the maximum operation accurately would significantly reduce errors in global criticality. Various techniques have been used to try to reduce the errors in the linear approximation of a set of Gaussian random variables. In [56], the authors give a detailed treatment of the errors in the MAX_θ operation by using error preserving transformations and precomputed lookup tables. These tables are used heuristically to order a set of random variables and compute their statistical maximum. In [68] the authors postpone the computation of the linear maximum during SSTA, if it results in significant non-linearity (distribution skewness is used as a measure of non-linearity). The maximum is propagated as a maximum tuple in such cases. At the primary outputs, a Monte Carlo simulation is performed on the tuple to obtain a better estimation of the circuit delay pdf. The authors in [37] use a moment matching technique to compute non-linear distributions more accurately. Such a technique can be used to get rid of the linearity restriction of the MAX_θ operation to reduce the errors in criticality computation.

Algorithm 8 LS (Σ, Ψ, R)
// Σ = cutset; $\Psi_p = N_{ls}$ x k array of i.i.d. gaussian samples
// $R = N_{ls}$ x $|\Sigma|$ array of i.i.d. gaussian samples

1: **for** $n = 1$ to N_{ls} **do**
2: **for all** $e_i \in \Sigma$ **do**
3: $d_i = \mu_i + \sum_{j=1}^{j=k} a_{ij} \cdot \Psi_p[n][j] + \zeta_i \cdot R[n][i]$
// $\Psi_p[n][j]$ = value of the jth PC, p_j, and
// $R[n][i]$ = value of random component, r_i,
// for edge e_i at simulation point n (see Equation 2.4)
4: **end for**
5: Increment count $M[i]$ of edge e_i with maximum d_i
6: **end for**
7: Compute $T_i = M[i]/N_{ls}$ for all $e_i \in \Sigma$

2.4.8 Reducing Errors

This section describes a simple solution to deal with global errors not eliminated by pruning. We then explore a popular graph reduction technique to speed up criticality computation and finally deal with errors due to independent parameter variations like gate oxide thickness, t_{ox}.

2.4.8.1 LS: Localized Sampling

To tackle edges having global errors (Def. 2.4.6), we perform a quick localized Monte Carlo sampling of the edges in a cutset Σ, pruned using Algorithm 5. The procedure is described in Algorithm 8. The inputs are Σ; N_{ls} samples of the k independent and identically distributed (i.i.d.) Gaussian principal components (PCs) (Eq. 2.4) stored in Ψ_p; array R of N_{ls} i.i.d. Gaussian samples for each edge in Σ. Every sample is used to instantiate the edges e_i in Σ (Steps 2-4), from which we compute the edge with maximum delay (Step 5). Array entry $M[i]$ keeps count of the number of samples for which an edge e_i takes on the maximum delay. This helps us compute the global criticality, T_i, of all edges $e_i \in \Sigma$ in Step 7.

Consider a cutset $\Sigma = \{e_1, \ldots, e_n\}$ with edge path delays, $\{e_1^\sigma, \ldots, e_n^\sigma\}$, represented in

terms of the k PCs (for the purpose of simplicity we ignore the spatially uncorrelated random component of variation r_i). In the k-dimensional space, let $\mathcal{R}$ be the region where e_i^σ takes on the greatest value in the probability space, i.e., $\mathcal{R}$ is the region of dominance of edge e_i in the cutset. The global criticality T_i of edge e_i is given by the volume integral of the joint pdf of the k i.i.d. PCs over $\mathcal{R}$. The LS procedure in Algorithm 8 is a Monte Carlo simulation to compute the volume integral of the joint pdf over region $\mathcal{R}$. The accuracy of LS therefore depends on the number of samples N_{ls} and the accurate computation of the path delay for every edge in the timing graph, or in other words, the forward and reverse SSTA to capture the sensitivities of edge path delays to the k i.i.d. PCs. Intuitively, since edges with high global criticality (large volume integral) have a region of dominance $\mathcal{R}$ near (or including) the origin, the number of samples needed for convergence is not very large. This will be seen in the results Section 2.4.9.

It should be noted that we apply the LS procedure to every cutset of the timing graph. The speedup in LS stems from the reduction of the cutset size using the clustering based pruning procedure of Algorithm 5.

2.4.8.2 Timing Graph Reduction

Since we perform a localized sampling on all the levels of the timing graph, G, reducing the number of levels, L, can speed up the runtime. We exploit the fact that the criticality of a node in G is equivalent to the sum of its fanin edge or fanout edge criticalities. To do this, we perform a timing graph reduction (TGR) procedure on nodes with a single fanin or fanout. A straightforward and practical example of this reduction is an inverter chain, wherein a path enters the chain if and only if it passes through all the edges in the chain. Therefore, the criticality of all these edges is the same. This is illustrated using Figure 2.11. We reduced the timing graph in two phases. In the forward reduction phase, nodes with a single fanout (such as e) are eliminated by connecting their fanins (nodes c and d) to their single fanout (node t). Similarly in the reverse reduction phase, single fanin nodes (such as

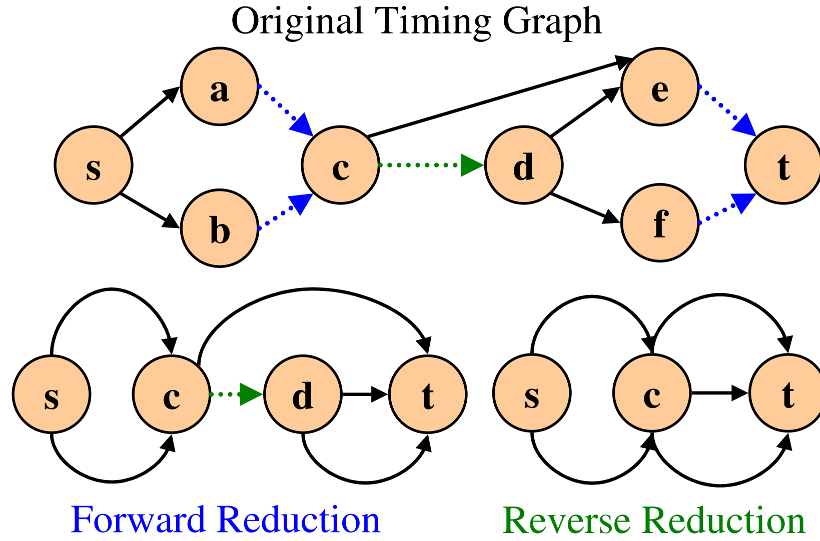


Figure 2.11: Illustration of the timing graph reduction procedure. Nodes a, b, e and f are eliminated in the forward reduction phase. Node d is eliminated in the reverse reduction phase.

d) are eliminated by connecting their fanout (node t) to their single fanin (node c).

The idea of TGR is borrowed from [61], wherein the objective is to eliminate timing graph nodes to reduce the number of variables and constraints in circuit timing optimization. In [61], multiple fanin/fanout nodes are also eliminated, which in our work is the equivalent of computing path criticalities in the timing graph. Path criticalities however do not allow us to deduce the individual edge criticalities that constitute the paths in a straightforward manner. So we restrict TGR to only single fanin/fanout nodes. Given the efficacy of Algorithm 5 to prune out a large number of non-dominant edges in a cutset, we observe that the approach in [61] could be applied to improve runtime, if a particular node or region of the timing graph is known to have a low criticality.

To perform a TGR we scan timing graph G in the forward and reverse directions merging fanins of single fanout nodes into their fanout and fanouts of single fanin nodes into their fanin respectively. Table 2.2 shows the effect of TGR on the number of levels, L, and maximum cutset size, η, on the five largest benchmark circuits. Column 2 shows the size

Table 2.2: The ISCAS89 benchmarks with number of gates, N_G, and independent sources of variation, N_ζ. The effect of TGR on circuit depth, L, and maximum cutset size, η is also shown.

Name	# of Gates	N_ζ	L		η	
			TGR	No TGR	TGR	No TGR
s27	10	36	10	4	7	8
s1196	529	2018	28	11	202	299
s5378	2779	8424	29	7	593	1041
s9234	5597	15942	62	16	644	1529
s13207	7951	22330	63	16	1329	2599
s15850	9772	27290	86	21	1688	2411
s38417	22179	64056	51	13	2821	6638
s35932	16065	56538	33	10	5473	10742
s38584	19253	65512	60	19	5680	10374

of the circuit. As their names imply, columns "TGR" and "No TGR" are results with and without TGR respectively, applied to G.

2.4.8.3 Spatially Uncorrelated Independent Parameter Variations

Revisiting Equation 2.4, independent (spatially uncorrelated) parameter variations like the variation in oxide thickness, t_{ox}, of a transistor, are captured by the single random variable r_i. This is done to avoid tracking the individual contribution of t_{ox} for every transistor in the design as a separate term in the canonical form, as done in [38]. However, errors can occur in the path delay of reconvergent paths, as shown in Fig. 2.12 taken from one of the ISCAS89 benchmarks.

The figure shows gate G_{11} driving 5 other gates. The arrival time (Definition 2.1.2) at the fanouts of G_{11} consists of a structurally correlated term to capture the variation in the oxide thickness of transistor G_{11}, denoted Δt_{ox}. Since the canonical form consists of a single term to capture spatially uncorrelated variations, r_i, in cutset Σ, these are considered independent, and may cause errors in high criticality paths (shown in bold) particularly when such fanouts have a high degree of correlation. In our experiments, ignoring the structural correlations led to errors of up to 60%, the main culprits being cutsets with reconvergences

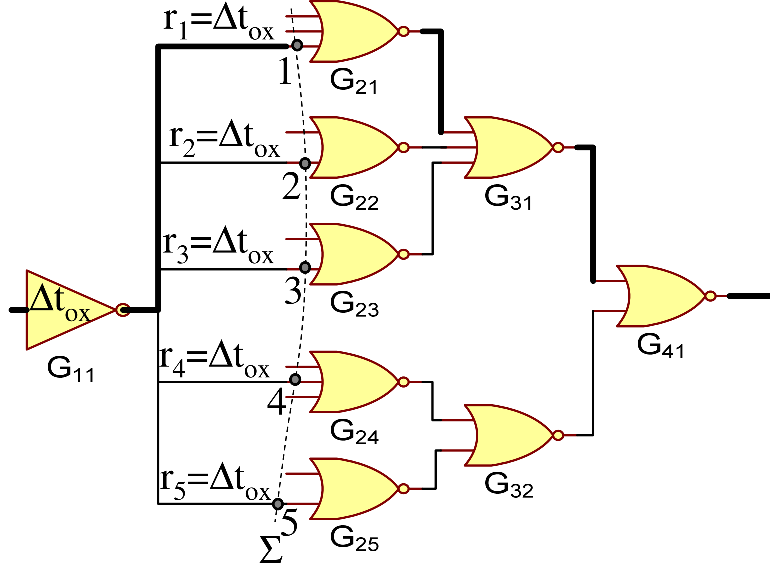


Figure 2.12: A reconvergent structure from one of the ISCAS89 benchmarks with a high criticality path indicated using bold lines. Arrival time correlations of fanouts in cutset Σ, denoted r_i, due to variation in oxide thickness, Δt_{ox}, of gate G_{11}, cause structural correlations in reconvergent fanouts like G_{41}.

similar to Fig. 2.12. Also, to calculate the statistical MAX_σ at the convergence of the paths containing gates $G_{21}, G_{22}, \ldots, G_{25}$, i.e., at gate G_{41}, we need to factor in the common Δt_{ox} of gate G_{11} to reduce inaccuracies in MAX_σ. To keep track of the structural correlations due to spatially uncorrelated independent parameter variations like Δt_{ox}, on encountering a multiple fanout gate like G_{11}, we expand the canonical form of the path delay with its Δt_{ox} variation to accurately compute the arrival time of the downstream gates in the circuit. A similar expansion is performed for gates with multiple fanins while reverse traversing the timing graph to compute the required times of upstream edges. Although the number of terms in the canonical form increases, using a linear sparse array, we only keep track of terms with non-zero sensitivities in the edge path delay. Table 2.2 shows the total number of independent sources of variation for the benchmarks under column three, labeled N_ζ. As seen in Section 2.4.9, this does not adversely impact the runtime.

2.4.9 Results

Our algorithms were implemented in C++ on top of an SSTA engine [15] and exercised on the 12 largest ISCAS89 benchmarks, with parameter values corresponding to the 100nm technology node [2]. Experiments were conducted on a Linux PC with a 3.0-GHz CPU and 2GB RAM. The average ratio of the standard deviation to the mean of circuit delay was about 12%. We compared four schemes with Monte Carlo simulations using 10000 samples, shown in Table 2.3.

The first scheme is the zone-based ZSC approach in Algorithm 3. Scheme ${}^{n}C_2$ additionally implements the pairwise pruning strategy of Section 2.4.7.1 with a pruning threshold, $\varepsilon = 5\%$. CPSC implements Algorithm 7 using our clustered pruning and ordering technique. CPSC+LS+TGR performs clustered pruning on the reduced timing graph (TGR) and computes criticalities using the LS procedure (Algorithm 8) with $N_{ls} = 1000$ samples. All approaches excluding ZSC account for structural correlations due to independent parameter variations as described in Section 2.4.8.3. Row "maximum $\%$ δ" reports the maximum difference between the edge criticality computed using any of the above mentioned schemes and the Monte Carlo simulations, row "runtime" reports the running time in seconds and "η" reports the maximum number of edges in any cutset of the timing graph after pruning. We exclude the times for SSTA and generating the N_{ls} samples in LS.

From Table 2.3, ZSC, which computes criticalities using Clark's MAX_θ formulation results in large errors (the largest being about 60%). As described in Section 2.4.6, this is mainly due to the propagation of local errors. CPSC with cutset pruning and ordering does better than ZSC in accuracy and runtime. For circuits exhibiting large global errors, the LS procedure helps reduce them further. Rows in bold compare ZSC with CPSC+TGR+LS. The combined approach greatly reduces the errors and runtime, due to pruning. Moreover, runtime increase is negligible compared to CPSC (an anomaly is s35932 wherein runtime decreases due to TGR). For the 3 large benchmarks we obtain about an order of magnitude

Table 2.3: Criticality run-times and errors for various benchmarks; $\varepsilon = 5\%$ and $N_{ls} = 1000$ ZSC - Zone based criticality, ${}^nC_2$ - pairwise pruning scheme, CPSC - clustering based pruning scheme, TGR - timing graph reduction, LS - localized sampling

Metric	Pruning Scheme	Benchmark											
		s3271	s3330	s3384	s4863	s5378	s6669	s9234	s13207	s15850	s38417	s35932	s38584
maximum $\%\ \delta$	ZSC	**44.51**	**36.19**	**43.23**	**31.82**	**59.95**	**40.24**	**38.17**	**41.75**	**44.56**	**34.78**	**21.11**	**48.21**
	${}^nC_2$	4.70	3.82	0.03	17.40	26.94	26.74	36.63	15.72	32.20	30.29	14.95	20.28
	CPSC	4.70	1.42	0.03	17.40	37.41	30.25	36.57	15.64	37.32	30.29	14.95	21.18
	CPSC+TGR+LS	**4.70**	**1.62**	**0.03**	**9.08**	**7.18**	**2.90**	**3.28**	**2.33**	**4.52**	**4.70**	**1.82**	**15.88**
runtime (sec)	ZSC	**0.05**	**0.04**	**0.07**	**0.11**	**0.12**	**0.16**	**0.19**	**0.24**	**0.28**	**1.43**	**1.32**	**1.47**
	${}^nC_2$	0.11	0.09	0.18	0.52	0.51	0.91	1.15	2.36	2.65	58.15	74.69	59.23
	CPSC	0.01	0.01	0.01	0.03	0.02	0.06	0.05	0.03	0.05	0.16	0.36	0.11
	CPSC+TGR+LS	**0.01**	**0.02**	**0.01**	**0.12**	**0.04**	**0.25**	**0.15**	**0.06**	**0.14**	**0.25**	**0.25**	**0.22**
η	ZSC	**622**	**451**	**603**	**528**	**593**	**965**	**644**	**1329**	**1688**	**2821**	**7340**	**5680**
	${}^nC_2$	2	13	1	6	7	11	19	7	8	14	66	15
	CPSC	2	13	1	6	7	11	19	7	8	14	66	15
	CPSC+TGR+LS	**2**	**13**	**1**	**7**	**6**	**12**	**17**	**7**	**8**	**12**	**66**	**15**

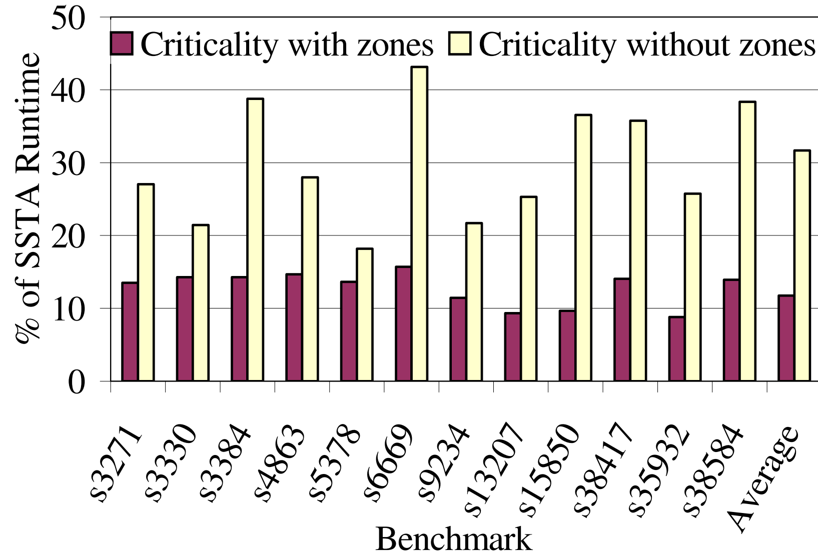


Figure 2.13: Runtime of criticality computation as a fraction of SSTA runtime. The two cases shown are with and without the zone-based algorithm to compute the statistical maximum of mc-edges crossing a level in the timing graph.

difference in run-times of ZSC and the combined approach. Most circuits have errors below 10%, except for s38584. On investigation, it was found that for large fanout structures, path delays themselves (computed in terms of the PCs) contained large errors and hence the LS procedure does not completely eliminate global errors. In terms of the efficacy of our pruning strategy, as expected we vastly outperform the ${}^{n}C_{2}$ procedure in runtime (about two orders of magnitude for the larger benchmarks). Each circuit also contained an identical number of edges remaining in the cutsets using the ${}^{n}C_{2}$ and CPSC pruning strategies, seen from the entries in row "η".

To evaluate the runtime effectiveness of the zone-based approach, Fig. 2.13 shows the criticality computation runtime with (denoted 'Criticality with zones') and without (denoted 'Criticality without zones') zones as a fraction of the SSTA runtime. In all cases, structural correlations due to independent parameter variations were not taken into account. On average, criticality computation with zones is about $10X$ faster than SSTA and we obtain a speedup of about $2.7X$ in the runtime compared to the case without zones. The runtime

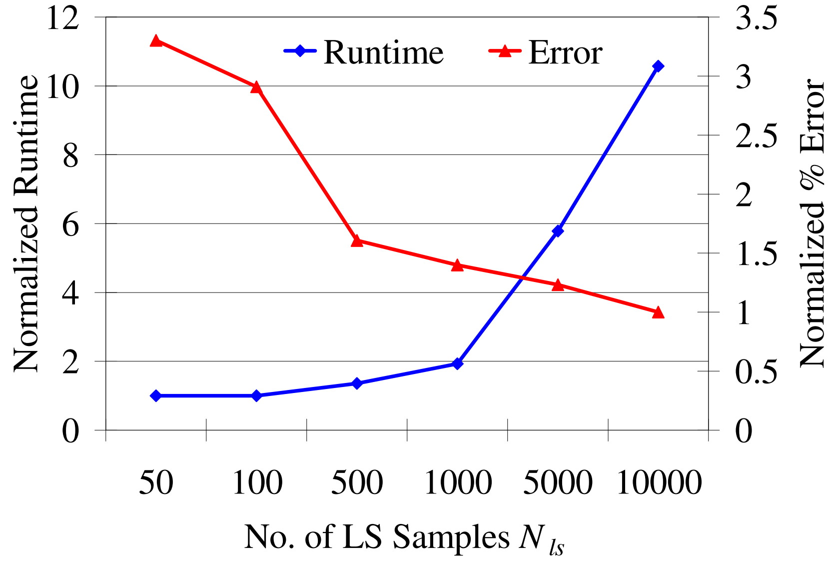


Figure 2.14: Trade-off showing number of LS samples, N_{ls}, vs the overall criticality computation runtime and maximum percentage error (with respect to a Monte Carlo simulation with 10000 samples), for the s38417 ISCAS89 benchmark. Runtimes are normalized to the case with $N_{ls} = 50$ and the error is normalized to the case with $N_{ls} = 10000$.

for the zone computation procedure of Algorithm 2 on average was less than 0.5% of the SSTA runtime.

In deciding N_{ls}, we observed that as the number of samples increases, the improvement in accuracy diminishes. Figure 2.14 shows the trade-off between the number of samples N_{ls} and the maximum percentage error δ, obtained between our clustering based approach and a Monte Carlo analysis with 10000 runs. As expected, with a small number of LS samples, $N_{ls} < 500$, the error is more than double that with $N_{ls} = 10000$. However, as the number of samples increases, say from 1000 to 5000, the overall runtime almost triples, without much reduction in error. Moreover, as N_{ls} increases, the overall runtime is dominated by the time for LS. In our algorithm, to maintain a reasonable trade-off of accuracy and runtime, we chose $N_{ls} = 1000$.

Figure 2.15 shows the variation of runtime and accuracy (averaged over all benchmarks) when pruning threshold, ε, is varied. With an increase in ε, the cutset size decreases,

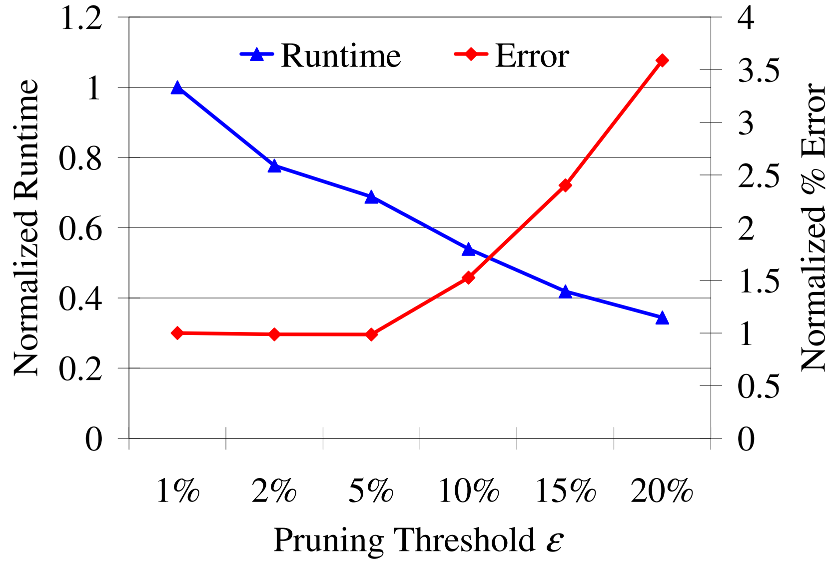


Figure 2.15: Trade-off showing pruning threshold, ε, vs the overall criticality computation runtime and maximum percentage error (with respect to a Monte Carlo simulation with 10000 samples), averaged over the 7 largest ISCAS89 benchmarks. The runtimes and error are normalized to the case with $\varepsilon = 1\%$. The number of samples used in LS, $N_{ls} = 1000$.

reducing the overall criticality runtime (mainly due to reduction in the runtime for LS). For pruning thresholds below 5%, the error is relatively constant since the non-dominant edges eliminated do not adversely affect the global criticality of dominant edges. Therefore in our algorithm, we chose a pruning threshold of 5% to obtain good accuracy with a reasonable runtime.

Finally, we implemented the approach of Xiong *et al.* in [65] and compared its performance with our clustering based approach for the benchmarks shown in Table 2.3. For a fair comparison, we ignored independent parameter variations when comparing the two approaches. On average, we obtain a speedup of about $5X$ over the approach in [65]. This is mainly attributed to cutset pruning, which eliminates a large number of non-dominant edges, thereby reducing the number of criticality computations. The advantage of cutset pruning is particularly pronounced for the larger sized benchmarks.

The difference in maximum percentage error ($\%\ \delta$) when compared to a Monte Carlo sim-

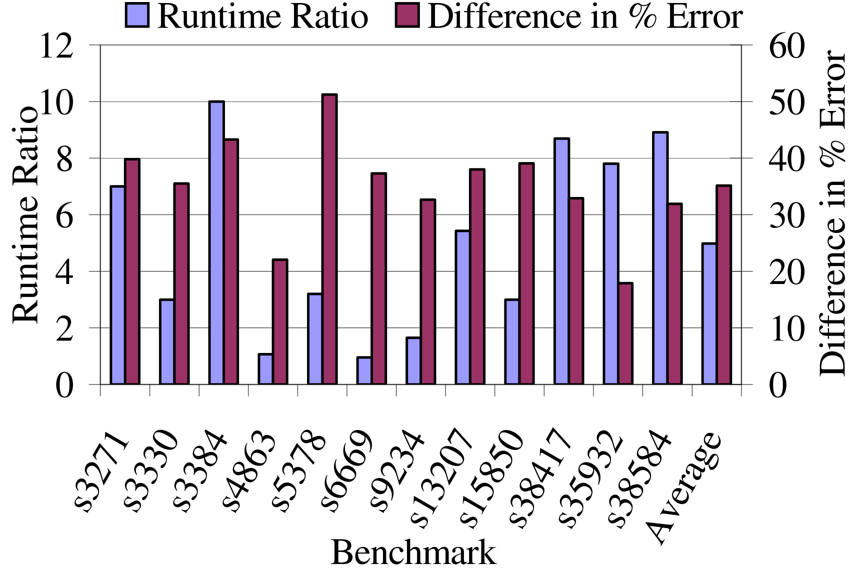


Figure 2.16: Comparison of runtime ratio and difference in maximum criticality percentage error between our implementation of the approach in [65] and the clustering based approach, referenced to a Monte Carlo simulation of 10000 samples. The number of samples used in LS, $N_{ls} = 1000$, and the pruning threshold, $\varepsilon = 5\%$.

ulation of 10000 runs, is shown in Figure 2.16 on the secondary axis. On average, over all the benchmarks, we see that if our algorithm reported the maximum criticality difference with a Monte Carlo simulation of $x\%$, the approach in [65] reported a maximum criticality difference of $x+35\%$. The errors are of similar magnitude to our zone-based scheme, ZSC, implemented without cutset pruning (Table 2.3), since fundamentally both the approaches are similar. Hence, as was seen in the *abc* problem, local and global errors contribute to large overall errors in criticality computation (Table 2.1).

2.5 Conclusion

In this chapter we presented a new linear time technique to compute statistical criticalities of nodes and edges in a timing graph. We used the idea of interval zones to process edges crossing multiple cutsets in linear time. We introduced the notion of local and global errors in the computation of the maximum of a set of random variables. In order to reduce errors in criticality computation, we have also developed a new clustering-based heuristic capable of

pruning and ordering edges in a cutset to reduce the local and global errors resulting from Clark's tightness probability formulation. Our clustering based pruning competes very well with a pairwise pruning strategy in terms of accuracy and delivers large speedups in runtime. Using the pruning technique with localized sampling and timing graph reduction, our computations reduce errors to around 5% when compared to Monte Carlo simulations, even in the face of large gate delay variations. An important topic for future work is to use our clustering based framework to compute criticality incrementally. This is important for any circuit timing optimization based framework, wherein either the designer perturbs the circuit incrementally, referred to commonly as the Engineering Change Order (ECO) flow, or the tool automatically perturbs the circuit, for example, to improve the timing of the design. A big advantage of our algorithm as pointed out in Section 2.4 is that we can process cutsets in any order to compute edge criticalities.

Chapter 3

Reduction of On-Chip Temperature via Static Thermal-Aware Floorplanning to Control Sub-threshold Leakage

3.1 Introduction

With scaling of technologies, the increasing on-die temperature of a silicon chip poses a threat to the successful and timely tape-out of a digital design. Moreover, with increasing device densities, multiple cores on a single die have become common place and thermal and power issues are assuming a greater significance in the overall digital design flow. For example, the new Penryn® family of Intel® processors has various states of operation to control the on-die power [4]. In this chapter we discuss the problem of increasing power density and temperature, which has exacerbated with smaller technology nodes. This is primarily due to the interdependence of temperature and subthreshold leakage power. We discuss ways of mitigating this by means of static and dynamic thermal management techniques which directly target the reduction of subthreshold leakage power, thereby reducing

the on-chip temperature. We first begin by describing the effect of temperature on the subthreshold leakage of a chip. Since the power dissipated on a chip manifests itself in the form of temperature buildup, it is important to discuss methods used to model the on-chip temperature of the die. We then discuss a flooplanning flow to improve the thermal profile of the chip taking into consideration the subthreshold leakage for micro-architecture designs.

3.2 Power Dissipation in a Digital Design

This section describes two major sources of power dissipation in a digital circuit. The first is the dynamic power or switching power which arises in the normal course of activity due to the active switching of gates on the chip. The second is the subthreshold leakage or idle power which is the power dissipated on chip when the system is idle. Although dynamic power is a major component of the total on-chip power, over the past few technology nodes, the subthreshold leakage has been increasing consistently and is expected to reach about 50% of the total power consumption in future.

3.2.1 Dynamic Power Dissipation

As we know from first principals, current flowing through a resistor causes energy to be dissipated in the form of heat. On the silicon die of a digital circuit, we have millions of transistors switching to perform the operations of the circuit. The load capacitance driven by the transistors are constantly being charged and discharged over regular intervals of time, decided by the operating frequency of the chip. The energy consumed is dissipated as heat. For instance, an inverter switching from the off to on state consumes an energy of

$$E_{i/2} = 1/2 \cdot C \cdot V^2 \tag{3.1}$$

In the above equation, C is the capacitance of the load the inverter drives and V is he supply voltage of the inverter. The energy is dissipated as heat in the inverter. An equivalent amount of energy is also stored in the load capacitor. On switching from the on to the off state, the capacitor discharges dissipating the same quantity of energy as heat. If the inverter switches in every cycle of operation, the total energy consumed by it during a single cycle is given by,

$$E_i = C \cdot V^2 \tag{3.2}$$

Therefore, the power consumed by the inverter as a function of the frequency f of operation is given by

$$P_i = C \cdot V^2 \cdot f \tag{3.3}$$

With N gates on a chip switching at different rates, very often the above equation is modulated by factor α, which gives a measure of the average switching activity of a gate. The dynamic power (switching power) of a design is therefore given by,

$$P_D = \alpha \cdot N \cdot C \cdot V^2 \cdot f \tag{3.4}$$

It is important to realize that the dynamic power is a function of circuit design. Techniques to reduce dynamic power include clock gating (reduces switching activity α), using short wires (to reduce load capacitance C), using a low operating voltage (V) and operating at a lower frequency (reducing f).

3.2.2 Subthreshold Leakage Power Dissipation

As mentioned before, over the last few technology nodes (mainly $90nm$ and below), another mode of power dissipation has gained increasing importance and is due to the subthreshold power also known as static power dissipation. Subthreshold leakage conduction is the phenomenon of current flow in a CMOS transistor even when it is not switching. The ITRS [55] predicts that in the next few years, full chip leakage power will double from it's current value. The rest of this chapter deals with the topic of subthreshold leakage (also referred to as leakage hereon) and its dependence on temperature. The subthreshold drain current of a CMOS transistor given by the BSIM3 model in [3] is shown below,

$$I_{ds} = I_{s0} \cdot (1 - e^{(-\frac{V_{ds}}{v_t})}) \cdot e^{(\frac{V_{gs}-V_{th}-V_{off}}{nv_t})} \tag{3.5}$$

where, I_{s0} is given by

$$I_{s0} = \mu \frac{W}{L} \sqrt{\frac{q\epsilon_{si}N_{ch}}{2\phi_s}} {v_t}^2 \tag{3.6}$$

and V_{th} is the transistor threshold voltage. The temperature dependence arises due to the thermal voltage v_t given by,

$$v_t = \frac{K_B T}{q} \tag{3.7}$$

where T is the temperature in Kelvin. It must be noted that in equation 3.5, the third term has a negative exponent since the subthreshold conduction region is defined by $V_{gs} < V_{th}$. This term primarily contributes to the exponential dependence of subthreshold leakage current on temperature. Rewriting Equation 3.5 to make explicit, the subthreshold leakage current dependence on temperature, we get,

$$I_{ds} = k_1 \cdot T^2 \cdot (1 - e^{(-\frac{k_2}{T})}) \cdot e^{\frac{-k3}{T}} \quad (3.8)$$

where k_1, k_2 and k_3 are factors > 0.0. As discussed previously, the power (dynamic) consumed in the gates of a silicon die is dissipated as heat. The rate of heat dissipation determines the temperature build-up in the chip. If heat is not dissipated efficiently, very high chip temperatures can result which can affect circuit reliability and cause failure. To alleviate this problem, typically high power consumption chips like microprocessors have large heat sinks to efficiently dissipate the heat and avoid high on-chip temperatures. In light of the above discussion, it can be easily seen that an increase in chip temperature will result in an increase in the chip power consumption due to the exponential dependence of leakage on temperature. This can result in a positive feedback loop also referred to as the phenomenon of thermal runaway[1]. This problem is exacerbated further because the temperature distribution of the chip is not even given that some portions of the silicon surface get hotter than others. These are referred to as hotspots in the design, where the danger of thermal runaway is even higher. It therefore becomes imperative that subthreshold leakage be controlled and managed effectively.

Although designing chips with larger and better heat-sinks can reduce on the on-chip temperature, this comes at the price of increased manufacturing costs and is not feasible in most applications. To counter the temperature challenge, over the recent years many architecture level techniques have been introduced to reduce on-chip temperature. The rest of this chapter discusses such approaches and describes in detail our approach to leakage and temperature reduction using physical floorplanning.

[1]Thermal runaway refers to the process in which an increase in the temperature of a system causes a further increase in its temperature leading to an instable and often destructive outcome

3.3 Thermal Modeling in a Digital Design

In this section we briefly describe a basic technique used to model on-chip temperature. The technique is crucial to evaluate the effect of our techniques for temperature reduction. The flow of heat in a system (for our purposes, on a chip) is governed by the well known partial differential equation (PDE)

$$\rho c_p \frac{\partial T(x,y,z,t)}{\partial t} = k_t \nabla^2 T(x,y,z,t) + g(x,y,z) \tag{3.9}$$

In the above equation, t represents time, x, y, z represent spatial coordinates, c_p the heat capacity of the chip material, k_t the thermal conductivity, ρ is the material density, g is the power density per unit volume and T is the temperature at the particular point of interest. For chip thermal analysis, the conductivity can be assumed to be a constant to simplify the solution of the linear PDE. The above equation describes the temperature of a system varying with time. For a steady state analysis, when the system reaches thermal equilibrium, the differential terms with respect to time become zero and we get

$$\nabla^2 T(x,y,z,t) = -\frac{g(x,y,z)}{k} \tag{3.10}$$

which is the well known Poisson's equation. This equation can be solved using a number of methods, given the boundary conditions, one of which is relatively popular and described next.

3.3.1 Finite Difference Thermal Model

One of the most commonly used methods is the Finite Difference Method (FDM), which divides the chip into three dimensional discrete elements of equal size, say δx, δy and δz, as illustrated in Figure 3.1 for two dimensions, and tries to solve Equation 3.10 at each

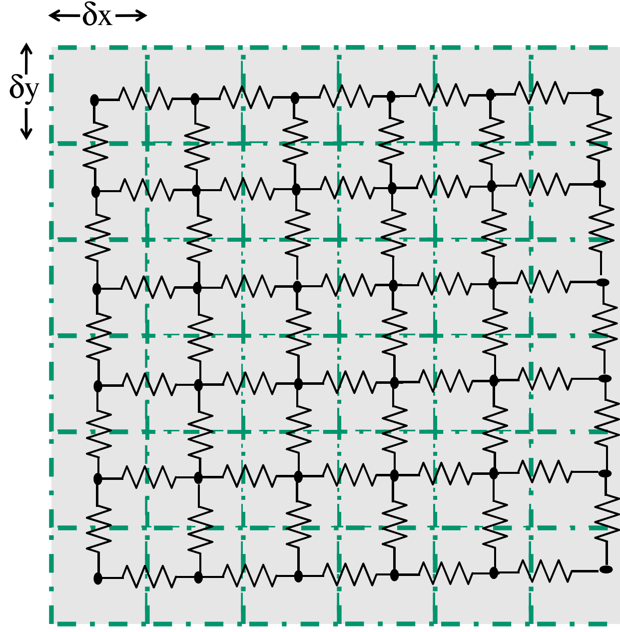


Figure 3.1: Illustration of a grid of discretized elements in two dimensions used in the Finite Difference Method for thermal modeling.

discrete point, say (i, j, k). Such a discretization results in a system of linear equations which can be solved by well known algebraic solvers. For Equation 3.9, the finite difference approximation can be written as,

$$\begin{aligned}
\frac{\rho c_p}{k}\frac{\partial T(i,j,k)}{\partial t} + \frac{-\delta g(i,j,k)}{\delta x \delta y \delta z} &= \frac{(T_{i-1,j,k}-T_{i,j,k})-(T_{i,j,k}-T_{i+1,j,k})}{(\delta x)^2/k} \\
&+ \frac{(T_{i,j-1,k}-T_{i,j,k})-(T_{i,j,k}-T_{i,j+1,k})}{(\delta y)^2/k} \\
&+ \frac{(T_{i,j,k-1}-T_{i,j,k})-(T_{i,j,k}-T_{i,j,k+1})}{(\delta z)^2/k}
\end{aligned} \tag{3.11}$$

The above equation is often rewritten as,

$$\begin{aligned}
c\frac{\partial V}{\partial t} - I(i,j,k) &= (V_{i-1,j,k} - V_{i,j,k})g_{i,i-1} + (V_{i+1,j,k} - V_{i,j,k})g_{i,i+1} \\
&+ (V_{i,j-1,k} - V_{i,j,k})g_{j,j-1} + (V_{i,j+1,k} - V_{i,j,k})g_{j,j+1} \\
&+ (V_{i,j,k-1} - V_{i,j,k})g_{k,k-1} - (V_{i,j,k+1} - V_{i,j,k})g_{k,k+1}
\end{aligned} \tag{3.12}$$

where the term $g_{i,i-1} = g_{i,i+1} = k\delta y \delta z/\delta x$ denotes the thermal conductance at node i, j, k

in the x dimension, temperature T is replaced by voltage V, and the power at node i, j, k is replaced by an equivalent current $I(i, j, k)$ and the rate of heat buildup is captured by a capacitance term c, where t represents time. This is the well known electrical/thermal duality of a system, wherein heat sources (the transistors of the silicon die) are represented by current sources, the temperature of each point in the system represents the voltage drop with respect to the ground (or ambient in the case of a thermal system) and the thermal conductance is similar to the electrical conductance and depends on the material properties of the system.

The above Equation 3.12 can be written for each node of the discretized thermal silicon die to give a system of linear equations written as,

$$\begin{bmatrix} c_{11} & 0 & \dots & 0 \\ 0 & c_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & c_{nn} \end{bmatrix} \begin{bmatrix} \frac{dT_1}{dt} \\ \frac{dT_2}{dt} \\ \vdots \\ \frac{dT_n}{dt} \end{bmatrix} + \begin{bmatrix} g_{11} & g_{12} & \dots & g_{1n} \\ g_{21} & g_{22} & \dots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & \dots & g_{nn} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_n \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{bmatrix}$$

$$C \quad \frac{dT}{dt} \quad + \quad G \quad T \quad = \quad P \tag{3.13}$$

In the above equation, g_{ij} is the thermal conductance between nodes i and j, c_{ii} is the thermal capacitance at node i, T_i is the temperature at node i and P_i is the power dissipated at node i. The diagonal matrix C is called the thermal capacitance matrix. Matrix G is called the thermal conductance matrix and is symmetric and positive semidefinite.

In many thermal simulation problems, with given boundary conditions, the steady state behavior of the system is more meaningful to solve. For such a condition, the derivative term of temperature with respect to time vanishes and we get the following steady state thermal equation.

$$\begin{bmatrix} g_{11} & g_{12} & \dots & g_{1n} \\ g_{21} & g_{22} & \dots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & \dots & g_{nn} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_n \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{bmatrix}$$
$$G \quad T = P \tag{3.14}$$

A physical interpretation of the thermal conductance matrix G in the above equation can be obtained by inverting it to obtain R, the transfer thermal resistance matrix [58] given below.

$$\begin{aligned} G^{-1} &= R \\ &= \begin{bmatrix} R_{11} & R_{12} & \dots & R_{1n} \\ R_{21} & R_{22} & \dots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ R_{n1} & R_{n2} & \dots & R_{nn} \end{bmatrix} \end{aligned} \tag{3.15}$$

In the above equation, r_{ij}, is defined as the transfer thermal resistance of node i with respect to node j and gives the rise of temperature at node i due to a unit of power dissipated at node j. Equivalently, r_{ij} represents the increase in the temperature of point i due to a unit increase in power at node j.

Rewriting Equation 3.14 we get,

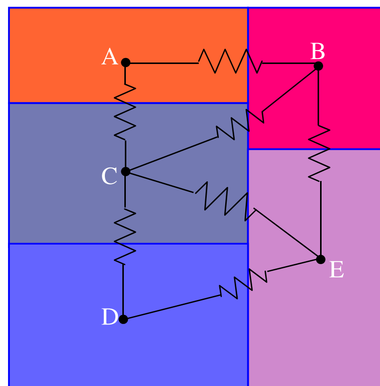
$$
\begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_n \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \dots & r_{nn} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{bmatrix} \tag{3.16}
$$

Equations 3.14 and 3.16 are the basis for many steady state thermal solvers which compute the temperature distribution of a digital chip given the distribution of power sources on the silicon die. The idea is to model the thermal properties of the system to compute the thermal conductance matrix in Equation 3.14 and using well know linear matrix solvers, which employ techniques like the Gauss-Siedel method, solve it efficiently.

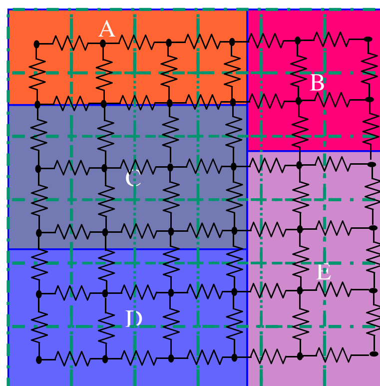
3.3.2 HotSpot Thermal Modeler

HotSpot [31] was developed at the University of Virginia as a block based thermal modeling tool. It exploits the duality between thermal power dissipation and current flow in an electrical network as described above, to construct an RC thermal model of resistors and capacitors in the system. There are primarily two flavors, called the **block** and **grid** models, illustrated in Figure 3.2 using a toy chip layout with 5 blocks labeled $A - E$.

In the **block** thermal model, shown in Figure 3.2(a), every block in the chip is modeled as a node in the RC thermal network, with a thermal conductance between nodes whose blocks are adjacent to each other in the layout. The value of the conductance is heuristically derived using the chip material thermal properties and the amount of overlap a block's boundary shares with its neighbor. Each block also consists of a vertical thermal resistance (not shown in Figure 3.2(a)) proportional to the thickness of the chip and a thermal capacitance proportional to the volume of the block. The thermal capacitance is useful to perform transient thermal simulations. Given the block power profile of the chip and the conductance matrix, the authors construct a matrix equation similar to Equation 3.16 to compute



(a)



(b)

Figure 3.2: Illustration of the HotSpot **block** and **grid** thermal models, on a toy chip layout with 5 blocks. Only the active device layer is shown.

a steady state thermal profile of the chip. It takes in the order of $O(n^2)$ time to construct the model, where n is the number of blocks in the design, since pairwise adjacency relationships need to be examined. Because the model is rather approximate, it can be used to estimate the steady state thermal profile very quickly.

In the **grid** thermal model, shown in Figure 3.2(b), the authors use the more conventional Finite Difference Model, to break the chip into evenly sized grids as discussed in Section 3.1. The power profile of each block in the layout is apportioned to grids according to the number of grids spanned by the block. Because the model is more detailed, it can be used to accurately estimate the steady state thermal profile, albeit at a larger runtime cost.

It must be mentioned that although we have only shown the active layer to illustrate the main idea, HotSpot also models the substrate and inter-device layers, in a similar manner. The heat sink is modeled by a small resistance to ground, which represents the heat conduction path from the chip to the ambient. Another point to note is that HotSpot also includes the ability to solve the transient thermal Equation 3.13, using a 4th order Runge-Kutta method.

3.4 Floorplanning

This section briefly describes the technique of floorplanning used in the early phases of digital design. Typically a complex design contains millions of gates which need to be placed on a silicon die. Detailed placement however is a very expensive process, since the automation tool has to consider effects such as timing closure, power drop and routing congestion to name a few. To tackle with this complexity, multi-million gate designs are often first partitioned into blocks using what is called a partitioning tool. The designer can then perform a detailed but tractable placement on each of these partitioned blocks to optimize certain cost functions. The blocks however need to be themselves placed on the silicon die and this process is known as floorplanning. In placing the blocks various considerations

must be given. At a minimum, the designer must consider the physical restrictions of the block, i.e., the area alloted to each block i, given by A_i, and the allowable aspect ratio of block i, given by AR_{imax} and AR_{imin}. In addition, many floorplanners consider other constraints like the half-perimeter wirelength of the floorplan, denoted by $HPWL$, which is a metric to measure the cost to connect the various blocks of the floorplan. The floorplanning problem can be defined as follows.

Definition 3.4.1 (Floorplanning Problem). The input is a primary set of n blocks, $\Phi_P = \{B_1, \ldots, B_n\}$, each with its associated area A_i and aspect ratio bounds $[AR_{imin}, AR_{imax}]$. The output is the location $\{x_i, y_i\}$ of each block in the floorplan, along with its width W_i and height $H_i = A_i/W_i$ such that the total area and half-perimeter wirelength of the die is minimized. In many cases, the outline of the floorplan is fixed, in which case the problem is referred to as fixed-outline floorplanning.

The key idea of a floorplanning algorithm is to first obtain a useful representation of the floorplan, which is flexible enough to represent an optimal solution. To obtain an optimal solution, we search through the various possible configurations that can be represented and find the one which yields the minimum cost, for example, the weighted sum of total area and $HPWL$. However, any useful algorithm contains millions of representations of the floorplan and the floorplanner has to work hard to intelligently select the optimal configuration. There is a vast amount of literature on various floorplanning techniques used in the EDA domain. For our work however, we use Parquet [8], a well known hierarchical floorplanning tool which uses the sequence pair representation and the simulated annealing optimization algorithm to perform floorplanning.

3.4.1 Sequence Pair Representation

The sequence pair method was invented by the authors in [43] as a means to develop an efficient floorplanner. The authors introduce the problem of rectangle packing (floorplanning)

and develop a novel representation of any packing of blocks, which is called a Sequence Pair. As its name implies, each packing of blocks can be represented by a pair of sequences, which is an ordered pair of module names in a certain sequence. For each packing of blocks, the authors use a gridding method to formulate the sequence pair representing the packing. This works by tracing what are called step lines, and can be done by moving an imaginary pebble which starts off from the upper right corner of a block and moves in an upward and right direction until it either hits another step line, the boundary of another block, or the chip boundary. The union of the up-right and down-left step line, called the positive step line, defines a linear ordering of the step lines which gives us one of the sequence pairs. A similar down-right and up-left tracing is done to obtain the negative step line which gives us the second sequence pair.

The inverse mapping from the sequence pair to a packing or placement of modules is also shown. This is done by pairwise examining the relationship between the modules. Each of these pair wise relationships define the relative placement between the two blocks in consideration, i.e., if we consider blocks a and b, then examining their ordering within each sequence pair, we can determine if a lies to the left of b and if a lies above b. The authors of [43] therefore propose an algorithm to construct an optimal packing given the sequence pair, by constructing what are called horizontal and vertical constraint graphs, assuming that each block has one fixed shape. These are directed acyclic graphs, where the modules are represented by nodes, and edges between them encode their pairwise relationship. An edge exists in the horizontal constraint graph (HCG), between nodes (modules) a and b, if a is to the left of b, as inferred from the sequence pair. An edge exists in the vertical constraint graph (VCG), between nodes a and b, if a is above b, as inferred from the sequence pair. The nodes in the graph are weighted by the respective block dimensions, width in the HCG and height in the VCG. Once these graphs are constructed, they are traversed in topological order (and independent of each other) to compute their coordinates. On a set of n modules, this algorithm takes $O(n^2)$ time. The authors in [43] use this basic algorithm in conjunction

with Simulated Annealing, to find an optimal floorplan of blocks, where each block has a fixed size (width and height).

3.4.2 Parquet Floorplanner

Over the years a number of works have helped to speed up the evaluation of the floorplan from its sequence pair representation, using algorithms based on the longest common subsequence. These algorithms improve the order of runtime from quadratic in the number of blocks n, to $O(n \log n)$ and even $O(n \log(\log n))$. The authors in [8] use the sequence pair method to develop a hierarchical floorplanner called Parquet. The major contribution of their work was to extend the techniques of previous sequence pair based floorplanners to achieve faster convergence and better quality of the final solution. In particular, the authors use the notion of slack from Statistical Timing Analysis (STA), to target certain blocks for moves during the floorplanning process, illustrated in Figure 3.3. The horizontal and vertical slack of a block indicates the amount it can be moved in the horizontal and vertical dimensions respectively, without increasing the size of the floorplan.

Like many floorplanners, Parquet uses the Simulated Annealing algorithm, first developed by the authors in [34], to obtain an optimal solution. The process works as follows. First, the cost function, C, of the floorplanner is defined, keeping in mind the objective function to be minimized. This is a combination of the total area, A, half perimeter wirelength, $HPWL$, and the aspect ratio, AR, of the floorplan, shown below.

$$C = \alpha \cdot A + \beta \cdot HPWL + \gamma \cdot AR \tag{3.17}$$

The factors, α, β and γ are used to appropriately weight the importance of the various cost factors. At each step, the floorplanner makes a move and computes the change in cost, ΔC, that would result in making this move. If the cost is improved ($\Delta C < 0$), then the

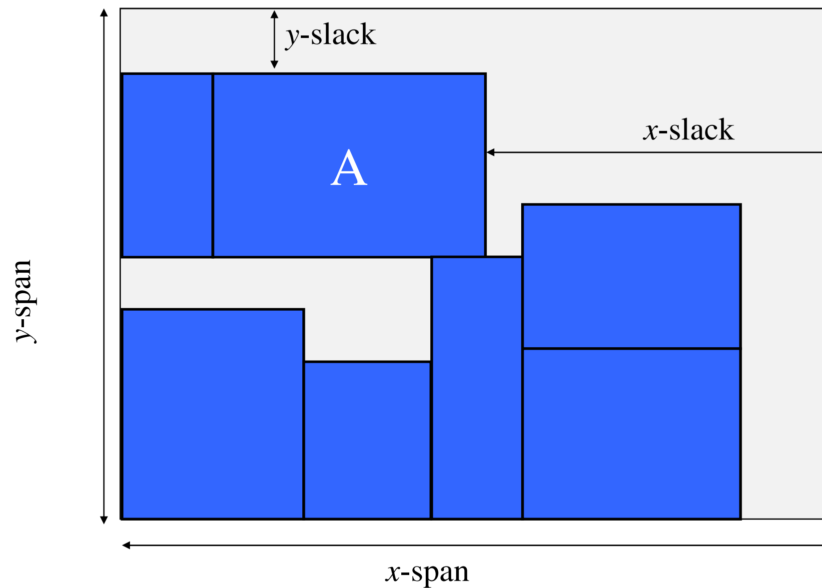


Figure 3.3: Parquet uses the concept of slack from static timing analysis to optimize floorplan area. The slack of block A in the x and y dimensions is shown in the figure.

move is unconditionally accepted. If the move worsens the cost however ($\Delta C > 0$), it is conditionally accepted, the condition being $r < e^{\frac{-\Delta C * k}{T}}$, where r is a random value which is chosen between 0.0 and 1.0, k is the Boltzman constant and T is the current temperature step during the annealing process. The main reason to accept a move with worse cost is to avoid the algorithm from being stuck in a local minimum. The temperature of the annealing process is gradually reduced (analogous to the physical process of annealing), and this is referred to as cooling. The manner in which the temperature of annealing is reduced as the combinatorial optimization algorithm advances is referred to as the cooling schedule. Since the cooling schedule directly affects the runtime and quality of results, a major part of using the Simulated Annealing algorithm is to tune the cooling schedule to suit the particular problem at hand. The initial temperature is chosen such that a high number of moves (say 99%) are accepted at the start of the algorithm.

It is important to understand that the units of the cost weighting coefficients are similar to the cost factors they weight. Typically however, to keep these factors ratio-less, the cost

factors are auto-normalized, for example by dividing the current area A of the floorplan, with another value A_{norm}, where A_{norm} could be the area of the floorplan from the previously accepted move, as done in Parquet, or the best area of the floorplan obtained so far. In such a case, with auto-normalized cost factors, $\alpha + \beta + \gamma = 1.0$

Another important part of the simulated annealing algorithm is how it explores the solution space at each temperature step. The idea is to perturb the current floorplan to a new state for which the floorplan cost can be reevaluated and accepted or rejected as described above. For the floorplanning problem, these state transitions are brought about by what are called floorplan moves. This can be accomplished by the swapping of a pair of blocks, the rotation of a block, changing its aspect ratio and depending on the floorplan representation, other moves which perturb the floorplan. A key component of the efficacy of a floorplanning algorithm is therefore the floorplan representation, because it dictates the ease with which different moves can be explored, and the efficiency with which the new floorplan cost can be reevaluated. For example, in the sequence pair representation, exchanging two blocks in a sequence pair can result in a completely different layout. Evaluation of this layout may take $O(n^2)$ time as discussed previously. The authors in [8] use the slack of the block in a floorplan to guide the moves via the sequence pair. It can be shown, as in [49], that the simulated annealing algorithm can converge to a global optimum with a probability of 1.0 as the annealing schedule is extended. This however is not practical in most problem domains including floorlpanning. The floorplanner therefore terminates after a fixed time or certain predefined termination criterion are met.

3.5 Microarchitecture Subthreshold Leakage Reduction Via Static Floorplanning

The rest of this chapter discusses our first contribution to reduce the on-chip leakage power consumed in a digital design. Our main aim is to take into account the critical dependence

of subthreshold leakage on temperature, and simultaneously try to tackle both factors on a silicon die. The focus of our work is mainly microarchitecture systems. This is because for such systems, the power dissipation is very high, exacerbating thermal issues. Over the recent years chip makers like Intel® and Advanced Micro Devices® have been trying to cope with the thermal dissipation of the design by using thermal sensors that detect overheating and correspondingly taking action to reduce the chip temperature. One solution to reduce the on-chip thermal power is to use a better package or heat sink, which can dissipate the heat more effectively. However, such a technique is not very practical because it increases the total cost of the system. Another technique in use is better cooling systems, which likewise suffers from the drawback of increased costs.

To help reduce the on-chip temperature for microarchitecture designs, over the recent years microarchitecture floorplanning has gained increased importance. The main idea is to exploit the lateral conduction of the silicon die to reduce the temperature of the core blocks in the design. These are often referred to as hotspots. In [52] the authors present HotFloorplan, a floorplanning tool to reduce the maximum on-chip temperature. The silicon die is first modeled as a network of thermal resistors using the well known microarchitecture thermal analysis tool HotSpot [31] to compute the steady state temperature of each block in the floorplan. Like the Parquet floorplanner described in Section 3.4.2, HotFloorplan uses the Simulated Annealing algorithm. At each step of the floorplanning process, HotSpot is used to model and compute the maximum temperature of the floorplan, which is one of the cost factors for the Simulated Annealing algorithm. The authors in [29] perform a similar computation but instead of detailed thermal models, they use block power density as a measure of the temperature. The idea is that the higher the power density of a block, the greater will be its temperature. This is true if most of the heat is transferred vertically to the heat sink. However, in lieu of the fact that heat is also dissipated by a module laterally to its neighbors, this assumption is not accurate.

Although reducing the maximum on-chip temperature does play an important role in reducing packaging costs, it does not directly tackle the problem of leakage reduction. This is because a physically larger module having a lower temperature than a physically smaller module having a higher temperature may dissipate more leakage power. This is very possible in current day microprocessors which employ large caches to improve performance. Moreover, the phenomenon of thermal runaway cannot be addressed directly using such a cost formulation. The authors in [63] explore different architectures via floorplanning to achieve a given performance objective with certain thermal constraints. The work in [16] presents a clustered microarchitecture wherein instruction scheduling is applied to control the average temperature of the chip, thereby also targeting leakage reduction. Although all these works use temperature in their floorplanning optimization objective, they do not employ accurate models of subthreshold leakage dependence on temperature, which could pose a serious limitation given the importance of leakage power dissipation in future technologies.

Our work attempts to reduce the subthreshold leakage power in a microarchitecture, considering its exponential dependence on temperature, using floorplanning. Our main contribution is to derive a formulation to capture the rise in leakage power of a given floorplan in order to obviate the need to perform time consuming transient simulations during the floorplanning process. Our leakage formulation is independent of any model used to describe leakage power dissipation in a transistor. We call this the **transient** model. A simpler formulation dependent on the leakage power model is also explored and is shown to have good fidelity with detailed simulations for a given floorplan. This is called the **simple** model. Lastly, we model whitespace in the floorplan to analyze its impact on reducing leakage power.

The rest of this section is organized as follows. In Section 3.6 we explain the overall flow of our leakage reduction methodology and the tools used for thermal simulation and give the

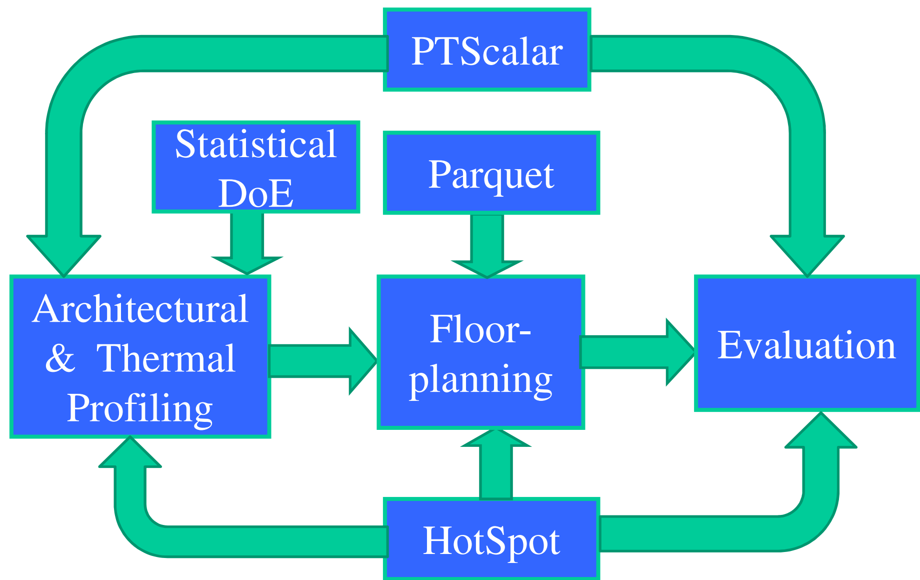


Figure 3.4: Overall flow of our leakage reduction and evaluation methodology.

transistor leakage model used for accurate computation of leakage power. Section 3.7 details our leakage aware floorplanner deriving the transient formulation for leakage power, a simpler model dependent formulation and the whitespace modeling algorithm. Section 3.8 provides experimentation results followed by the chapter conclusion in Section 3.9.

3.6 The Flow of Our Method

Our floorplanning framework revolves around the non-slicing floorplanner, Parquet, described in Section 3.4.2. The overall flow of our methodology is shown in Figure 3.4, with a brief description given below.

- **Thermal profiling** using detailed instruction level simulations to obtain average dynamic and leakage power numbers for the different microarchitecture blocks.
- **Architectural profiling** with statistical Design of Experiments (DoE) to reduce the number of simulations and gauge the impact of microarchitectural busses on performance [47]. We compute factor weights for each critical microarchitecture bus.

- **Floorplanning** using Parquet to include leakage and performance costs. The input to our floorplanner consists of the profiled average powers and factor weights computed above. Temperature is computed using the **block** model in HotSpot. The leakage is computed according to two formulations

  - Leakage power model independent **transient** formulation: This captures the interdependence of leakage on temperature without performing expensive transient simulations during the floorplanning phase.
  - Leakage power model dependent **simple** formulation: This is derived from the transient formulation and depends on the leakage power model used.

- **Evaluation** of the optimized floorplan using detailed instruction level transient simulations and the HotSpot **grid** model to compute the leakage power.

The following sections elaborate on each of these steps in the overall flow.

3.6.1 Architecture Profiling and Thermal Modeling

We run our leakage reduction formulation on a microarchitecture based on the Alpha 21264 design scaled down to $130nm$. A compacted floorplan of the core blocks in this design is shown in Figure 3.5. The configuration details for each block of the microarchitecture are shown in Table 3.6.1. In our formulation, only the core blocks are floorplanned and the L2 cache is wrapped around the core so as to obtain a square die of side $15.9mm$.

Prior to performing the actual floorplanning however, we profile the microarchitecture for its power consumption and performance. To do this, we compute the average dynamic power consumption of each block using the PTScalar framework described in [39], which is based on the very popular microarchitecture simulation tool SimpleScalar [13]. The idea is similar to the WATTCH [12] power profiler, wherein during each simulation cycle the

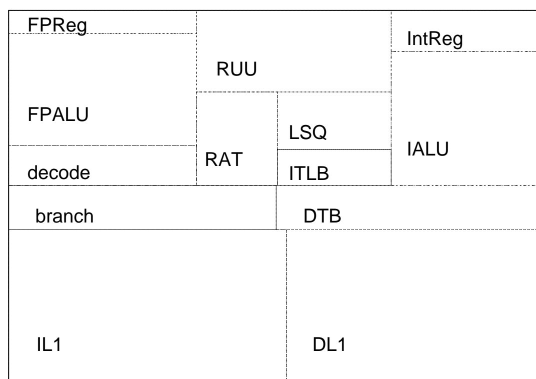


Figure 3.5: Compacted floorplan for the core of the Alpha 21264 microarchitecture. The L2 cache surrounding the core is not shown.

Block Name	Description
Register Update Unit(RUU) size	16 Instructions
Load Store Queue(LSQ) size	8 Instructions
Fetch queue size	4 Instructions
Issue width	4 Instructions / cycle
Decode width	4 Instructions / cycle
Commit width	4 Instructions / cycle
Integer ALU (IALU)	3 adder and 1 multiplier unit
Floating point ALU (FPALU)	1 adder and 1 multiplier unit
Branch predictor	Combined, Bimodal with 2K entries and 2 level with 1K entries and 8-bit history
Instruction TLB (ITLB)	64 entry fully associative with 30 cycle miss penalty
Data TLB (DTB)	128 entry fully associative with 30 cycle miss penalty
Branch Target Buffer	512 entries and 4-way associative
L1 Data cache	16KB, 4-way set-associative, with 32B blocks and 1 cycle latency
L1 Instruction cache	16KB, direct mapped, with 32B blocks and 1 cycle latency
L2 cache	256KB, 4-way set-associative, with 64B blocks and 6 cycle latency

Table 3.1: Microarchitecture configuration

number of accesses to every block in the microarchitecture is monitored to compute its activity, which is then used to calculate its dynamic power.

Such a power profiling is conducted for a set of 10 Spec2000 [57] benchmarks consisting of 5 integer (bzip2, gcc, gzip, crafty, twolf) and 5 floating point benchmarks (mesa, art, mgrid, swim, equake). The benchmarks are run for 500 million instructions after an initial architectural warm up period of a 100 million instructions in order to avoid any cold start effects. We also allow an additional thermal warmup of 200 million instructions before monitoring any temperature dependent statistics. The main purpose of these simulations is to arrive at floorplan (i.e., temperature) independent average powers dissipated by each of the blocks assuming the instruction mix covers a wide range of applications.

We also thermally monitor the simulations, and perform transient simulations for leakage power computation, by integrating Equation 3.13 over fixed size simulation intervals using HotSpot (described in detail in Section 3.3.2). Since we need to be accurate in our computation of leakage, we use the **grid** model. The leakage numbers are used as weighting constants when computing the leakage penalty using the **simple** model because different types of blocks (for instance logic vs memory) having the same area and temperature consume different amounts of leakage power. We will elaborate more on this in Section 3.7.

3.6.2 Performance Modeling

As discussed in Section 3.4, one measure of performance during floorplanning is the widely used half perimeter wirelength metric (HPWL). However, as pointed out in [52] and [47], since this metric lacks the notion of criticality of a wire, floorplans with smaller total wirelength may have a worse performance than those with larger HWPL. For example, busses connecting the register update unit to the integer unit are very critical to the overall performance of the microarchitecture and hence have a large criticality. In [52], the authors conduct a series of profiling runs with different latencies on certain critical busses to gauge

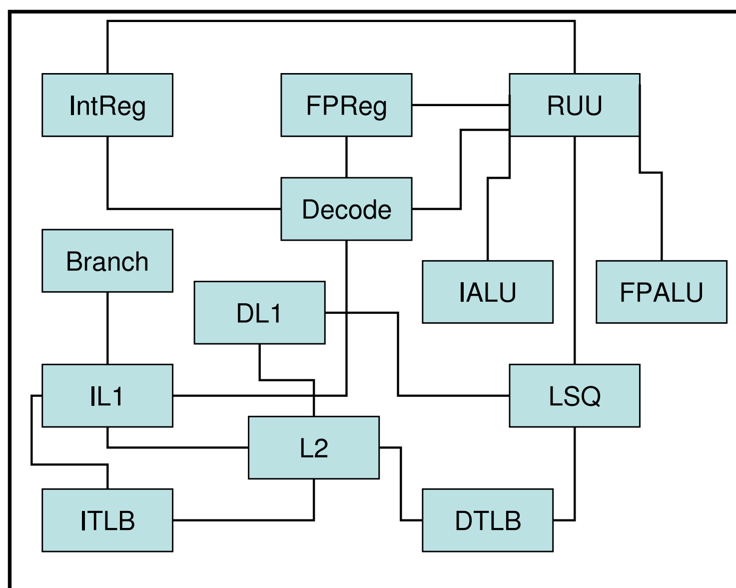


Figure 3.6: Busses modeled for performance evaluation. The fetch unit is considered part of IL1.

their impact on the performance. A linear regression analysis is then performed to obtain weights for each of the busses to be used during floorplanning for evaluating the impact of the floorplan on performance.

As leakage reduction tends to create whitespace between blocks in a floorplan, which in turn hurts performance, we wire-pipeline long busses to avoid large performance penalties in the microarchitecture. Towards this goal the authors in [20] consider both the architectural and physical design spaces in their floorplanning tool to devise an optimal floorplan meeting the input constraints. The work in [47] uses a statistical Design of Experiments (DOE) approach so as to avoid running a prohibitively large number of experiments to gauge the impact of each microarchitecture bus on performance. We use the weights in [47] (also called factors) obtained using the DOE approach for critical busses in the floorplan to evaluate their impact on performance. Briefly, the authors use a 2 level, resolution III fractional factorial design wherein busses are assigned latencies from 0 to their maximum value. The SimpleScalar microarchitecture simulator [13] is modified to incorporate the ad-

ditional latencies on the wires and a set of experiments performed to determine the weights of the busses. For our purposes we consider the 17 critical busses as shown in Figure 3.6. Given N factors, the number of experiments run is the nearest highest power of 2, i.e., for 17 factors the DoE based approach needs to perform 32 experiments. Multiple factor interactions are ignored in our work since they are negligible compared to single factor interactions.

The main assumption made here is that these factors affect the system response monotonically. Hence while computing the performance penalty, we weight the latency of each bus during floorplanning with its corresponding factor. The wirelength term during floorplanning is therefore computed as the sum of weighted latencies,

$$HPWL_W = \sum_{i=1}^{n} w_i \cdot l_i \tag{3.18}$$

where, w_i is the factor weight and l_i is the latency of bus i. The latency of a particular wire is determined based on the microarchitecture frequency of operation fed to the floorplanner. It must be emphasized that our primary contribution in this work is to invent and evaluate a methodology to reduce leakage current at the microarchitecture level via floorplanning by avoiding expensive transient simulations. Although the main thrust of this work is not in modeling performance, we intend to show savings in leakage through floorplanning with a tolerable loss in performance.

3.6.3 Floorplanning

Our floorplanning framework uses the Parquet floorplanner described in detail in Section 3.4.2. Since Parquet is based on simulated annealing, it is relatively easy to modify the object function to account for the subthreshold leakage power. In order to thermally model each candidate floorplan, we use HotSpot (described in Section 3.3.2) for thermal model-

ing in our overall flow. Due to the computationally intensive nature of the floorplanning algorithm, we use the **block** model of HotSpot for thermal modeling inside the floorplan optimization loop for computational efficiency.

At each floorplanning step, Parquet constructs a new non-slicing floorplan which is evaluated according to a given cost function. The cost function consists of area, performance and leakage terms to be described in Section 3.7.

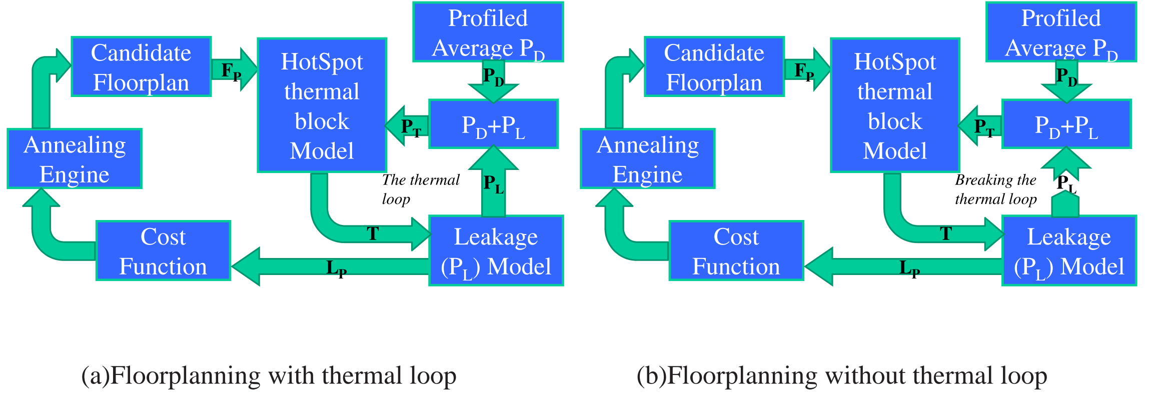
3.6.4 Evaluation

Once we obtain an optimized floorplan, the evaluation phase is conducted in a manner similar to the thermal profiling phase and consists of running detailed simulations on all 10 benchmarks for 500 million instructions after an architectural warmup of 100 million instructions and a thermal warmup of 200 million instructions. Similar to the profiling phase, we use HotSpot's *grid* model for better accuracy at the cost of increased runtime.

3.7 Floorplanning Details

We use Parquet in fixed-outline mode, implying that the area of the floorplan is to be constrained to an upper limit. The limit is computed by specifying a bound on the total whitespace acceptable in the floorplan along with the total area of the microarchitectural blocks. The individual blocks can have aspect ratios between $AR_{min} = 1.0$ and $AR_{max} = 3.0$, whereas the overall microarchitecture core aspect ratio is provided as an input to the floorplanner. Whitespace in the floorplan is defined as any portion of the floorplan not occupied by the blocks. Most floorplanners try to minimize whitespace (also referred to as deadspace) by minimizing the overall area of the floorplan. Whitespaces in a floorplan can typically be used, among other purposes, for routing global interconnects, placing decoupling capacitors and voltage regulators on the chip.

Besides the regular slack based moves in Parquet, we supplement it with two additional



(a)Floorplanning with thermal loop (b)Floorplanning without thermal loop

Figure 3.7: Illustration of the details of the floorplanning process. Figure 3.7(b) shows how we break the thermal loop of Figure 3.7(a) using our own leakage computation technique.

temperature-specific moves to the annealing move-set, to be presented in Section 3.7.3. The annealing cost function used is similar to the annealing cost function of Parquet in Equation 3.17. Apart from the overall area, half-perimeter wirelength and aspect ratio, we add the leakage cost of the floorplan as an additional cost item, shown below.

$$\begin{aligned} C &= \alpha \cdot A + \beta \cdot HPWL_W + \\ &= \gamma \cdot L_P + (1-\alpha-\beta-\gamma) \cdot AR \end{aligned} \tag{3.19}$$

where A is the summation of the width and height of the floorplan, AR is the aspect ratio, $HPWL_W$ is the weighted sum of bus latencies as given by Equation 3.18 and L_P is a term to capture the leakage power dependence on temperature. Sections 3.7.1 and 3.7.2 show how L_P is calculated. The coefficients α, β and γ are used to weight each of the auto-normalized cost terms.

At each step of the floorplanning process, a candidate floorplan is generated and its thermal model evaluated using HotSpot's **block** model to compute the leakage cost, L_P. The total power is the summation of the leakage power of the blocks and the dynamic power, obtained from the profiling phase described in Section 3.6.1. This is illustrated in Figure 3.7(a). The next two sections describe in detail our method of evaluating the leakage

cost terms used during floorplanning.

3.7.1 Transient Formulation Independent of Leakage Power Model

As mentioned previously, each block in the floorplan is modeled as a node in the thermal network given by Equation 3.16, where ideally, the power dissipated at any node in the network is a constant. However, considering the dependence of leakage power on temperature, the total power dissipated at any node is now a function of the temperature, T. Moreover, since the temperature evolves over time, the leakage power also varies with respect to time. One method of solving this system is to assume a constant power P_t at time t for a given time step Δt, integrate Equation 3.13 over that interval and compute the temperature $T + \Delta T$ at the new time $t + \Delta t$. This can then be used to update the power at time step $P_{t+\Delta t}$ and we repeat the process till convergence is achieved. This is referred to as the thermal loop in 3.7(a). Assuming that the blocks have a fixed dynamic power consumption, the steady state temperature profile can be computed and the overall leakage power at this profile evaluated. Performing such a transient analysis however is highly time consuming at each step of floorplanning and is only done during the thermal profiling and leakage evaluation phase as described in Sections 3.6.1 and 3.6.4. We avoid such expensive iterative computations with an approximation to the leakage power, to be described next.

Rewriting Equation 3.13 with the temperature dependence of power gives

$$\frac{dT}{dt} = C^{-1} P(T) - C^{-1} G\, T \tag{3.20}$$

In the above equation $P(T)$ is the total power dissipated at temperature T. Writing the finite difference approximation for the above equation for time Δt we obtain

$$\begin{aligned} \frac{\Delta T}{\Delta t} &= C^{-1}\, P(T_1) - C^{-1} G\, T_1 \\ \Rightarrow T_2 &= T_1 + (C^{-1}\, P(T_1) - C^{-1} G\, T_1)\, \Delta t \end{aligned} \tag{3.21}$$

where T_1 and T_2 are the block temperatures at times t and $t + \Delta t$ respectively. Expanding $P(T)$ in terms of its first order Taylor series assuming a small increment in temperature ΔT we get

$$\begin{aligned} P(T + \Delta T) &= P(T) + \frac{dP}{dT} \cdot \Delta T \\ \Rightarrow P(T_2) &= P(T_1) + \left.\frac{dP}{dT}\right|_{T_1} (T_2 - T_1) \end{aligned} \tag{3.22}$$

Combining Equations 3.21 and 3.22 and writing them in matrix form we obtain

$$P(T_2) = P(T_1) + J\Big|_{T_1} C^{-1} \Big[P(T_1) - G\, T_1 \Big]\, \Delta t \tag{3.23}$$

where $J\Big|_{T_1}$ is the Jacobian of power P with respect to temperature T evaluated at T_1. Let T_1 be the steady state temperature attained with blocks dissipating their average dynamic power P_D, and without any leakage effect taken into account. As shown in Equations 3.14 and 3.16, we can write,

$$\begin{aligned} G \quad T_1 &= P_D \\ T_1 &= R \quad P_D \end{aligned} \tag{3.24}$$

where $R = G^{-1}$ is the transfer thermal resistance matrix given by Equation 3.15. Consider the leakage power $P_L(T_1)$ dissipated at temperature T_1, given below,

$$P_L(T_1) = P(T_1) - P_D \tag{3.25}$$

Using this in Equation 3.23 we obtain

$$
\begin{aligned}
\frac{P_L(T_2)-P_L(T_1)}{\Delta t} &= J\Big|_{T_1} C^{-1}\left[P(T_1) - G\,T_1\right] \\
\Rightarrow \frac{\Delta P_L}{\Delta t} &= J\Big|_{T_1} C^{-1}\,P_L(T_1)
\end{aligned}
\tag{3.26}
$$

Equation 3.26 gives us the "temperature dependent" increase in leakage power over time Δt given the average dynamic power dissipation P_D. We call this the **transient** formulation of leakage power and use this as leakage power, L_P, in Equation 3.19. Pictorially, this technique can be seen as breaking the thermal loop of Figure 3.7(a) and is shown in Figure 3.7(b). This formulation is unable to capture a highly non-linear rise in leakage power, for example, with temperatures high enough to facilitate thermal runaway. However, given the large time constants of thermal power dissipation compared to the frequency of operation of the chip, we can reasonably assume a linear increment in leakage power given by the Taylor expansion. Since we try to minimize Equation 3.26 during floorplanning, this also reduces the temperature of the blocks helping to avoid thermal runaway. Moreover, as was shown in [39], thermal runaway relates the increment in power dissipation to the package heat removal ability. Equation 3.26 gives us the increment in power which could be used in rejecting floorplans with thermal runaway. Performing transient simulations would require an additional runtime overhead of $O(n \cdot N^2)$ where n is the number of iterations to converge and N is the total number of blocks in the floorplan. It must be emphasized that this formulation is independent of the subthreshold leakage model used.

3.7.2 Simple Formulation Dependent on Leakage Power Model

For our microarchitecture floorplanning purpose, we use the subthreshold leakage power model from [39] as described below,

$$P_L(T) \;=\; A\,T^2\,e^{-\left(\frac{\alpha\;V_{dd}+\beta}{T}\right)} \tag{3.27}$$

where $P_L(T)$ is the leakage power dissipated at temperature T, A is proportional to the area of the block, and captures the number of transistors that leak current in the block, V_{dd} is the supply voltage magnitude and α and β are empirical constants whose values depend on whether the block is a functional unit such as an arithmetic ALU, or a memory unit such as a cache. These constants were derived by performing curve fitting by running spice simulations for different circuits. For more detail on the derivation of the model we refer the reader to [39]. It can be seen that this equation resembles the BSIM model approximation given by Equation 3.8, in that the temperature dependence of the leakage current is captured accurately. From Section 3.7.1, we can now plug in Equation 3.27 in the transient formulation of Equation 3.26 to obtain

$$\left.\frac{\Delta P_L}{\Delta t}\right|_T \;=\; C^{-1}A^2T^2e^{-2\left(\frac{\alpha\;V_{dd}+\beta}{T}\right)}\Big[2T+\alpha V_{dd}+\beta\Big] \tag{3.28}$$

The thermal capacitance(see Equation 3.13) of a block is directly proportional to its area. Expanding the exponential in its Taylor series and observing the dominant term gives us a cubic polynomial in T. We therefore use the cubic formulation shown below

$$L_P \;=\; k\,A\,T^3 \tag{3.29}$$

where k is a proportionality constant obtained using the average leakage power dissipation of the blocks as computed in the profiling phase (see Section 3.6.1. Intuitively the model makes sense since larger blocks, i.e., more leaky transistors, with higher temperature dissipate more leakage power. The proportionality constant accounts for the fact that different types of blocks having the same area dissipate different amounts of leakage power at the

same temperature. Equation (3.29) offers the advantage that it is easier to compute compared to 3.26, which incurs an additional overhead in computing the leakage powers and the Jacobian for each block at temperature T. Note that term T is calculated using HotSpot and does indeed consider the interaction between neighboring modules.

3.7.3 Heuristic Moves

We perform two additional kinds of moves in addition to those provided by Parquet described below.

1. **Temperature Move:** We randomly choose among the top 5 hottest and coolest blocks in the floorplan and swap them. The idea behind this move is to alleviate the thermal impact of neighboring blocks on the hottest block in the floorplan.

2. **Power Move:** Consider the steady state transfer thermal resistance matrix in Equation (3.15). As mentioned before, element R_{ij} corresponds to the increase in temperature of block i for a unit change in power of block j, i.e., $R_{ij} \cdot P_j$ gives us the influence of block j on block i, where P_j is the power dissipated by block j. We randomly choose a block i among the top 5 most leaky blocks and place it next to the block exerting the least influence on block i, i.e., the block j with the minimum $R_{ij} \cdot P_j$ value.

3.7.4 Whitespace Modeling

As described in Section 3.4.2, Parquet uses a non-slicing sequence pair representation to perform the floorplanning. Since whitespaces are regions of the silicon die without any active devices, they can potentially influence the temperature distribution of the chip. Hence, they need to be accounted for during the floorplanning process. In [29], which also uses Parquet, the authors increase the area of certain blocks to fill the whitespace. It is not clear to us as to how this could be done in a non-slicing floorplan. Although our work does not

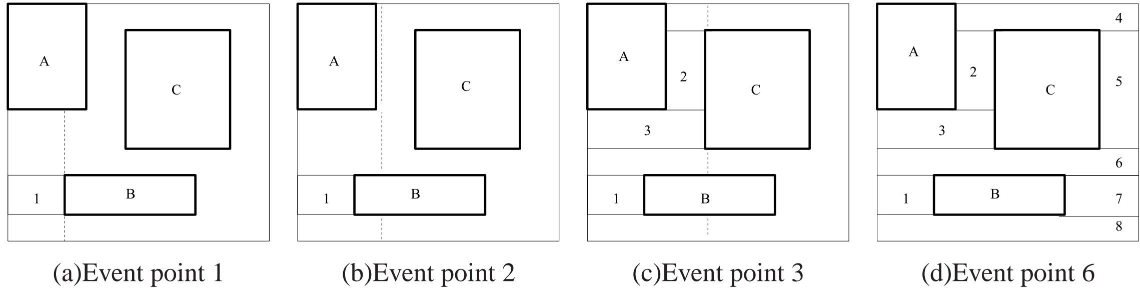
Algorithm 9 $ComputeWhitespace(Blocks, N, W, H)$

1: **for** $i = 1$ to N **do**
2: Store the left end interval $I_L(x_1)$ of $Block(i)$ in queue Q_L
3: Store the right end interval $I_R(x_2)$ of $Block(i)$ in queue Q_R
4: **end for**
5: Sort Q_L and Q_R according to their interval event points
6: Insert dummy interval $I_L(0) = [0, H]$ in the interval tree $IntTree$
7: Push dummy interval $I_R(W) = [0, H]$ in Q_L
8: **while** $Q_L \neq empty$ **do**
9: $I_L = Q_L.pop()$; $I_R = Q_R.pop()$
10: I is the earlier occurring interval. In case of ties choose I_R
11: **if** $I = I_R$ **then**
12: Insert interval I_R in $IntTree$; $I_R = Q_R.pop()$
13: **else**
14: Store list of intervals in $IntTree$ that overlap I_L in O
15: **for all** Interval I_O in O **do**
16: Create new dead block with $x_1 =$ event point of I_O, $x_2 =$ event point of I_L and $[y_1, y_2] =$ interval overlap between I_O and I_L
17: Create new intervals from non-overlapping section(s) of I_O
18: Delete I_O from $IntTree$
19: **end for**
20: $I_L = Q_L.pop()$
21: **end if**
22: **end while**

directly exploit the available whitespace during fixed die floorplanning, we need to model whitespace for the annealer engine to evaluate the temperature distribution and thereby the leakage cost of the floorplan accurately. For this purpose we model whitespaces as additional blocks in the floorplan with zero power dissipation. Algorithm 9 used for computing whitespaces is based on the sweep-line paradigm using a balanced interval tree [21] and takes as input the blocks in the floorplan, their total number, N, and the width, W, and height, H, of the floorplan.

Given a block in the floorplan with lower left and top right corners (x_1, y_1) and (x_2, y_2) respectively, we represent it by left end interval $I_L(x_1) = [y_1, y_2]$ and right end interval $I_R(x_2) = [y_1, y_2]$. Here x_1 and x_2 are the interval event points of I_L and I_R respectively.

We sweep the floorplan in the horizontal direction from 0 to W. Every time we encounter



(a)Event point 1 (b)Event point 2 (c)Event point 3 (d)Event point 6

Figure 3.8: Execution of the sweep-line on a toy example. Disjoint dotted lines are whitespace intervals showing the advance of the sweep-line at event points. A, B, C are blocks and 1-8 are whitespaces filled in. Two event points between 3.8(c) and 3.8(d) are not shown.

a right end interval, we start a new whitespace block by inserting it into the interval tree. We call this a whitespace interval. On encountering a left end interval, we check all current whitespace intervals in the interval tree that overlap it. A new whitespace block is created having a horizontal span from the event point of the whitespace interval to the event point of the left end interval and a vertical span equal to the overlap of the two intervals. For all non-overlapping portions of the whitespace interval, we create new whitespace intervals and insert them into the interval tree. The algorithm terminates when the left end interval queue is empty.

Figure 3.8 shows the sweep-line execution on a small example. Event point 1 corresponding to the left end interval of block B, creates whitespace block 1 and two new whitespace intervals above and below block B, seen in Figure 3.8(a). Event point 2, corresponding to the right end interval of block A, creates a new whitespace interval to the right of block A, illustrated in Figure 3.8(b). At event point 3, corresponding to the left end interval of block C, we find overlaps creating whitespace blocks 2 and 3 and two new whitespace intervals, one above block C and the other between blocks B and C, shown in Figure 3.8(c). The final floorplan after the sweep-line algorithm terminates is shown in Figure 3.8(d). The runtime complexity of our algorithm is $O(NlogN + K)$ where N is the number of blocks in the floorplan and K is the number of whitespace blocks (which is at most linear in N).

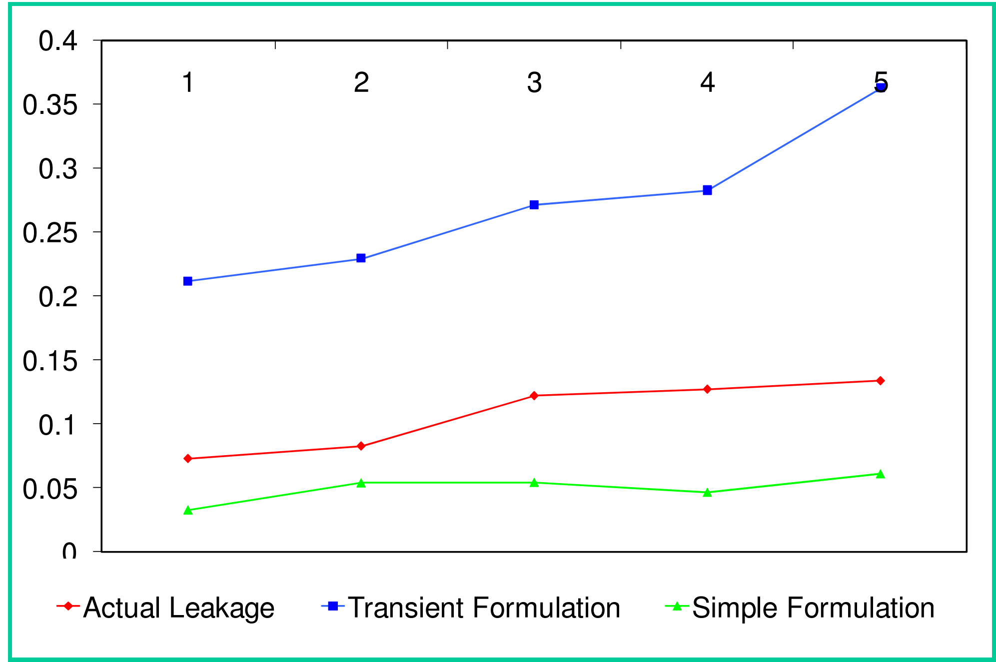


Figure 3.9: Normalized fidelity plots for Equation 3.26 'transient', Equation 3.29 'estimated' and accurate transient simulations 'accurate'. The leakage values are normalized to a chosen base floorplan and are numbered in increasing value of their 'accurate' leakage for better visualization.

3.8 Results

We ran our experiments on an Intel® 3.2 GHz CPU with 2GB RAM on 10 Spec2000 [57] benchmarks. To evaluate the leakage power of each floorplan, detailed instruction level transient simulations were performed for all 10 benchmarks for 500 million instructions after architectural and thermal warmups of 100 million and 200 million instructions respectively. Each run of Parquet chooses the best among 10 generated floorplans.

3.8.1 Fidelity of Our Formulation

To compute the fidelity of our leakage cost function used by the floorplanner in evaluating different floorplans in terms of their overall leakage cost, we generate a set of 5 floorplans (optimized with different leakage weights) and compute their leakage according to Equations 3.26 and 3.29. We only chose to compare with accurate leakage power simulations

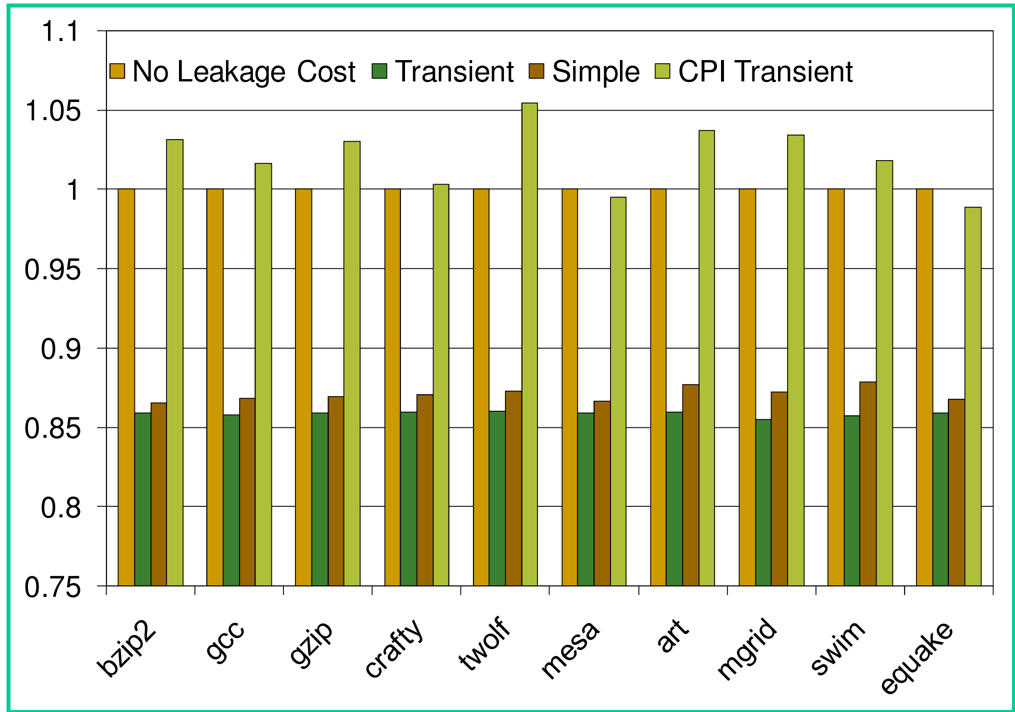


Figure 3.10: Normalized graphs of the leakage power and CPI in comparison with a floorplan optimized without leakage called Base Case. Y-axis begins at 0.75.

of benchmark bzip2 due to its higher average chip temperature and leakage power compared to the other benchmarks. We call the leakage obtained through these formulations the transient, estimated and actual leakage respectively. Figure 3.9 shows normalized plots of the three formulations with respect to the leakage of floorplan 1 for each formulation. As can be seen, all three curves track each other well with the transient formulation having high sensitivity to floorplans with different leakage. Moreover, the plot also shows that we can use our simple estimation method with low computational complexity and yet achieve similar results compared to the transient formulation.

3.8.2 Leakage Savings

Figure 3.10 compares the gains in leakage of two floorplans, one with leakage optimization, called the transient case, and the other without leakage optimization, called the base case. We compare the leakage and performance of these two schemes, in terms of the cycles per

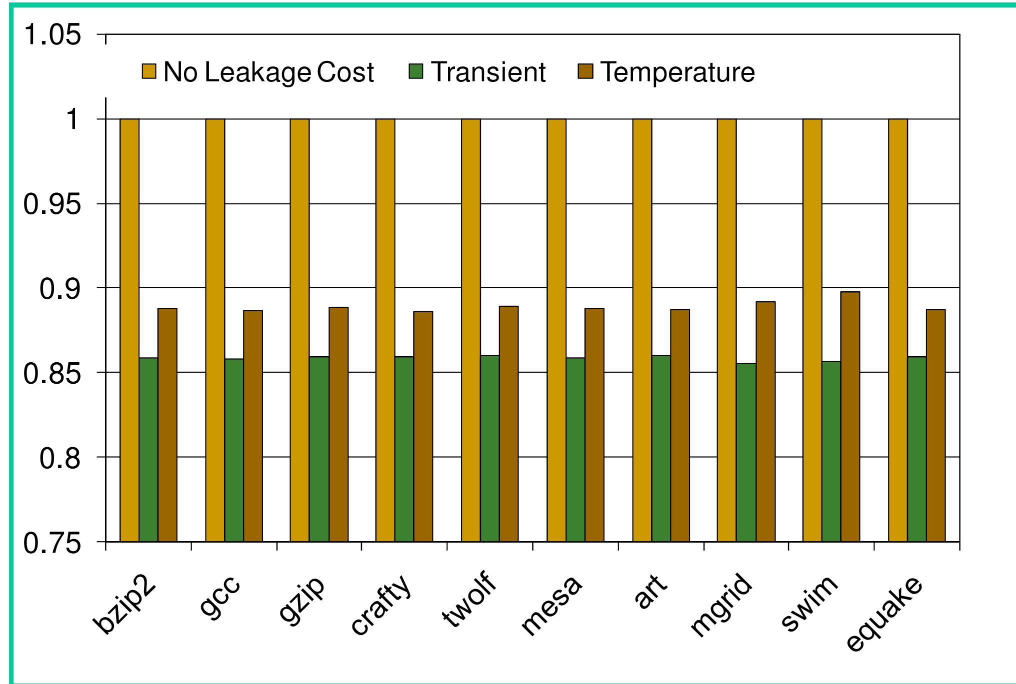


Figure 3.11: Normalized graphs of leakage power with maximum temperature and transient formulation as objective functions compared to a base floorplan without leakage optimization. Y-axis begins at 0.75.

instruction or CPI, for each of the 10 benchmarks.

From the figure, we observe that our leakage optimization formulation results in 15% savings on an average with a performance penalty of under 5%. Both floorplans were constrained to have a maximum whitespace of 10%. We also observe from the figure that optimizing the floorplan with the simple formulation from Equation 3.29 (called the simple case) results in similar leakage savings.

3.8.3 Maximum Temperature Reduction

Similar to [52] and [29], most works on thermal-aware floorplanning aim at minimizing the maximum temperature of the floorplan. As mentioned previously, the same cannot be applied to the case of leakage minimization since large blocks like the data caches although have a lower temperature could have more leakage due to their larger area. Such

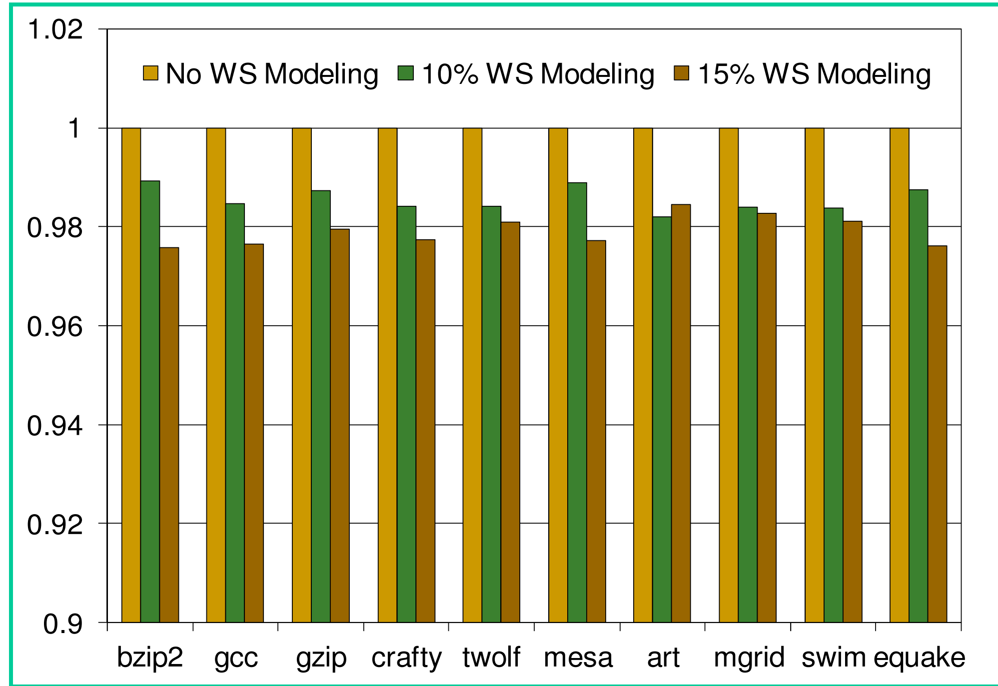


Figure 3.12: Normalized graphs showing the impact of WS on leakage reduction. Bar 1, 2 and 3 are leakage with no WS modeling, 10% WS and 15% WS modeling during floorplan optimization respectively. Y-axis begins at 0.75.

an effect could be the result of, for instance, placing high temperature modules next to the large leaky modules. Figure 3.11 compares the leakage savings when the thermal objective function is to reduce the maximum chip temperature as opposed to our leakage objective in Equation 3.26. The figure shows that temperature reduction results in floorplans with an average leakage about 4% more than the case with leakage optimization. Additionally, as discussed in Section 3.7.1, we have no way of detecting thermal runaway in the floorplan with such an objective function. It must be noted that we observed similar reductions in peak temperature of these two formulations compared to the floorplan optimized only for performance, indicating that our formulation in Equation 3.26 is effective in reducing the peak chip temperature.

3.8.4 Whitespace Considerations

We perform floorplanning with 10% and 15% whitespace(WS) and compute the leakage using transient simulations. Figure 3.12 shows the results. It also shows the impact of not modeling whitespace (No WS Modeling) during floorplanning. Negligible differences are observed in leakage savings of floorplans with 10% and 15% whitespace. Moreover, modeling whitespace (as blocks with zero dynamic power dissipation) has little impact on the overall leakage savings. We believe the primary reason for such behavior is because the simulations were performed with an ambient temperature of $40°C$ which is an optimistic estimate of the temperature inside a CPU tower. Our simulations resulted in maximum block temperatures of about a $100°C$. Due to its exponential dependence, small differences in temperature at higher temperature ranges result in larger changes in leakage power than large differences in temperature at lower temperature ranges. This also points to the important fact that increased whitespace in the floorplan yields diminishing leakage power savings when the average chip temperature is not very high.

3.8.5 Runtimes

Table 3.2 lists the runtimes of various schemes employed in the floorplanning process. Scheme A only models performance. Scheme B models leakage of the various blocks in the floorplan using Equation 3.26 but no consideration is given to whitespaces. Schemes C and D also model leakage but unlike B they partition the surrounding L2 cache into smaller blocks for better thermal modeling accuracy. Scheme D additionally includes whitespace modeling. As observed in Section 3.7.4, the difference between C and D in terms of results on leakage savings is negligible. Comparing B and C we found that partitioning the L2 cache into smaller blocks does not impact the overall results but runtime increases considerably due to the increased overhead of thermal modeling and leakage computations. However, considering the fact that L2 leakage increases with future technology nodes, schemes that improve its thermal modeling will be essential to accurate leakage cost computation.

Floorplan scheme	Leakage Modeled	L2 cache partitioned	Whitespace modeled	Runtime ($secs$)
A	No	No	No	0.24
B	Yes	No	No	19
C	Yes	Yes	No	235
D	Yes	Yes	Yes	520

Table 3.2: Runtimes for different floorplanning schemes

3.9 Conclusion

This chapter presents new formulations for subthreshold leakage power reduction enabled via floorplanning. Our results show an improvement on average of 15% compared to the case when no leakage power optimization is performed, with a tolerable loss in performance. We show that our leakage formulation has good fidelity with transient simulations justifying its use during the floorplanning phase. We also show the effect of increasing whitespace in the floorplan. As future work, we would like to explore floorplanning with different microarchitectures and smaller technology nodes where leakage power dissipated is a larger fraction of the system dynamic power. It may be useful to thermally model only the region of the L2 cache surrounding the core blocks instead of its entirety, since this would reduce runtime and increase accuracy. An interesting problem is that of finding the appropriate amount of whitespace required in the floorplan to extract appreciable leakage savings.

Our leakage formulation is important in the light that it makes the floorplanning cost modeling relatively inexpensive, by avoiding expensive iterative computations to achieve convergence. It also gives us an intuitive measure of the leakage cost. This, as we will see in the next chapter, will be used as a building block for the leakage formulation of the next chapter, which deals with a new technique called task migration, which *actively* tries to manage and control the subthreshold leakage power dissipation and thereby the on-chip temperature of a die.

Chapter 4

Thermal-Aware Floorplanning for Task Migration Enabled Active Sub-threshold Leakage Reduction.

In this chapter, we present a new approach to active sub-threshold leakage reduction using a concept called task migration. The main idea is to replicate a *hot* module in a design so as to actively migrate its computation at regular intervals, reducing the on-chip temperature and thereby the sub-threshold leakage. We observe that choosing which blocks to migrate and their placement in a floorplan is a chicken-and-egg problem. To solve this, we propose a two step floorplanning methodology, wherein, given a base floorplan, we first choose the modules to replicate, and then effectively utilize the deadspaces in it by exploiting the lateral conduction of heat in the floorplan to place a module's replica. With an optimized floorplan, using task migration we obtain an average savings of 29% in the active sub-threshold leakage at the expense of about 6% additional area.

4.1 Introduction

In the sub-100nm era, sub-threshold leakage power becomes a larger fraction of the total power due to lower transistor threshold voltage and increased device density brought about by device scaling [55]. This increase in power density creates what are known as thermal hotspots in the chip, which degrade performance and reduce the lifetime of the chip. More importantly, as discussed in Chapter 3, this exacerbates the problem of sub-threshold leakage (hereby referred also to as leakage) since it is exponentially dependent on the temperature, and could also result in the phenomenon of thermal runaway due to the positive feedback between power and temperature. Due to the finite lateral heat conduction of the silicon substrate, the spatial location of devices plays an important role in dictating the temperature and thereby the overall leakage of the design.

Most digital circuits primarily operate in two modes.

- **Standby mode:** In this mode of operation, circuit blocks do not perform useful computation, for example, a memory unit such as a cache might have a large portion of its silicon inactive at any given time.
- **Active mode:** In this mode of operation, circuits perform their intended computation. For example, an ALU unit is active when computing arithmetic operations.

Circuit designers have tried to leverage the standby mode of operation to reduce subthreshold leakage power [45], using schemes at the transistor level of design. Some of the more common ones are listed below

- **Input vector control** In this method a certain set of input vectors is applied to the inactive blocks on the chip so as to lower their leakage in the standby mode. This technique exploits the well known stack effect [66], which simply put states that the

leakage through two disabled transistors in series is an order of magnitude lower than a single disabled transistor.

- **Multi-threshold CMOS (MTCMOS)** In this technique [44], high threshold voltage (V_t) transistors (often referred to as sleep transistors) are used to serially connect low V_t blocks of the circuit to the power supply rails. In the standby mode of operation, these transistors are shut-off, and produce orders of magnitude lower leakage than a low V_t circuit operating in standby mode on its own. This technique is one of the most popluarly used schemes to reduce sub-threshold leakage power.

- **Dynamic Voltage Scaling (DVS)** This is another popular technique which falls under the general umbrella of power reduction techniques. In this method the operating voltage of the blocks on the chip is varied to reduce leakge power consumption in the standby mode, since power is a product of voltage and current.

- **Variable threshold voltage CMOS (VTCMOS)** This technique was first introduced in [35], wherein body biasing is used to effect a change in threshold voltage and hence a reduction in leakage of the transitors on the chip.

Althouth not exhaustive, this is a fairly representative list of the schemes used to reduce leakage power in a digital design. It is imperative to note however, that each of these schemes is generally applied to circuit blocks in standby mode to reduce leakage, either because they cannot be applied in the active mode due to loss of logic state information (works like [32] apply the MTCMOS scheme to sequential circuits in the standby mode) or because they may cause considerable performance degradation if applied in the active mode. On the other hand, the large reduction in leakage that can be achieved (sometimes orders of magnitude) through techniques like MTCMOS warrant an investigation as to how these can be leveraged easily in the active mode.

Another technique orthogonal to the transistor level schemes for sub-threshold leakage

reduction is using floorplanning to reduce the on-chip temperature profile by exploiting the lateral heat conduction of the silicon substrate. Chapter 3 described this paradim and our work exploiting the lateral heat conduction of the floorplan to reduce its leakage footprint. Although these schemes have the potential to reduce the active leakage, due to their passive nature, they cannot adapt to the operating environmental conditions.

In lieu of the above discussion, this paper attempts to use task migration (TM) between replicated units of a design as a means to directly reduce temperature and therefore the sub-threshold leakage power of a design without adversely affecting its performance. The basic idea is to redistribute the active computation of certain high activity blocks in different parts of the chip at regular intervals to enable a reduction in the active leakage through a reduction in the temperature profile of the chip. Since task migration involves the replication of blocks, it would seem to imply that the overall leakage would increase with such a scheme. This would not be the case however, if such a paradigm relies on *leveraging* the high reduction obtainable by means of a standby leakage reduction technique, to reduce the leakage of the inactive blocks in the design.

Figure 4.1 illustrates the task migration paradigm in concept. Block set A consists of a subset of blocks, which may be those with high activity, called the primary blocks in the design. We replicate these blocks into a set B called the shadow or replica blocks in the design. At periodic intervals, denoted $t_1, t_2, \ldots, t_7$, computation is transfered from the primary block set A to the secondary block set B and vice-versa. The length of such an interval is called the switching or migration interval and denoted as τ. When block set A is active, block set B is put into standby mode, wherein its leakage is considerably reduced by using one of the several standby leakage reduction schemes mentioned above. We transfer activity from one region of the chip to another, reducing the on-chip temperature profile and therefore the active sub-threshold leakage directly. It is important to note that such a scheme to reduce leakage power can be applied over and above all other schemes that are

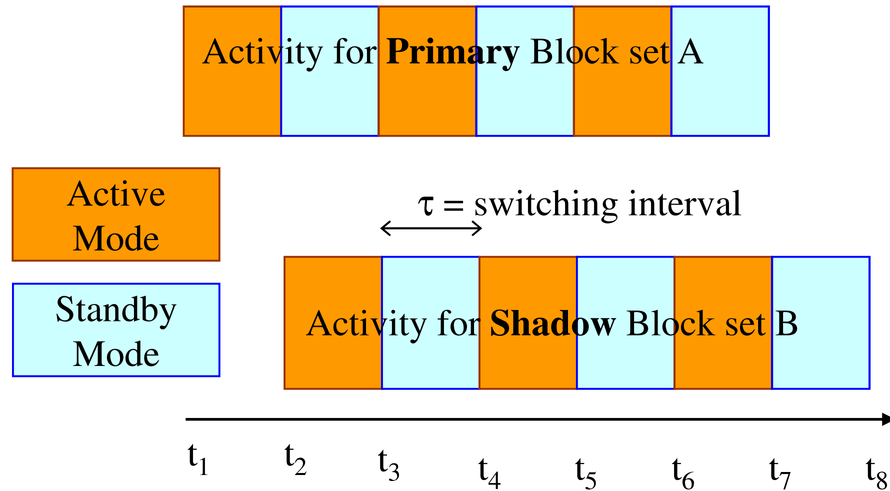


Figure 4.1: Task migration illustrating the activity time-line of the primary, and shadow (or replica) set of blocks in a design.

currently in use today.

The idea of TM is not new. It was first proposed in [30], in which the authors apply the scheme to a set of processor micro-architecture blocks, and use a dynamic voltage scaling technique to obtain better performance for the same maximum chip temperature constraint. Task migration can also be thought of as a Dynamic Thermal Management (DTM) technique, in which a chip is not allowed to reach a critical temperature threshold. DTM schemes generally require the use of thermal sensors to detect a thermal emergency, the condition of crossing the critical temperature threshold, and activate the appropriate thermal management scheme. These are called reactive DTM schemes since activity migration is only performed when a thermal emergency demands it [25, 54]. On the other hand, we propose to use TM as a preventive DTM scheme, in which we migrate activity at regular intervals to prevent temperature buildup. Although easier to implement, unlike reactive schemes which require thermal monitoring, detection and actuation, the disadvantage with preventive schemes is that they may incur an unnecessary performance overhead, swapping

activity even when no thermal emergency is present. However, our main application is not a DTM scheme, but a method to reduce the active leakage of a chip, which is worsening with scaling trends.

Our work, like [30], performs the migration of tasks among two sets of replicated modules to reduce temperature hotspots in a design. However, we differ in many important aspects. First, the authors in [30] use a very simplistic thermal model, whereas we use a more detailed model to determine the temperature profile of the chip, with the given boundary conditions. Secondly, no consideration is given to the placement of the replica blocks in the floorplan. Given the dependence of the on-chip temperature profile to the spatial distribution of the blocks in the floorplan, we directly try to optimize the leakage consumption by a judicious placement of replica blocks using floorplanning. Moreover, we do not rely on the assumption that reducing the hotspot temperature would as a consequence reduce the overall leakage consumption. This is because large blocks like the cache although have low temperature may be very leaky. Finally, adding replica blocks can potentially affect the overall performance of the floorplan, which we take into account. We believe, this is the first time such a problem has been explored in the floorplanning context.

The rest of the chapter is organized as follows. Section 4.2 describes the thermal and leakage models used in our methodology, followed by Section 4.3 which uses a simple example to illustrate the benefits of TM. Section 4.4 begins by describing the floorplanning problem statement, followed by a discussion of the key ideas used in our floorplanning algorithm. This is followed by the results in Section 4.5 and finally we conclude in Section 4.6.

4.2 Preliminaries

In this section, we first briefly mention the tools used in our methodology for the sake of continuity, since these have already been described in detail in Chapter 3.

4.2.1 Thermal Model

We use the HotSpot [31] thermal modeling tool to provide a detailed thermal profile of the chip. Although described in detail in Section 3.3.2, we briefly recapitulate for the sake of continuity. HotSpot's *block* thermal model is represented in terms of the thermal system's equivalent thermal resistance and capacitace, and the network of resistors and capacitors is used to model heat conduction in the system. The formulation of this system is given by,

$$C \frac{dT}{dt} + G\,T = P \tag{4.1}$$

where G and C are the thermal conductance and capacitance matrices respectively, T is the temperature vector and P is the power vector giving the temperature and power at each node (each block is represented by a node) in the thermal RC network respectively.

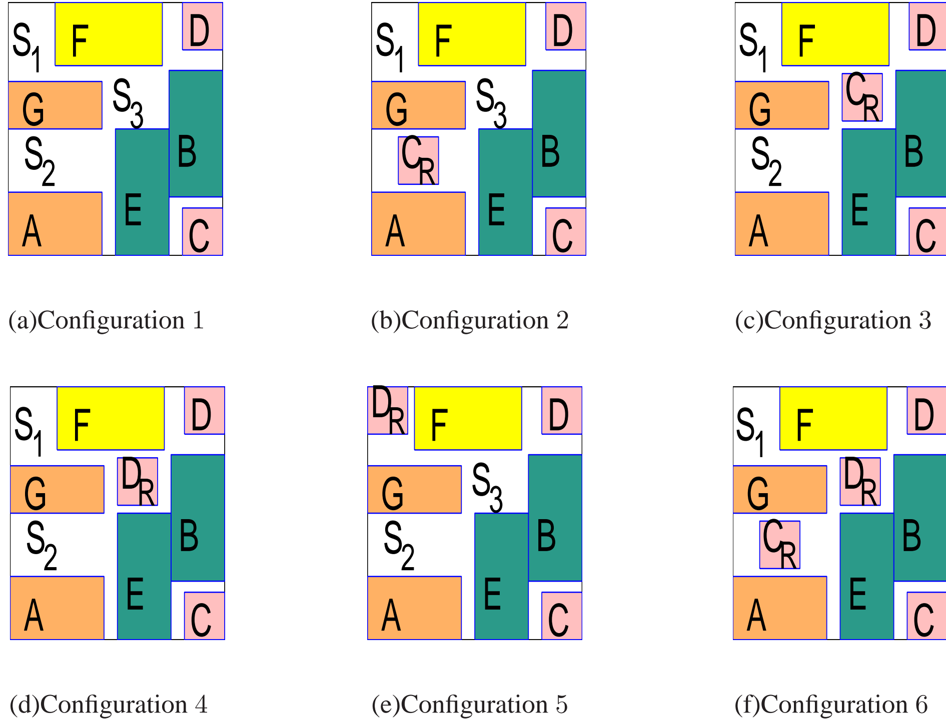
Another more accurate model is the *grid* model, where the silicon substrate is modeled as a rectangular mesh for a finer granularity of modeling temperature (see Figure 3.2). Equation 4.1 still remains the same, except that the number of nodes depends on the grid size. However, for a fine grid, this model is more compute intensive and slower to evaluate than the block model for obvious reasons.

4.2.2 Leakage Model

The sub-threshold leakage power model is similar to the one we use in Chapter 3, from [39], based on the BSIM 3 model [3], which is restated here

$$P_L(T) = A\,T^2\,e^{-\left(\frac{\alpha\,V_{dd}+\beta}{T}\right)} \tag{4.2}$$

where $P_L(T)$ is the sub-threshold leakage power dissipated at temperature T, A is a factor proportional to the area of the block (representive of the number of transistors on chip), V_{dd}



(a)Configuration 1 (b)Configuration 2 (c)Configuration 3

(d)Configuration 4 (e)Configuration 5 (f)Configuration 6

Figure 4.2: An example illustrating the motivation for TM via floorplanning. Figure 4.2(a) shows design blocks $A \ldots G$, and S_1, S_2, S_3 represent the three main deadspace regions in the floorplan. Figures 4.2(b) to 4.2(f) show different configurations of the primary and replica blocks (denoted with a '$_R$' suffix) for TM.

is the supply voltage magnitude and α and β are empirical constants whose values depend on whether the block is a functional unit such as an arithmetic unit or a memory unit such as a cache.

4.3 Motivation

We motivate the use of task migration (TM) for the reduction of active leakage via floorplanning using a simple example, shown in Figure 4.2, and also illustrate some of the important aspects that govern the TM technique. The base floorplan consists of 7 blocks labeled A through G, shown in Figure 4.2(a). These blocks have different characteristics in terms of their type, shown in Table 4.1. Also shown are the main deadspace (also known as whitespace) regions labeled S.

Table 4.1: Architectural and power characteristics of blocks shown in Figure 4.2.

Block	Architectural Features	Power Density ($Watt/m^2$)
A	Low activity combinational	$1.79e^4$
B	Low activity memory	$1.56e^4$
C	High activity combinational	$1.33e^6$
D	High activity combinational	$1.33e^6$
E	Low activity memory	$1.56e^4$
F	High activity memory	$3.13e^4$
G	Low activity combinational	$2.38e^4$

For each of these floorplans, we evaluated the total leakage by performing a simulation of Equation 4.1 with $P = P_D + P_L(T)$, where P_D is the dynamic power of a block and $P_L(T)$ is the temperature dependent leakage power of a block at temperature T given by Equation 4.2. This is done by iteratively solving the equation until convergence is reached. We use the hotspot *grid* model with the substrate divided into a 50×50 mesh. In the case of floorplans with replica blocks participating in TM, the dynamic power is equally split between the primary and replica blocks assuming an equal migration interval (we ignore the difference in dynamic power which arises due to the different interconnect capacitances driven by them). For this illustration purpose, we also ignore the switching penalty that arises due to switching between the primary set of blocks with its replica. This is reasonable because the average thermal time constants is orders of magnitude greater than the frequency of operation of the digital circuit.

The different TM configurations in Figures 4.2(b)-4.2(f) are used to illustrate various aspects of TM, the results of which are tabulated in Table 4.2.

The base floorplan is shown in Figure 4.2(a) where no TM is performed. The high activity blocks C and D are surrounded by memory blocks B, E and F and impact their temperature (and therefore sub-threshold leakage) adversely. There are two primary effects into play when considering TM.

Table 4.2: Leakage power evaluation for the TM configurations in Figure 4.2.

Config-uration	TM Block(s)	Normalized Leakage Power	Leakage Savings (%)		
			Overall	TM Blocks	non-TM Blocks
1	None	1.000	0	0	0
2	C	0.797	20.3	37.98	8.89
3	C	0.813	18.7	36.97	6.93
4	D	0.813	18.7	36.87	7.12
5	D	0.868	13.2	25.24	5.57
6	C, D	0.655	34.5	40.62	12.50

- First, the power density of the block participating in TM (for example block C in Fig. 4.2(b)) reduces thereby reducing its temperature and leakage power. We call this the *primary TM effect*.
- Second, due to the lateral conduction of heat in the substrate, a block can affect the temperature and hence leakage of its neighboring modules. We call this the *secondary TM effect*.

Configuration 2 results in more than 20% overall leakage savings in large part due to the reduction in leakage of TM block C. The reduction of leakage in block C due to the primary TM effect (show in column 5 in Table 4.2) is almost 38%. Configurations 3 and 4 shown in Figures 4.2(c) and 4.2(d) respectively, result in less leakage reduction compared to configuration 2. This is mainly because the replica module adversely affects the leakage of blocks B, E and F, resulting in a weaker secondary TM effect compared to configuration 2. From column 6 of Table 4.2, the reduction of leakage due to the secondary TM effect decreases from about 9% to 7% between configurations 2 and 3. Configuration 5 results in the least leakage savings (when one TM block is used) due to reduced primary and secondary TM effects when the replica block is placed in the chip corner. Finally, when using two TM blocks, C and D, we get a larger leakage savings due to an increase of both the primary (with 41% savings) and secondary (with 13% savings) TM effects.

4.4 Floorplanning for Task Migration

We first describe the problem statement in the context of task migration (TM). This is followed by our approach to the thermal-aware floorplanning algorithm to reduce sub-threshold leakage via TM.

4.4.1 Problem Statement

Chapter 3, describes in detail the floorplanning paradigm used in chip design automation. We also defined the floorplanning problem in Definition 3.4.1. We now define the floorplanning problem in the context of task migration below.

Definition 4.4.1 (TM-aware Floorplanning Problem). The input is a primary set of n blocks, $\Phi_P = \{B_1, \ldots, B_n\}$, each with its associated area, A_i, aspect ratio bounds, $\{AR_i^{min}, AR_i^{max}\}$, average dynamic power consumption, P_{Di}, and leakage power profile, $P_{Li}(T)$, where T denotes the temperature of the block. We compute a floorplan with Φ_P and a new set of blocks, $\Phi_R = \{B_{R1}, \ldots, B_{Rm}\}$, $m \leq n$, such that certain performance criteria are optimized or met. The m set of blocks Φ_R are the replica blocks of a subset of Φ_P, which take part in TM. The allowable area budget, ρ, represents the maximum ratio of the total area of the blocks in Φ_R to the blocks in Φ_P, and is given as an input to constrain the total area overhead due to TM. We aim to optimize the floorplan area, its overall leakage and the half-perimeter wirelength, given the above constraints. In computing the leakage of a floorplan with TM, a migrating block B_i, and its replica B_{Ri}, are each assumed to have a dynamic power dissipation of $P_{Di}/2$.

We note an interesting chicken-and-egg dilemma for the above problem statement. To compute a TM-enabled floorplan we need both the primary set of blocks, Φ_P, and the replica set of blocks, Φ_R. However, choosing the TM block set, Φ_R, depends not only on the power density and leakage profile of the primary block set Φ_P, but also on their spatial distribution in the floorplan. This is because of the finite lateral conduction of heat in the

silicon substrate wherein the temperature of a block (and therefore its leakage) is affected by that of its neighboring modules. Thus, a base floorplan and its associated temperature map are essential to compute the set Φ_R, thereby leading to a chicken and egg situation. The following sections describe in detail our method of floorplanning to choose a set of replica blocks Φ_R and compute the TM enabled floorplan.

4.4.2 Candidate Replica Block Selection using TM Leakage Sensitivity

As noted above, to compute the set Φ_R, we need a base floorplan, denoted $\mathcal{F}_\mathcal{P}$ and its associated thermal map. The steady state thermal map is computed using Equation 4.1, by setting $dT/dt = 0$ and is given by the following equation,

$$T = R \cdot P \tag{4.3}$$

where $R = G^{-1}$ is the transfer thermal resistance matrix [59] of the nodes (given by Equation 3.15 and described in detail in Section 3.3.1), and can be readily obtained from the block thermal model of HotSpot . Entry $R(i, j)$ denotes the temperature increase at node i due to a unit amount of power increase at node j.

Consider block $B_i \in \Phi_P$ as a candidate for TM. In choosing this block, we need to consider its impact vis-a-vis the floorplan leakage. As described in Section 4.3, we need to account for both primary and secondary TM effects of the migrating block B_i. The primary TM effect relates to the reduction of leakage of the block B_i, whereas the secondary TM effect relates to the reduction of leakage of the rest of the blocks, i.e., $B_j, \forall B_j \in \Phi_P, j \neq i$ in the floorplan. The leakage sensitivity of the base floorplan, $\mathcal{F}_\mathcal{P}$, to block B_i, can therefore be written as

$$\begin{aligned} \frac{\Delta P_L}{\Delta P_{Di}} &= \frac{\partial P_{Li}(T_i)}{\partial P_{Di}} + \sum_{j=1}^{n, j \neq i} \frac{\partial P_{Lj}(T_j)}{\partial P_{Di}} \\ \Rightarrow S_i &= S_i^P + S_i^R \end{aligned} \tag{4.4}$$

where S_i gives us the change in the total leakage of the floorplan due to a change in the dynamic power of block B_i. In the above equation, the first term, S_i^P, gives us the change in leakage of B_i, with respect to a change in its dynamic power. This term therefore relates to the primary TM effect. The summation term, S_i^R, computes the change in leakage of the other modules in the design with respect to a change in the dynamic power of block B_i. This term relates to the secondary TM effect of B_i. Given the steady state thermal profile, and the leakage profile of all the blocks in the floorplan, we can write the individual terms in Equation 4.4 as

$$\begin{aligned} S_i^P &= \frac{\partial P_{Li}(T_i)}{\partial T_i} \cdot \frac{\partial T_i}{\partial P_{Di}} \\ &= J_i(T_i) \cdot R(B_i, B_i) \end{aligned} \tag{4.5}$$

where $J_i(T_i)$ is the jacobian of the leakage power with respect to temperature computed at T_i and $R(B_i, B_i)$ denotes the change in temperature of B_i due to a change in its dynamic power. Similarly, S_i^R can be written as,

$$\begin{aligned} S_i^R &= \sum_{j=1}^{n, j \neq i} \frac{\partial P_{Lj}(T_j)}{\partial T_j} \cdot \frac{\partial T_j}{\partial P_{Di}} \\ &= \sum_{j=1}^{n, j \neq i} J_j(T_j) \cdot R(B_j, B_i) \end{aligned} \tag{4.6}$$

where $J_j(T_j)$ denotes the jacobian of the leakage power of block B_j computed at temperature T_j and $R(B_j, B_i)$ denotes the change in the temperature of block B_j due to a change in the dynamic power of block B_i (obtained from the thermal resistance matrix).

Although sensitivity gives us the relative importance of a block to its effect on TM, the absolute effect needs to take its dynamic power into consideration, i.e., a block with larger

Algorithm 10 Φ_R = TMCandidates $(\Phi_P, \mathcal{F}_\mathcal{P}, \rho)$ // Φ_P = set of primary blocks in base floorplan $\mathcal{F}_\mathcal{P}$; ρ = allowable area budget; Φ_R = set of candidate replica blocks partaking in TM

1: $\Phi_R = NULL$
2: Compute the steady state temperature profile of $\mathcal{F}_\mathcal{P}$
3: **for all** $B_i \in \Phi_P$ **do**
4: Compute C_i // see Equation 4.7
5: **end for**
6: Sort Φ_P in non-increasing order of C_i
7: $A_{\mathcal{F}_\mathcal{P}}$ = total area of all blocks in Φ_P
8: $A_o = \rho \cdot A_{\mathcal{F}_\mathcal{P}}$ // allowable total replica block area
9: $i = 0$
10: **while** $A_o > 0$ and $i < |\Phi_P|$ **do** // iterate through all blocks
11: **if** Area$(B_i) < A_o$ **then**
12: Insert B_i into Φ_R
13: $A_o = A_o-$ Area(B_i)
14: **end if**
15: **end while**
16: **return** Φ_R

dynamic power consumption will have a larger TM effect than a block with a lower dynamic power consumption. Therefore, to compute the overall impact of block B_i in TM mode, we need to weight its sensitivity with its total dynamic power. A measure, C_i, of the reduction in leakage power due to a participating block B_i is obtained as

$$C_i = S_i \cdot P_{Di} \tag{4.7}$$

Algorithm 10 computes the set of TM candidate replica blocks, Φ_R, given base floorplan, $\mathcal{F}_\mathcal{P}$, consisting of primary blocks, Φ_P.

We begin by first computing the steady state thermal profile of the floorplan $\mathcal{F}_\mathcal{P}$ followed by the computation of the leakage reduction measure due to TM of each block, given by C_i in Equation 4.7. The blocks in $B_i \in \Phi_P$ are then sorted in decreasing order of their measure values so that the highest priority is given to the block with the largest C_i value. Steps 7-8 compute the allowable total area of the replica modules, denoted A_o, from the

area budget ρ. Finally, we greedily select the candidate TM blocks, Φ_R, in Steps 10-15, until we exhaust the available area overhead. This set of blocks is then inserted into $\mathcal{F_P}$ to compute a new floorplan $\mathcal{F_R}$, a procedure we call *floorplan patching*, described next.

It must be noted that although the problem of selecting replica modules can be solved using dynamic programming (similar to the knapsack problem), we settled on a simple heuristic. After all, the gain C_i is only an approximation of the power savings due to the fact that the modules in the floorplan change location as a result of resizing after the replicas have been inserted.

4.4.3 TM-Aware Floorplan Patching

As described above, given the primary set of blocks, Φ_P, and its accompanying base floorplan $\mathcal{F_P}$, we compute the replica set of blocks Φ_R to be inserted, and compute a new TM-aware floorplan. We judiciously place the replica blocks making use of the available deadspaces in $\mathcal{F_P}$. The primary objective in using the existing deadspaces in $\mathcal{F_P}$ is to avoid a large increase in the floorplan area and to avoid large perturbations in the floorplan. This is in keeping with the chicken-and-egg dilemma described in Section 4.4.1, since our selection of Φ_R was based on $\mathcal{F_P}$. Moreover, as was seen in the example in Section 4.3, it is important to place the replica blocks in the appropriate deadspace to exact the maximum leakage reduction. We therefore divide the task of deadspace allocation into three parts, the first dealing with deadspace selection, the second dealing with matching the blocks in Φ_R with the selected deadspaces, and finally inserting the replica blocks in their alloted deadspaces.

4.4.3.1 Deadspace Selection

Once we select the candidate TM replica blocks, Φ_R, the next task is to place those blocks within the existing deadspaces of the floorplan. To do this, we first need to select a set

Algorithm 11 DS_R = DSSelection ($\mathcal{F_P}, \Phi_R, \epsilon$) // $\mathcal{F_P}$ = base floorplan; Φ_R = set of candidate replica blocks partaking in TM; ϵ = deadspace threshold; DS_R = subset of deadspaces in floorplan $\mathcal{F_P}$ that are candidates for accommodating replicas Φ_R

1: $DS_R = NULL$
2: DS_P = deadspaces of non-slicing floorplan $\mathcal{F_P}$ using plane sweep // See Algorithm 9
3: B_{min} = block in Φ_R with minimum area
4: $A_{thresh} = \epsilon \cdot$ Area(B_{min}) // compute deadspace area threshold
5: **for all** $DS_i \in DS_P$ **do**
6:     **if** Area(DS_i) $> A_{thresh}$ **then**
7:         Insert DS_i into DS_R
8:     **end if**
9: **end for**
10: **return** DS_R

of candidate deadspaces, which we call the deadspace selection problem. Algorithm 11 describes the procedure of selecting candidate deadspaces in the non-slicing base floorplan $\mathcal{F_P}$. In Step 2 we first compute the deadspaces in $\mathcal{F_P}$ by a plane sweep algorithm using a balanced interval tree [21]. This has already been described in detail in Section 3.7.4 and given by Algorithm 9. Next, in Steps 5-9 with the candidate TM blocks in Φ_R, in order to avoid increasing the area and perturbing $\mathcal{F_P}$ by a large amount, we choose those candidate deadspaces which have an area greater than a certain fraction, ϵ, of the block with minimum area. We refer to ϵ as the deadspace threshold. In our experiments ϵ was chosen as 0.75.

In the above algorithm, DS_P denotes the set of all deadspaces in the base floorplan Φ_P and DS_R denotes the set of candidate TM deadspaces into which we intend to insert the candidate replica TM blocks, Φ_R. The next two subsections describe how we use these deadspaces to derive a TM-aware floorplan.

4.4.3.2 Deadspace Allocation Via Bipartite Matching

As seen from the example in Section 4.3, for a single TM block, configuration 2 results in much better leakage savings than configuration 5, implying that it is important to correctly allocate the TM replica blocks to the candidate deadspace blocks. To prudently assign modules in Φ_R to deadspaces in DS_R, we need a measure of the fitness of allocating TM

block $B_{Ri} \in \Phi_R$ to deadspace $DS_{Rj} \in DS_R$. We compute a fitness measure similar to Equation 4.7 described in Section 4.4.2, which was used to decide the candidate TM block set Φ_R. We denote the fitness of placing block B_{Ri} at deadspace DS_{Rj} as C_i^j, given by Equation 4.8 below,

$$C_i^j = S_i^j \cdot P_{Di}^j \tag{4.8}$$

where S_i^j is the leakage sensitivity and P_{Di}^j the dynamic power of the replica block B_{Ri} placed in deadspace D_{Rj}. As in Equation 4.7, this is given by

$$S_i^j = \frac{\Delta P_L(T)}{\Delta P_{Di}^j} \tag{4.9}$$

where $P_L(T)$ is the leakage of the floorplan. Similar to Equations 4.5 and 4.6 we can compute S_i^j as

$$\begin{aligned} S_i^j &= S_i^{jP} + S_i^{jR} \\ &= J_i(T_j) \cdot R(DS_{Rj}, DS_{Rj}) \\ &\quad + \textstyle\sum_{k=1}^{n} J_k(T_k) \cdot R(B_k, DS_{Rj}) \end{aligned} \tag{4.10}$$

where $J_j(T_j)$ is the jacobian of the leakage of replica block B_{Ri} computed at T_j the temperature of deadspace D_{Rj}, and $R(B_k, DS_{Rj})$ is the increase in temperature of primary block B_k due to a unit increase in the dynamic power at DS_{Rj}. As before, S_i^{jP} captures the primary TM effect of placing replica block B_{Ri} at deadspace DS_{Rj} and S_i^{jR} captures its secondary TM effect.

We next compute a bipartite graph $G(U, V, E)$, illustrated in Figure 4.3, where U and V denote the two partitions and E denotes the set of edges, $e(u_i, v_j)$, between node $u_i \in U$

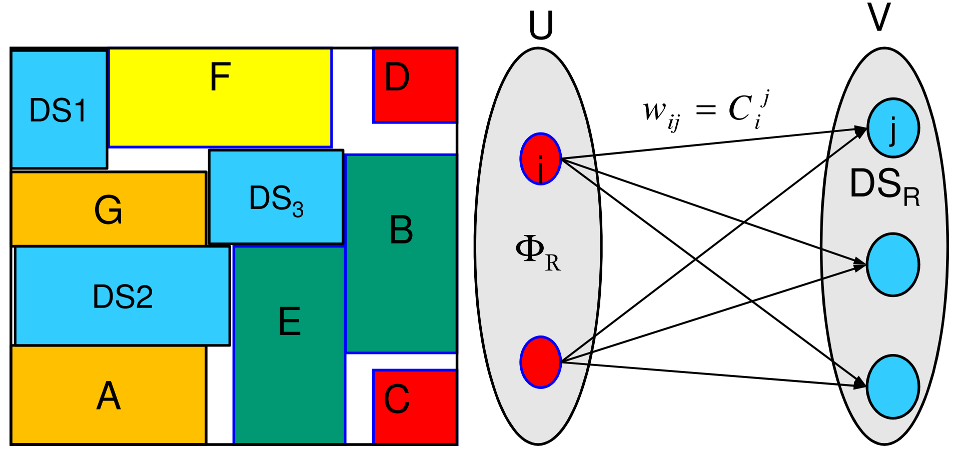


Figure 4.3: Illustration of deadspace allocation using bipartite matching.

and $v_j \in V$. For every candidate replica block in $B_{Ri} \in \Phi_R$, we create a node u_i in U and for every candidate deadspace in $DS_{Rj} \in DS_R$ we create a node v_j in V. Edges are created between every $u_i \in U$ to $v_j \in V$ pair and the weight of edge $e(u_i, v_j)$, w_{ij}, is given by the fitness measure C_i^j of allocating a replica block B_{Ri} to deadspace D_{Rj} computed from Equation 4.8. The problem of bipartite assignment is that of finding a *perfect matching* such that every node of the graph is incident on a *matched* edge. Because the problem of heat conduction in a system follows the superposition principle, the total cost of the assignment is a good indication of the leakage increase of the floorplan with the replica blocks brought into active mode. We use a *minimum weighted bipartite assignment* algorithm [42] to pair each candidate replica block with a single dead space such that the total leakage measure C_i^j of the assignment is minimized. The complexity of this algorithm is $O(n \cdot (m + n \log n))$ where n is the number of nodes and m is the number of edges in the bipartite graph. The overall procedure for deadspace matching is shown in Algorithm 12

Note the use of a min-cost objective in deciding the assignment of TM replica blocks to deadspaces as opposed to a max-based objective in their selection in Algorithm 10. This

Algorithm 12 M = DSMatching ($\mathcal{F_P}, \Phi_R, DS_R$) // $\mathcal{F_P}$ = base floorplan; Φ_R = set of candidate replica blocks partaking in TM; DS_R = selected deadspaces in floorplan $\mathcal{F_P}$; M = matching between blocks in Φ_R and DS_R

1: **for all** $B_{Ri} \in \Phi_R, DS_j \in DS_R$ **do**
2: Compute C_i^j // see Equation 4.8
3: **end for**
4: Create bipartite graph $\mathcal{G_B}(U, V, E)$ where node $u_i \in U$ represents $B_{Ri} \in \Phi_R$ and node $v_i \in V$ represents $DS_{Rj} \in DS_R$. Edge $E(u_i, v_j)$ has weight $w_{ij} = C_i^j$
5: Compute M = Minimum-weighted Bipartite Matching of $\mathcal{G_B}$
6: **return** M

is because in the former, a replica block needs to be optimized when turning on from idle to active mode whereas in the latter, a primary block needs to be optimized when turning off from active to idle mode. There is a bias in the optimization of the primary block in switching from its active to idle mode and this is owing the lack of prior knowledge of the blocks participating in TM.

4.4.3.3 Deadspace Substitution

The above two steps describe the process of obtaining the candidate set of deadspaces DS_R and matching them up with the candidate TM block set Φ_R. It must be kept in mind however, that the deadspace will likely not have the same area as the block, neither will it have the same dimensions. Therefore, we need to appropriately insert a candidate TM block B_{Ri} into its matched deadpsace DS_{Rj}. This is the last step in the floorplan patching process. Algorithm 13 shows our method of inserting the candidate TM blocks, Φ_R into their respective matched deadspaces (given by matching M), to compute a patched floorplan $\mathcal{F_R}$. Step 1 first computes the horizontal and vertical constraint graphs of the base floorplan $\mathcal{F_P}$ so that the relative ordering of the blocks is obtained. Next, in Steps 2-5 we replace the width and height of the matched deadspaces with those of the candidate TM blocks in Φ_R, thereby patching $\mathcal{F_P}$. When inserting the block, we orient it such that its dimensions conform to that of the deadspace, i.e., for a block that does not fit, it is inserted such that its area outside the deadspace is minimized. If it fits, then it is inserted

Algorithm 13 $\mathcal{F_R}$ = DSSubstitution ($\mathcal{F_P}, \Phi_R, M$) // $\mathcal{F_P}$ = base floorplan; Φ_R = set of candidate replica blocks partaking in TM; M = matching between blocks in Φ_R and floorplan deadspaces; $\mathcal{F_R}$ = patched floorplan with TM blocks

1: Compute horizontal (HCG) and vertical constraint graphs (VCG), of base floorplan $\mathcal{F_P}$ // finds relative ordering
2: **for all** $B_{Ri} \in \Phi_R$ **do**
3: Find deadspace D_{Rj} which matches B_{Ri} using M // see Algorithm 12
4: Orient B_{Ri} to conform with the dimensions of deadspace D_{Rj}
5: **end for**
6: **if** there exists a non-conforming block B_{Ri} inserted in D_{Rj} **then**
7: Compute patched floorplan $\mathcal{F_R}$ from HCG and VCG
8: **end if**
9: **return** $\mathcal{F_R}$

such that the centers of the deadspace and the block align. This step however could result in an invalid floorplan with overlaps. In such a case (Steps 6-8), we recompute the block coordinates from the horizontal and vertical constraint graphs of the floorplan computed in Step 1, such that there are no overlaps, and return the newly patched floorplan $\mathcal{F_R}$.

4.4.4 Patching the Example Floorplan

We execute the TM-aware floorplan patching procedure described in Section 4.4.3 on a base floorplan, $\mathcal{F_P}$ shown in Figure 4.4(a). With an area budget $\rho = 10\%$, execution of Algorithm 10 gives a single TM candidate block $\Phi_R = \{C\}$, shown as the red shaded block in Fig. 4.4(b). Despite their equal power dissipation, C has a higher leakage measure (Equation 4.7), $C_C = 4.796$ compared to block D with leakage measure $C_D = 4.693$. This is due to a greater influence of C on its neighboring modules, implying a greater secondary TM effect compared to block D. Algorithm 11 selects dead spaces $DS_R = \{DS_1, DS_2, DS_3\}$ as candidates for replacement, shaded green in Figure 4.4(c). The individual costs of assigning TM block C_R to the deadspaces in DS_R computed using Equation 4.8 are $C_{C_R}^{DS_1} = 2.585$, $C_{C_R}^{DS_2} = 2.317$ and $C_{C_R}^{DS_3} = 2.587$, which results in a matching of deadspace DS_2 using Algorithm 12. Finally, insertion of the TM block into $\mathcal{F_P}$ using Algorithm 13 gives floorplan $\mathcal{F_R}$ shown in Fig. 4.4(e). Fig. 4.4(f) shows the patched floorplan obtained when the

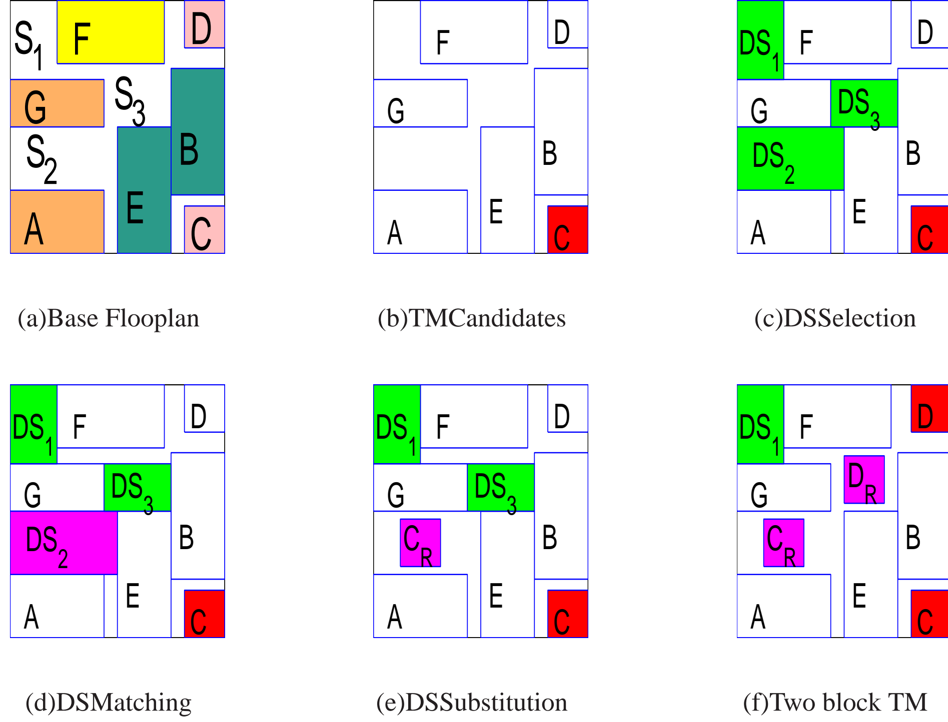


Figure 4.4: Execution of TM-aware floorplan patching on the base floorplan in Fig. 4.4(a). Fig. 4.4(b) shows the selection of a candidate TM block C (in red) for TM, followed by the selection of candidate deadspace locations DS_1, DS_2, DS_3 (in green) in Fig. 4.4(c), matching of C with DS_2 (in pink) in Fig. 4.4(d) and finally insertion of TM replica block C_R to obtain the patched floorplan in Fig. 4.4(e). Fig. 4.4(f) shows the case of floorplan patching with two candidate TM blocks selected.

area budget $\rho = 15\%$ resulting in the selection and matching of two candidate TM blocks $\Phi_R = \{C, D\}$.

4.4.5 Bringing it Together: Floorplanning

We can now describe our floorplanning algorithm as shown in Figure 4.5. To recapitulate what we have mentioned previously, we use the Parquet floorplanning tool [8], which is a fixed die non-slicing floorplanner based on the simulated annealing combinatorial optimization algorithm and can handle multiple constraints like area, aspect ratio and wire-length. At every step, a candidate base floorplan $\mathcal{F}_\mathcal{P}$ is evaluated for TM by first computing the TM candidates, Φ_R, using Algorithm 10. These are then used to select the candidate

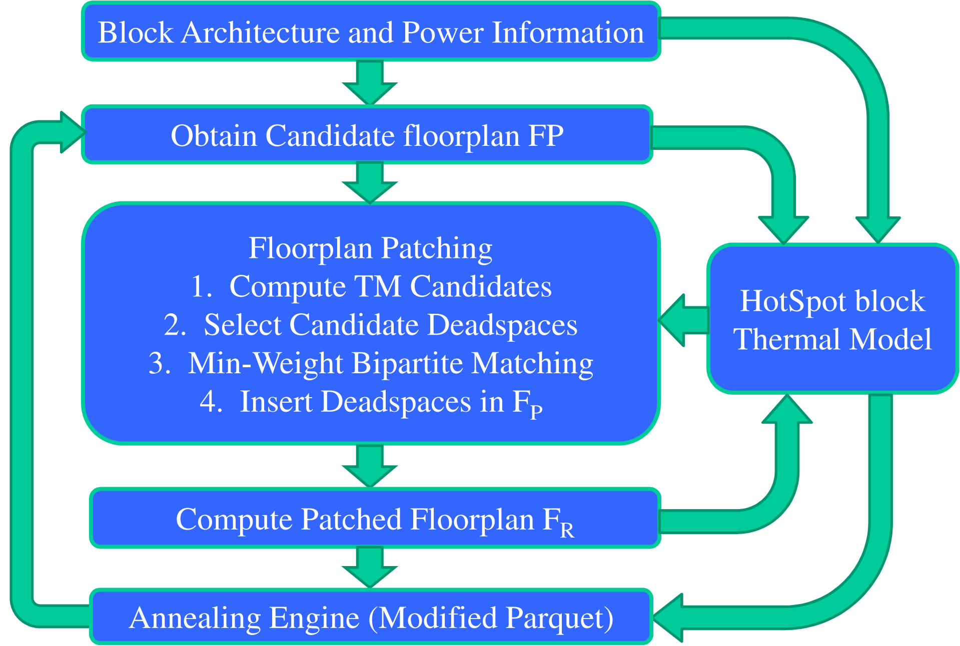


Figure 4.5: Overall flow of our TM-aware floorplanning approach.

deadspaces $DS_R \subset \mathcal{F_P}$ in which a TM block can be inserted (Algorithm 11). We perform a min-cost bipartite assignment of TM blocks to candidate deadspaces with Algorithm 12 and use Algorithm 13 to obtain a TM-aware floorplan $\mathcal{F_R}$. The TM-aware floorplan is used to compute the cost term during annealing as follows,

$$\begin{aligned} C &= \alpha \cdot A + \beta \cdot HPWL_w + \gamma \cdot L_T \\ &\quad +(1-\alpha-\beta-\gamma) \cdot AR \end{aligned} \tag{4.11}$$

where A is the area, $HPWL_w$ is the weighted half-perimeter wirelength, L_T is the temperature dependent leakage cost and AR is the aspect ratio of the TM-aware floorplan $\mathcal{F_R}$. The factors α, β and γ are used to weight the individual cost terms. Parquet in fixed-outline floorplanning mode allows the user to specify a given bound on the total deadspace. We replicate nets which contain TM blocks, adding their contribution to $HPWL_w$ when performing TM-aware floorplanning.

In order to avoid time consuming iterations to compute the temperature dependent leakage

power of floorplan $\mathcal{F}_{\mathcal{R}}$, we use our work from Chapter 3, a formulation which captures the temperature dependent rise in leakage power with respect to time, in what we call the transient formulation of leakage. The basic idea is to capture the transient effect by computing the temperature dependent increase in the leakage power over time Δt given the average dynamic power dissipation P_D. This is described in detail in Section 3.7.1. This formulation is independent of the type of leakage model used. For purposes of continuity, we restate the formulation below.

$$\begin{aligned} L_T &= \frac{\Delta P_L(T)}{\Delta t} \\ &= J(T_s) \cdot C^{-1} \cdot P_L(T_s) \Big|_{T_s = G^{-1} \cdot P_D} \end{aligned} \tag{4.12}$$

where G and C are the thermal conductance and capacitance matrices respectively, T_s is the steady state temperature distribution vector of the blocks when dissipating their dynamic power P_D, $P_L(T_s)$ (see Equation 4.2) is the vector of block leakage powers computed at temperature T_s and $J(T_s)$ is the Jacobian of block leakage powers with respect to temperature evaluated at T_s.

4.5 Experiments Setup and Results

This section describes the benchmarks evaluated in our work, along with the experimental setup and simulation results. All experiments were performed on a Linux Intel Pentium® 3.2 GHz processor with 2GB RAM, in the $100nm$ technology node.

4.5.1 SOC Benchmarks

To evaluate our task migration (TM) aware floorplanning approach we chose the SOC test benchmarks [40]. The benchmarks designs are expressed in a tree-based hierarchy format. For our floorplanning, we chose those modules that are in the bottom most level of the

tee. Every internal node of the design tree was considered a net with its terminals as the modules in its bottom most leaf nodes. The weight of the net was assigned a value proportional to the height of the internal node in the tree. Architectural information was extracted by using the given number of module terminals and applying a Rent's rule based formulation with a suitable Rent's coefficient to obtain the number of logic gates in a module. This, in combination with the design technology node ($100nm$ for our work), gives us the block area information. It is important to distinguish between combinational and memory type modules mainly due to their different power densities and leakage sensitivities to temperature. Due to the lack of dynamic power information, the power density was chosen randomly between the range $\{5.0e^4, 1.0e^6\}$ for combinational blocks and $\{1.0e^4, 4.0e^4\}$ for memory blocks. Table 4.3 shows various parameters of interest for the benchmarks used in our experiments.

Table 4.3: The SOC benchmarks. N_C, N_M and N_T denote the count of combinational, memory and total blocks in the design. P_{den} denotes the average power density of a module in the design.

Benchmark Name	Number of Modules			Avg. P_{den} $(Watt/m^2)$
	N_T	N_C	N_M	
$d695$	10	6	4	$3.14e^5$
$h953$	8	3	5	$2.8e^5$
$g1023$	14	6	8	$2.21e^5$
$p34932$	19	4	15	$2.23e^5$

4.5.2 Results

Although our floorplanning framework uses the HotSpot *block* model for thermal modeling, when evaluating an optimized floorplan for leakage savings, we use the *grid* model with a mesh of 50×50, to get better accuracy of evaluation. For each benchmark, we run the floorplanner 5 times, first without any task migration (TM) of modules, and second in the TM-aware mode with an area budget $\rho = 10\%$, and choose the best floorplan. Recall that

ρ is the factor used in determining the number of candidate modules chosen for TM.

Table 4.4 shows the savings in leakage obtained by our TM-aware floorplanner. On average, we obtain leakage savings of about 29% with a 5.7% increase in area. An important benefit of TM is also lowering the maximum chip temperature by greater than $20°$. Moreover, the area penalty is in consonance with ρ indicating that our floorplanner is effectively able to assign deadspaces to the TM blocks. Also note that there is no correspondence between the number of TM replica modules and the leakage power savings, and relates to the interdependency between the allowable deadspace and the chosen TM replica blocks. Due to a small number of nets in each benchmark evaluated, the $HPWL$ term is primarily a reflection of the chip dimensions as a result of which the degradation is proportional to the square-root of the increase in area (which cannot be avoided in TM-aware floorplanning).

Table 4.4: Performance results for TM-aware floorplanning with respect to the floorplan area, leakage power, maximum chip temperature and half-perimeter wirelength. N_{TM} denotes the number of replica modules for the TM-aware floorplan.

Bench-mark	Area Increase ($\%$)	N_{TM}	Leakage Savings ($\%$)	Max. Temperature		$HPWL$ ratio
				No TM	TM-aware	
$d695$	9.40	3	18.11	142.4	116.6	1.25
$h953$	7.06	1	24.13	141.8	115.9	1.11
$g1023$	0.68	3	39.47	139.2	98.9	1.09
$p34932$	5.31	3	34.98	152.2	91.9	1.47

Figure 4.6 shows the contour plots of the chip temperature distribution of the benchmark $h953$ for floorplans obtained without and with TM-aware floorplanning. In spite of block $SOC7$ being the module with the greatest leakage power and the hotspot in the design, it cannot be selected due to the limited area budget. Also, right neighboring high power density block $SOC8$ restricts the spread of heat of $SOC7$ in the horizontal direction. By migrating activity from $SOC8$, we take advantage of the secondary effect of TM to allow block $SOC7$ to better dissipate its heat laterally. This however results in a slightly higher leakage for $SOC8$ due to a reduced primary TM effect of the block. Our floorplanning

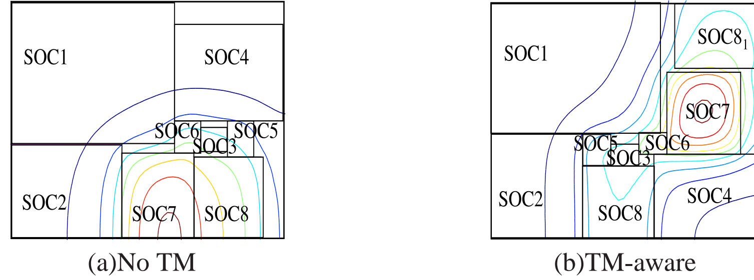


Figure 4.6: Chip temperature contours for floorplans without and with TM. The replica module is labeled $SOC8_1$.

algorithm effectively trades off the primary and secondary TM effects of a block in both its selection and placement.

4.6 Conclusion

In this chapter we proposed to reduce the active sub-threshold leakage of a design by a new formulation of task migration enabled floorplanning. Ours is the first work to formulate the task migration problem in the floorplanning context. We recognize the chichen-and-egg nature of the problem and solve it using a novel scheme to exploit what we call the primary and secondary task migration effects. Using a simple greedy heuristic on a base floorplan, we first select blocks for task migration followed by a bipartite matching to judiciously assign deadspaces to place replicated blocks in the base floorplan. Experiments show a 29% reduction in leakage power with a 6% area overhead.

One important aspect of the task migration paradigm that has not yet been addressed and is an open problem is the granularity of blocks that participate in TM. For instance, choosing a large block like an entire ALU unit might not be feasible due to the overhead in the interconnects required for wiring the block and its replica with the rest of the circuit. Moreover, the performance penalty when transferring data from one block set to its replica may be significant. On the other hand, choosing a very small granularity for replication may not give us the intended benefits in terms of reducing the active power density, because it would

most likely need to be placed close to its primary block set.

It would also be interesting to compare this scheme with other thermal management techniques like dynamic voltage scaling (DVS). Such schemes can be compared when for instance, the objective is to constrain the maximum chip operating temperature. In the DVS scheme, the operating voltage of the chip is reduce to reduce the power density, albeit at a loss in performance. In the TM scheme however, we believe the same operating conditions can be applied, albeit at a smaller migration interval. This is an open area and needs more research.

Chapter 5

CAD and Architecture Exploration of Three-Dimensional FPGAs

5.1 Introduction

Shrinking feature sizes allows an increase in the number of transistors packed on a single die. These facilitate the implementation of more complex and large designs. Such an increase in device count however brings about problems of signal integrity, increase in wire delay accompanied by an increase in the power budgets of circuits, process variations, larger IR-drops and increased temperature variability. 3D IC integration is a new design paradigm which attempts to alleviate some of these issues by stacking multiple thin active device layers and interconnect levels on top of each other, resulting in short inter-layer vias with small aspect ratios [51]. There have been a number of initiatives in both academia [14] and industry [1] towards making 3D IC integration more main stream. This is because such a design paradigm can result in shorter interconnect lengths and hence faster circuits which allows us to boost yield.

There have been a number of CAD tools developed for the standard cell technology. In [22]

the authors present a placement, routing and 3D layout editor tool to assist in 3D designs. The authors use min-cut partitioning for interlayer via minimization. Using up to 5 layers showed an improvement of 51% in wire-length. The Gravity placement tool [48] reports reductions of up to 74% reduction in wire-length. Along with the benefit of reduced interconnects, 3D ICs need to deal with increased on-chip temperatures due to vertical stacking. The work in [27] addresses the issue of via placement in 3D chips to reduce hotspots in the 3D design.

On the 3D FPGA front, Alexander et al. proposed to build a 3D FPGA architecture by stacking a number of 2D FPGA bare dies [9]. The Rothko architecture in [36] places routing-and-logic blocks (RLBs) on multiple layers to facilitate the 3D design. An improvement of this architecture by separating the routing switches and logic in different layers was proposed in [17]. In all these works inter-layer connections are provided between adjacent layers only. However, current day technologies allow us to fabricate 3D vias that span multiple layers.

The goal of this work is to provide efficient placement and routing tools for 3D FPGAs wherein we investigate the effect of 3D integration on delay in addition to wire-length. We use multi-segment architectures in which a 3D via can span multiple layers. We have developed 2 placement algorithms. The first is based on simulated annealing and the second is a hybrid approach, with partitioning used to divide the circuit into its layers and annealing used for high quality placement refinement. Our routing algorithm is a direct adaptation of the VPR [11] algorithm. The main contributions of this work are as follows:

1. **SA-TPR:** In addition to the partitioning-based 3D placement tool in TPR, we have developed a Simulated Annealing based version of TPR (called SA-TPR) to provide speed / quality trade-offs.
2. **Hybrid-TPR:** We also developed a partitioning based placement tool with simulated

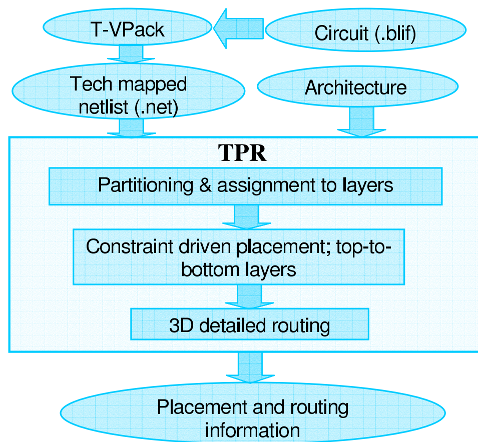


Figure 5.1: Flow diagram of TPR: 3D Placement and routing tool is similar to VPR's flow diagram.

annealing based refinement. This was done to obtain the benefits of the execution speed of TPR and the quality of SA-TPR.

3. **Architectural analysis:** We analyze potential benefits that 3D integration can provide for FPGAs. More specifically, we place and detail route circuits onto 3D FPGA architectures and study the variation in circuit delay and total wire-length compared to their 2D counterparts, under different 3D architectural assumptions. The results of this study and similar studies in future can guide researchers in designing high performance 3D FPGA fabric architectures.

5.2 Overview of TPR

This section gives a brief overview of the TPR tool flow and a description of the placement algorithm described in [7]. The tool flow is shown in Figure 5.1.

The circuit is converted from the .blif format and technology mapped using T-VPack [41], to obtain a .net netlist file. This along with the FPGA architecture description file are fed to the TPR placement engine. The placement begins by performing a partitioning of the netlist using the well known partitioning tool h-metis [33]. The number of partitions is made equal

to the number of layers in the 3D FPGA. The primary objective of the partitioning phase is to reduce the number of cuts between layers. This is done to minimize the number of vertical vias to be used, which due to their lower density are scarce compared to horizontal channels.

Once the partitioning is performed, the next phase is layer assignment using a linear placement technique. Briefly, the clustered netlist is organized as a matrix incorporating the number of cuts and the wire-length of the vias between layers. The technique described in [6] is then used to minimize the 'bandwidth' of the matrix. Once the circuit is partitioned and assigned to layers, placement within each layer is performed in a top-down fashion. Initially, the top-most layer is placed by unconstrained recursive partitioning. Critical nets in this layer are then projected to obtain constraints on the placement of blocks in lower layers. By doing this, the placement algorithm attempts to minimize the 3D bounding box.

The routing algorithm of TPR is similar to that used in VPR and is based on the Pathfinder negotiated congestion algorithm [23]. Extra penalties are added to bends created by a route using vertical vias in order to discourage the routing engine from choosing a route spanning multiple layers for a net which is placed in a single layer.

5.3 SA-TPR: Simulated Annealing Based 3D Placement

To provide the designer flexibility we also provide a simulated annealing based engine for the 3D placement. We call this SA-TPR. As will be seen in the results section, SA-TPR offers better quality placements at the expense of increased run-time. Our annealing placement engine can also be used to evaluate the impact of any new placement approach in the future. The flow of our algorithm is shown in Fig. 5.2.

The simulated annealing placement engine is fully 3D integrated and is based on the model that VPR uses for its 2D placement. The user specifies the number of layers as well as the annealing schedule to use. As in VPR, our SA engine can place circuits with constraints

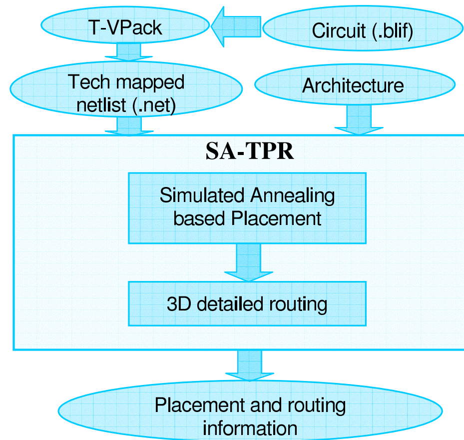


Figure 5.2: Flow diagram of SA-TPR: The partitioning based placement is replaced with a simulated annealing engine.

of both wire-length and timing. Wire-length of a net is calculated as the weighted sum of its projected 2D bounding box and its vertical span. The weight on the vertical span is set to a high value to discourage heavy usage of scarce vertical vias. The cost of a net e is described by the equation below

$$C_{3D}(e) = q \cdot C_{2D}(e) + \alpha \cdot S_Z(e) + \beta \cdot N_L(e) \tag{5.1}$$

In Equation 5.1, C_{3D} is the cost of placing net e, used in the annealing algorithm; q is a correction factor to 2D bounding box computation, which accounts for nets that have more than 3 terminals (the original VPR code uses this factor), C_{2D} is the half perimeter bounding box of the projection of all the terminals of e, S_Z is the vertical span of e and N_L gives the number of layers on which terminals of net e are placed. To see the importance of factors S_Z and N_L consider the two placements in Fig. 5.3.

The two placement scenarios would be treated identically if we did not separately consider both the vertical span of a net, and the number of layers in which its terminals are placed. Each of these cost components are scaled by appropriate scaling factors, α, which discourages placing the terminals of a net far apart in the z dimension (otherwise routing of the net

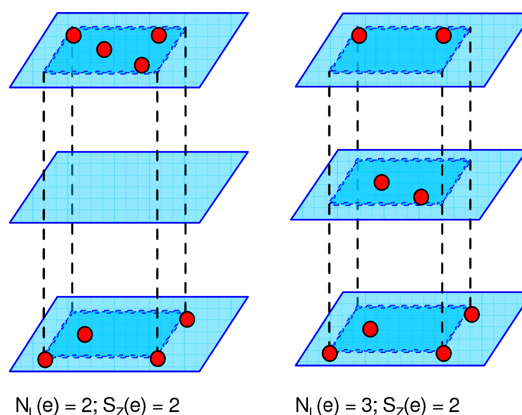


Figure 5.3: Two possible placement scenarios of the same net e with different values of S_Z and N_Z.

would require increased length vertical vias), and β, which restricts the number of vertical vias (vertical channel density is lower than horizontal channel density and β reflects that ratio). The placement on the left in Fig. 5.3 is preferred to the one on the right, as it could potentially use only one vertical segment of length two to connect the terminals in different layers. The placement on the right of the figure is likely to use more vertical routing resources.

Computation of the timing cost of a net essentially follows the approach of VPR (timing criticalities for nets are computed based on slacks). The modification for the 3D case is as follows. First, the source-sink connection whose delay we compute is projected onto 2D and its separation Δx and Δy is calculated. Delay lookup tables similar to VPR are used to calculate these values wherein unlimited routing resources are assumed. To accommodate a 3D structure, the separation of the connection in the third dimension is found and its delay is looked up using only one dimension of the delay tables, i.e. a net that spans a distance of Δz in the vertical dimension has the same delay as a 2D net with $(\Delta z, 0)$ bounding box. Finally, the annealing engine constrains movement in the x and y directions similar to VPR (initially movement is allowed across the entire dimension of the chip and then gradually it is shrunk to neighboring CLBs). Movement in the vertical dimension (the z dimension)

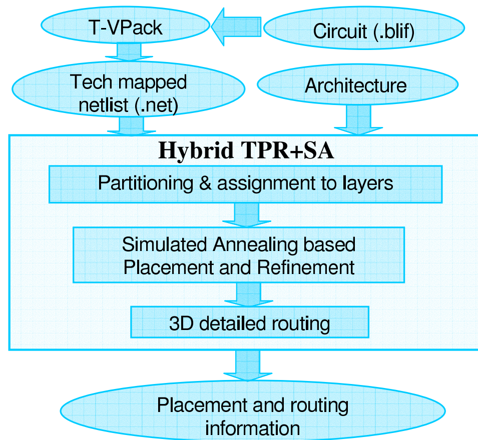


Figure 5.4: Flow diagram of Hybrid-TPR: The partitioning based initial placement is followed by a simulated annealing based layer refinement.

is unrestricted in order to fully explore the vertical connection space.

5.4 Hybrid-TPR: Combined Partitioning and Simulated Annealing Based 3D Placement

We also implemented a mixed partitioning and simulated annealing placement algorithm. This is called Hybrid-TPR. The flow of Hybrid-TPR is shown in Fig. 5.4. The first step consists of using TPR's partitioning and linear placement to obtain an initial solution. The next step uses simulated annealing on individual layers (under the restriction of not moving cells between layers) for a detailed layer placement. As will be shown in the results section, our algorithm takes advantage of the fast partitioning based initial placement of TPR to obtain high quality final placements without hurting performance. The algorithm also provides designers with an effective tool to trade-off runtime with quality.

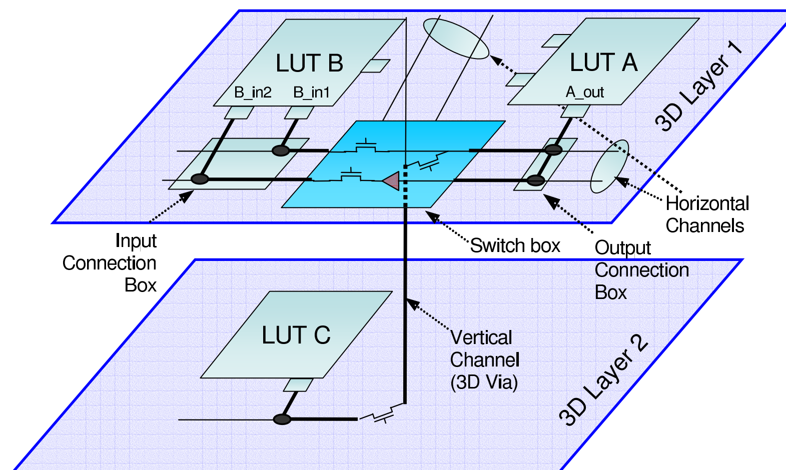


Figure 5.5: Illustration of the 3D FPGA architecture.

5.5 Simulation Results

5.5.1 3D Architectures

The main objective of our 3D FPGA CAD tools is to facilitate the study of different 3D architectures and their impact with respect to performance metrics like delay and wire-length. We considered a *Sing-Seg* and a *Multi-Seg* architecture. Both have the same horizontal layers and are similar to the Virtex II architecture (with segment lengths of 1, 2, 6 and long lines). The difference lies in the fact that whereas *Sing-Seg* has vertical (inter-layer) vias of length one only (i.e. direct vertical connections can be made between adjacent layers only) *Multi-Seg* consists of vertical vias with lengths 1, 2 and long lines, which span all the layers of the FPGA. Our work assumes that length one vertical vias have the same delay as their 2D counterparts. Our architectures have one 4-input LUT per CLB (a common assumption made in previous works). More description about the CLBs can be found in [10]. Fig. 5.5 illustrates the basic elements of the 3D FPGA architecture.

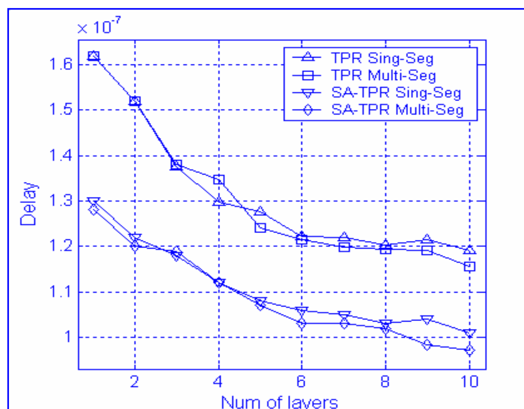


Figure 5.6: Variation of delay as reported after detailed routing.

5.5.2 Delay Results of TPR and SA-TPR Algorithms

We report comprehensive results on wire-length and circuit delay as well as metrics such as area, horizontal and vertical channel widths, and run-times on all twenty circuit benchmarks of the VPR package. We performed placement and detailed routing on all circuits (see Table 5.1) with the number of layers varying between one (the 2D case) and ten. We recorded the average circuit delay and the average total wire-length of four different runs for each circuit. The comparison between the variation of the average delay values obtained using partitioning-based (TPR) and SA-based (SA-TPR) placement algorithms is illustrated in Figure 5.6. We observe that delay decreases by about 30% for both architectures using either placement algorithm compared to the 2D case, although architecture *Multi-Seg* shows slightly better delays, as the number of layers increases beyond six. However, the difference is not large, mainly because the routing algorithm considers fully buffered routing resources, which leads to comparable delay values for both architectures. Delay achieved using the SA-based placement algorithm is smaller compared to the delay achieved using the partitioning-based placement algorithm, which is not surprising, because annealing takes longer runtimes.

We noticed that the amount of delay decreased in comparison to the 2D case for different

circuits and same number of layers can vary. For example, delay decreases by 27% for Pdc benchmark but only 18% for Spla benchmark when nine layers are used. We suspect this is due to the internal structure (such as higher connectivity) of Pdc as opposed to Spla, which leads to a larger fraction of nets spanning different number of layers. During our experiments we also noticed that benchmarks with large number of terminals (relative to the number of cells), such as Des (see Table I), tend to lead to more delay decrease compared to the 2D case. This can be explained by the fact that in the 2D case, the chip size necessary to accommodate all terminals is bigger than if the circuit had less terminals, i.e., final chip will have more ”white-space”, and therefore delays of nets involving input or output terminals will have larger routing delays in the 2D case.

5.5.3 Wire-length Results of TPR and SA-TPR Algorithms

Wire-length results of TPR on the *Multi-Seg* architecture is shown in Figure 5.7. Results for architecture *Sing-Seg* and using the SA-based placement algorithm are similar. We observe that wire-length after detailed routing decreases by 25% on average as the number of layers increases. If the length of the inter-layer via increases, then the total wire-length decrease will be less. This is mainly because the fraction of the vertical wire-length relative to the total wire-length will become significant and also the average net delay will increase due to bending (i.e., switches) of nets spanning more layers. Wire-length achieved using the SA-based placement algorithm is smaller compared to the wire-length achieved using the partitioning-based placement algorithm. We note that the decrease in total wire-length can have a favorable impact on routing congestion (hence channel width), as well as power dissipation (especially due to the fact that most of the power dissipated in FPGAs is due to interconnects, which account for more than 80% of the total area) as predicted in [50].

The decrease in total wire-length (WL) is directly related to a decrease in the average net wire-length, which in turn relates to the overall usage of routing resources and therefore to the circuit delay. Variations of the average net wire-length, and other metrics of interest

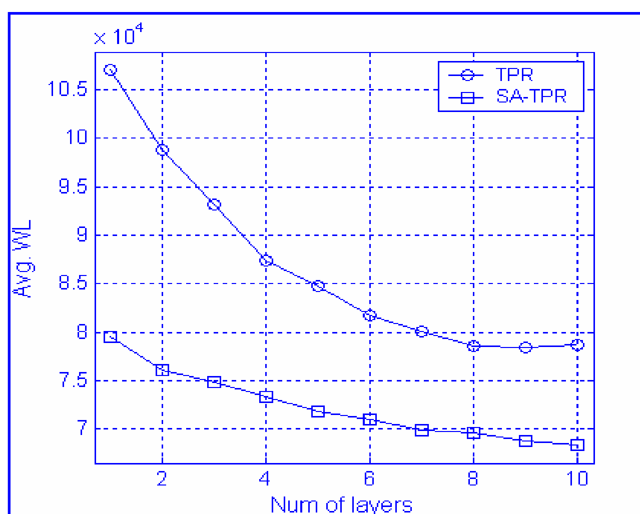


Figure 5.7: Variation of delay as reported after detailed routing.

are shown in Tables 5.2 and 5.3. Routing area counts the exact number of pass transistor and SRAM cells that control them, as suggested in [10]. We observe that the overall area, i.e., chip foot-print area multiplied by the number of layers, slightly increases for a small number of layers. This increase is due to the higher connectivity inside of a switch box (a track entering a 3D switch box will have to connect to 5 correspondents as opposed to only 3 in the 2D case; see Figure 5.8).

However, routing area decreases as the number of layers increases for Multi-Seg architecture, as a direct consequence of a decrease of the horizontal channel width (HCW) required for successful routing. Routing area increase is overall less than 10% when SA-based placement algorithm is used. Except for cases where only few layers are used, CW decreases significantly. The reason is that the number of vertical connections is minimized. In other words, the partitioning algorithm is able to find the clustering structure of the circuit and practically divide the initial netlist into a number of smaller circuits with similar internal structure (in terms of connectivity or Rent's parameter) to the initial one. As a consequence, the horizontal channel width for each layer will be in the same range as when the initial netlist is placed in 2D.

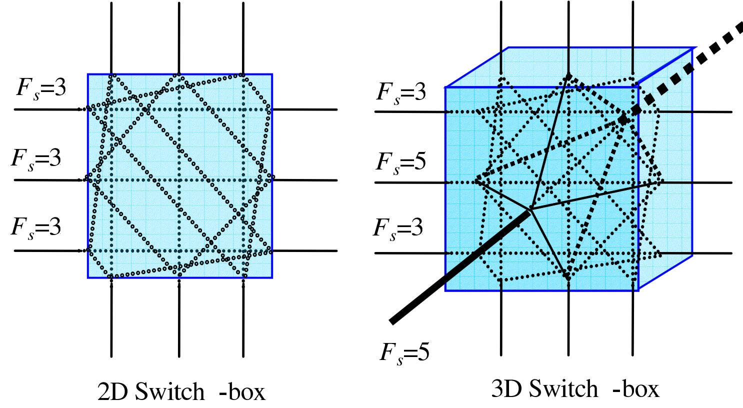


Figure 5.8: A 3D switch box. The third dimension adds vertical tracks which require 5 connections.

We observe that overall, run-times of SA-based placement are about twice the run-times of detailed routing (see Table 1) and about an order of magnitude longer than run-times of partitioning-based placement. Therefore, partitioning-based placement can be used for efficient solution space (especially for big circuits) and different architectural assumptions exploration. Also, the vertical channel widths reported in Table I are 1/5 of the horizontal channel widths, which demonstrates that our layer partitioning and linear placement as well as the routing algorithm are very well tuned to minimize the use of vertical tracks. Another advantage of using fewer vertical tracks greatly reduces the required area for switch-boxes. It should be noted that a vertical/horizontal intersection with $F_s = 3$ requires 6 ($\binom{4}{2}$) switches, while an intersection with $F_s = 5$ requires 15 ($\binom{6}{2}$) switches. Each switch has a pass transistor, an SRAM cell, and possibly a buffer.

5.5.4 Experiments using Hybrid-TPR

As we can see in Tables 5.2 and 5.3 Hybrid-TPR leads to a decrease in the wire-length when compared to TPR, whereas delay is virtually the same when compared to SA-TPR,

accompanied by a smaller horizontal channel width. Overall, this scheme achieves the best results especially when the number of layers is small. These results attest to the quality of the layer partitioning and linear placement scheme of TPR. As in most annealing based algorithms, a good initial solution can be very important and lead to better results as opposed to a random initial placement.

Table 5.1: Simulated circuits: statistics, vertical channel width (VCW) and run-time.

Circuit	No. CLBs	No. IOs	VCW		CPU (s)		
			TPR	SA-TPR	TPR-Placement	SA-TPR Placement	Detailed Routing
Ex5p	1064	71	6	5	7	85	77
Apex4	1262	28	6	5	8	105	76
Misex3	1397	28	6	5	9	55	84
Alu4	1522	22	5	5	16	145	61
Des	1591	501	5	5	11	227	69
Seq	1750	76	5	5	12	212	114
Apex2	1878	42	5	5	13	243	148
Spla	3690	62	5	6	37	912	532
Pdc	4575	56	7	7	60	1354	1039
Ex1010	4598	20	4	5	56	1238	273
Dsip	1370	426	5	5	28	154	34
Tseng	1407	174	5	5	8	70	14
Diffeq	1497	103	5	5	14	154	46
Bigkey	1707	426	5	5	22	209	48
S298	1931	10	5	5	23	208	53
Frisc	3556	136	5	5	56	881	227
Elliptic	3604	245	5	5	74	818	142
S38417	6406	135	5	5	133	2000	210
S38584.1	6447	342	5	5	230	2034	268
Clma	8383	144	5	5	199	892	950
				Sum	1016	11996	4465

Table 5.2: Average of delay, wire-length (WL), horizontal channel width (HCW) and routing area for the *Sing-Seg* architecture.

Num. Layers	TPR				SA-TPR				Hybrid-TPR			
	Delay ($\times 10^{-7}$)	WL	Routing Area	HCW	Delay ($\times 10^{-7}$)	WL	Routing Area	HCW	Delay ($\times 10^{-7}$)	WL	Routing Area	HCW
1	1.62	107087.6	1	1	1.3	79553.76	1	1	1.27	79626.34	1	1
2	1.52	98808.3	1.088	0.92	1.22	76072.43	1.016	0.95	1.18	73663.53	1.113	0.99
3	1.37	93162.93	1.097	0.89	1.18	74786.49	1.096	0.96	1.13	69913.86	1.114	0.90
4	1.3	87410.88	1.096	0.85	1.12	73277.69	1.091	0.93	1.1	67746.77	1.108	0.85
5	1.28	84741.11	1.041	0.77	1.08	71817.41	1.117	0.93	1.06	68154.17	1.119	0.85
6	1.22	81707.36	1.089	0.74	1.06	70975.92	1.106	0.91	1.05	67045.21	1.121	0.81
7	1.22	80143.2	1.065	0.70	1.05	69902.95	1.116	0.89	1.03	65753.97	1.119	0.78
8	1.2	78589.86	1.005	0.71	1.03	69589.44	1.096	0.87	1.01	67361.31	1.117	0.80
9	1.21	78456.85	1.018	0.67	1.04	68800.65	1.100	0.87	1.04	66559.84	1.117	0.77
10	1.19	78643.86	1.072	0.69	1.01	68411.58	1.084	0.85	1.02	66185.75	1.122	0.77

5.6 Conclusion

Benefits which 3D FPGA integration can offer were analyzed using the placement and detailed routing tool called TPR. Placement can be done using either a partitioning-based or a simulated annealing based approach. Simulation experiments, after detailed routing,

Table 5.3: Average of delay, wire-length (WL), horizontal channel width (HCW) and routing area for the *Multi-Seg* architecture.

Num. Layers	TPR				SA-TPR				Hybrid-TPR			
	Delay ($\times 10^{-7}$)	WL	Routing Area	HCW	Delay ($\times 10^{-7}$)	WL	Routing Area	HCW	Delay ($\times 10^{-7}$)	WL	Routing Area	HCW
1	1.62	107087.6	1	1	1.28	79410.53	1	1	1.27	79626.34	1	1
2	1.52	98808.3	1.088	0.92	1.2	76088.11	1.015	0.96	1.18	73663.53	1.113	0.99
3	1.38	93162.93	1.096	0.89	1.19	75304.16	1.068	0.96	1.14	70729.52	1.112	0.91
4	1.35	87410.88	1.053	0.83	1.12	73796.21	1.075	0.95	1.07	67383.43	1.107	0.86
5	1.24	84741.11	1.025	0.78	1.07	72228.37	1.045	0.90	1.06	66798.35	1.113	0.82
6	1.21	81707.36	1.038	0.74	1.03	70888.9	1.064	0.90	1.05	66682.31	1.115	0.80
7	1.2	80143.2	1.003	0.70	1.03	70710.4	1.052	0.88	1.03	65605.21	1.115	0.77
8	1.19	78589.86	0.942	0.70	1.02	69849.53	1.054	0.87	0.989	67049.56	1.113	0.81
9	1.19	78456.85	0.967	0.67	0.9.84	69190.21	1.062	0.87	0.987	65536.51	1.114	0.77
10	1.16	78643.86	0.981	0.68	0.971	68840.17	1.058	0.85	0.987	65439.38	1.115	0.77

showed potential total decrease of 25% for wire-length and 35% for delay using either the partitioning-based algorithm or the SA-based algorithm. We observed that the *Multi-Seg* architecture shows slightly better delay characteristics compared to the *Sing-Seg* architecture. We also augmented the tool with a hybrid approach consisting of a partitioning based initial placement followed by a simulated annealing based layer placement. Source code and documentation of the implementation of the algorithms presented in this paper are available for download at: http://www.ece.umn.edu/users/kia

Chapter 6

Conclusion

Overall, this thesis attempted to tackle some of the important challenges facing the current day semi-conductor industry. This is in light of the fact that EDA tools need to keep up with the increasingly complex nature of silicon manufacturing. The process, voltage and temperature effects that result make it increasingly difficult and problematic to design modern day digital integrated circuits (IC) in a timely fashion. With increasing market pressures, it becomes more imperative for design tools to provide efficient techniques to solve such issues.

The first part of this thesis dealt with process variation effects in the sub-nanometer regime. We described a novel algorithm to determine the criticality of gates in a digital circuit, within an SSTA framework. The criticality of a gate is defined as the probability that it lies on the critical path of some manufactured die. We achieved this by using cutsets and dividing the timing graph into zones. This helped us analyze the circuit in time linear in the number of elements it contains. Although efficient, such an algorithm had limitations in terms of its accuracy, mainly stemming from the fact that we used Clark's formulation to compute the maximum of a set of Gaussian random variables. Such sources of error, categorized as local and global errors, during criticality computation were analyzed and

a new clustering-based algorithm developed to aptly deal with them. A fast yet accurate technique was used to deal with structural correlations of random variations. Finally, a localized sampling-based scheme was deployed to further improve the accuracy of our criticality computation. One direction of future research is to adapt our clustering-based scheme in a timing optimization framework to see its benefits.

In the second part of the thesis, we dealt with temperature effects in modern day digital ICs. The main thrust of our work was to reduce the sub-threshold leakage power which is exponentially dependent on the temperature. We showed how static floorplanning for microarchitectures can be used to effectively distribute the blocks on-chip, such that the leakage of the chip is minimized. Towards this end, we developed two novel methods of modeling the temperature-dependent rise of leakage power during the floorplanning process, that avoid the expensive iterative computation loops needed for convergence of the thermal solution.

The third part of the thesis dealt with reducing the sub-threshold leakage power of a design by employing the idea of task migration, wherein hot modules are replicated so as to redistribute their power over regular intervals of time. We first defined the problem of task migration within a floorplanning framework followed by a novel solution to solve the chicken-and-egg nature of the problem. An important aspect of our work is the implications it has for current computing trends with multi-core processors.

The final part of this thesis described a new framework to analyze three-dimensional (3D) integration technology. Our work focused on the application of 3D ICs to Field Programmable Gate Arrays (FPGAs). We developed new algorithms for the placement of FPGAs using the Simulated Annealing paradigm that can serve as a future reference for other innovative fast approaches to 3D placement.

Bibliography

[1] Industry summary for 3d ics. [Online]. Available: http://www.tezzaron.com/technology/3D_IC_Summary.html

[2] Predictive technology model (PTM). [Online]. Available: http://www.eas.asu.edu/~ptm/

[3] Bsim3v3 MOSFET model - user's manual. [Online]. Available: http://www-device.eecs.berkeley.edu/~bsim3

[4] Power management enhancements in the 45nm intel core microarchitecture. [Online]. Available: http://www.intel.com/technology/itj/2008/v12i3/2-paper/1-abstract.htm

[5] C. Ababei, Y. Feng, B. Goplen, H. Mogal, T. Zhang, K. Bazargan, and S. Sapatnekar, "Placement and routing in 3D integrated circuits," *IEEE Design & Test of Computers*, vol. 22, no. 6, pp. 520–531, Nov./Dec. 2005.

[6] C. Ababei and K. Bazargan, "Non-contiguous linear placement for reconfigurable fabrics." in *IPDPS '04: Proceedings of the 18th International Symposium on Parallel and Distributed Processing*, 2004.

[7] C. Ababei, H. Mogal, and K. Bazargan, "Three-dimensional place and route for FPGAs." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1132–1140, 2006.

[8] S. Adya and I. Markov, "Fixed-outline floorplanning: enabling hierarchical design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 1120–1135, Dec 2003.

[9] M. J. Alexander, J. P. Cohoon, J. L. Colflesh, J. Karro, and G. Robins, "Three-dimensional field-programmable gate arrays," in *ASIC '95: Proceedings of the Eighth Annual IEEE International Conference and Exhibit*, Austin, TX, Sep 1995, pp. 253–256.

[10] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Cambridge, MA: Kluwer Academic Publishers, 1999.

[11] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for fpga research," in *FPL '97: Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 1997, pp. 213–222.

[12] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *ISCA '00: Proceedings of the 27th Annual International Symposium on Computer Architecture*. New York, NY, USA: ACM Press, 2000, pp. 83–94.

[13] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *SIGARCH Comput. Archit. News*, vol. 25, no. 3, pp. 13–25, 1997.

[14] J. A. Burns, B. F. Aull, C. K. Chen, C.-L. Chen, C. L. Keast, J. M. Knecht, V. Suntharalingam, K. Warner, P. W. Wyatt, and D. R. W. Yost, "A wafer-scale 3-d circuit integration technology," *IEEE Transactions on Electron Devices*, vol. 53, no. 10, pp. 2507–2516, Oct. 2006.

[15] H. Chang and S. S. Sapatnekar, "Statistical timing analysis under spatial correlations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 9, pp. 1467–1482, Sept. 2005.

[16] P. Chaparro, J. Gonzalez, and A. Gonzalez, "Thermal-aware clustered microarchitectures," in *ICCD '04: Proceedings of the 2004 IEEE International Conference on Computer Design*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 48–53.

[17] S. Chiricescu, M. Leeser, and M. M. Vai, "Design and analysis of a dynamically reconfigurable three-dimensional FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 186–196, Feb 2001.

[18] K. Chopra, S. Shah, A. Srivastava, D. Blaauw, and D. Sylvester, "Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation," in *Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1023–1028.

[19] C. E. Clark, "The greatest of a finite set of random variables," *Operations Research*, vol. 9, no. 2, pp. 145–162, Mar-Apr 1961.

[20] J. Cong, A. Jagannathan, G. Reinman, and M. Romesis, "Microarchitecture evaluation with physical planning," in *DAC '03: Proceedings of the 40th Conference on Design Automation*. New York, NY, USA: ACM Press, 2003, pp. 32–35.

[21] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. Cambridge, MA: The MIT Press, 1989.

[22] S. Das, A. Chandrakasan, and R. Reif, "Design tools for 3-d integrated circuits," in *ASPDAC '03: Proceedings of the 2003 Conference on Asia South Pacific Design Automation*. New York, NY, USA: ACM Press, 2003, pp. 53–56.

[23] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns, "Placement and routing tools for the triptych FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 4, pp. 473–482, 1995.

[24] M. C. Golumbic, *Algorithmic Graph Theory and Pefect Graphs*. Boston, MA: Elsevier, 2004.

[25] M. Gomaa, M. D. Powell, and T. N. Vijaykumar, "Heat-and-run: leveraging smt and cmp to manage power density through the operating system," in *ASPLOS-XI: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM Press, 2004, pp. 260–270.

[26] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, no. 2-3, pp. 293–306, 1985.

[27] B. Goplen and S. Sapatnekar, "Efficient thermal placement of standard cells in 3d ics using a force directed approach," in *ICCAD '03: Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design*. Washington, DC, USA: IEEE Computer Society, 2003, p. 86.

[28] U. I. Gupta, D. T. Lee, and J. Y.-T. Leung, "Efficient algorithms for interval graphs and circular-arc graphs," *Networks*, vol. 12, no. 4, pp. 459–467, 1982.

[29] Y. Han, I. Koren, and C. A. Moritz, "Temperature aware floorplanning," in *TACS-2: Second Workshop on Temperature-Aware Computer Systems*, 2005.

[30] S. Heo, K. Barr, and K. Asanovic;, "Reducing power density through activity migration," in *ISLPED '03: Proceedings of the 2003 International Symposium on Low Power Electronics and Design*. New York, NY, USA: ACM Press, 2003, pp. 217–222.

[31] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[32] J. Kao and A. Chandrakasan, "MTCMOS sequential circuits," in *ESSCIRC '01: Proceedings of the 27th European Solid-State Circuits Conference*, Sept. 2001, pp. 317–320.

[33] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: application in VLSI domain," in *DAC '97: Proceedings of the 34th Annual Conference on Design Automation*. New York, NY, USA: ACM Press, 1997, pp. 526–529.

[34] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[35] T. Kuroda, T. Fujita, S. Mita, T. Nagamatsu, S. Yoshioka, K. Suzuki, F. Sano, M. Norishima, M. Murota, M. Kako, M. Kinugawa, M. Kakumu, and T. Sakurai, "A 0.9-V, 150-MHz, 10-mW, 4 mm2, 2-d discrete cosine transform core processor with variable threshold-voltage (VT) scheme," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1770–1779, Nov. 1996.

[36] M. Leeser, W. Meleis, M. M. Vai, S. M. S. A. Chiricescu, W. Xu, and P. M. Zavracky, "Rothko: A three-dimensional fpga." *IEEE Design & Test of Computers*, vol. 15, no. 1, pp. 16–23, 1998.

[37] X. Li, J. Le, P. Gopalakrishnan, and L. T. Pileggi, "Asymptotic probability extraction for nonnormal performance distributions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 1, pp. 16–37, Jan. 2007.

[38] X. Li, J. Le, M. Celik, and L. T. Pileggi, "Defining statistical timing sensitivity for logic circuits with large-scale process and environmental variations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1041–1054, June 2008.

[39] W. Liao, L. He, Lepak, and K.M., "Temperature and supply voltage aware performance and power modeling at microarchitecture level," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1042–1053, July 2005.

[40] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," in *ITC '02*, Oct. 7–10, 2002, pp. 519–528.

[41] A. S. Marquardt, V. Betz, and J. Rose, "Using cluster-based logic blocks and timing-driven packing to improve fpga speed and density," in *FPGA '99: Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*. New York, NY, USA: ACM Press, 1999, pp. 37–46.

[42] K. Melhorn and S. Naher, *Leda: A platform for combinatorial and geometric computing*. New York, NY: Cambridge University Press, 1999.

[43] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1518–1524, Dec 1996.

[44] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, and J. Yamada, "1V high-speed digital circuit technology with 0.5 μm m multi-threshold CMOS," in *ASIC '93: Proceedings of the Sixth Annual IEEE International Conference and Exhibit*, Rochester, NY, Sept./Oct. 1993, pp. 186–189.

[45] S. G. Narendra and A. Chandrakasan, *Leakage in Nanometer CMOS Technologies*. Newyork, NY: Springer US, 2005.

[46] S. R. Nassif, "Design for variability in DSM technologies," in *Proceedings of the 2000 IEEE First International Symposium on Quality of Electronic Design*. Washington, DC, USA: IEEE Computer Society, 2000, p. 451.

[47] V. Nookala, Y. Chen, D. J. Lilja, and S. S. Sapatnekar, "Microarchitecture-aware floorplanning using a statistical design of experiments approach," in *DAC '05: Proceedings of the 42nd Annual Conference on Design Automation*. New York, NY, USA: ACM Press, 2005, pp. 579–584.

[48] S. T. Obenaus and T. H. Szymanski, "Gravity: Fast placement for 3-d VLSI," *ACM Transactions on Dessign Automation of Electronic Systems*, vol. 8, no. 3, pp. 298–315, 2003.

[49] E. H. A. P. J. van Laarhoven, *Simulated Annealing: Theory and Applications*. Norwell, MA: Kluwer Academic, 1987.

[50] A. Rahman, S. Das, A. Chandraksan, and R. Reif, "Wiring requirement and three-dimensional integration of field-programmable gate arrays," in *SLIP '01: Proceedings of the 2001 International Workshop on System-level Interconnect Prediction*. New York, NY, USA: ACM Press, 2001, pp. 107–113.

[51] R. Reif, A. Fan, K.-N. Chen, and S. Das, "Fabrication technologies for three-dimensional integrated circuits (invited)." in *ISQED '02: Proceedings of the 3rd International Symposium on Quality Electronic Design*, 2002, pp. 33–37.

[52] K. Sankaranarayanan, S. Velusamy, M. Stan, and K. Skadron, "A case for thermal-aware floorplanning at the microarchitectural level," *The Journal of Instruction-Level Parallelism*, vol. 7, Oct 2005. [Online]. Available: http://www.jilp.org/vol7/

[53] S. S. Sapatnekar, *Timing*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2004.

[54] A. Shayesteh, E. Kursun, T. Sherwood, S. Sair, and G. Reinman, "Reducing the latency and area cost of core swapping through shared helper engines," in *ICCD '05: Proceedings of the 2005 International Conference on Computer Design*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 17–23.

[55] SIA. International technology roadmap for semiconductors. [Online]. Available: http://www.itrs.net

[56] D. Sinha, H. Zhou, and N. V. Shenoy, "Advances in computation of the maximum of a set of gaussian random variables," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 8, pp. 1522–1533, Aug. 2007.

[57] SPEC, *Standard Performance Evaluation Corporation CPU2000 Benchmarks*. [Online]. Available: http://www.specbench.org/osg/cpu2000

[58] C.-H. Tsai and S.-M. Kang, "Cell-level placement for improving substrate thermal distribution," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 2, pp. 253–266, Feb. 2000.

[59] C.-H. Tsai and S.-M. S. Kang, "Standard cell placement for even on-chip thermal distribution," in *ISPD '99: Proceedings of the 1999 International Symposium on Physical Design*. New York, NY, USA: ACM Press, 1999, pp. 179–184.

[60] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett, "First-order incremental block-based statistical timing analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2170–2180, Oct. 2006.

[61] C. Visweswariah and A. R. Conn, "Formulation of static circuit optimization with reduced size, degeneracy and redundancy by timing graph manipulation," in *ICCAD '99: Proceedings of the 1999 IEEE/ACM International Conference on Computer-Aided Design*. Piscataway, NJ, USA: IEEE Press, 1999, pp. 244–252.

[62] F. Wang, Y. Xie, and H. Ju, "A novel criticality computation method in statistical timing analysis," in *DATE '07: Proceedings of the 2007 Conference on Design, Automation and Test in Europe*. San Jose, CA, USA: EDA Consortium, 2007, pp. 1611–1616.

[63] Y.-W. Wu, C.-L. Yang, P.-H. Yuh, and Y.-W. Chang, "Joint exploration of architectural and physical design spaces with thermal consideration," in *ISLPED '05: Proceedings of the 2005 International Symposium on Low Power Electronics and Design*. New York, NY, USA: ACM Press, 2005, pp. 123–126.

[64] Z. Xiang, "Color image quantization by minimizing the maximum intercluster distance," *ACM Transactions on Graphics*, vol. 16, no. 3, pp. 260–276, 1997.

[65] J. Xiong, V. Zolotov, N. Venkateswaran, and C. Visweswariah, "Criticality computation in parameterized statistical timing," in *DAC '06: Proceedings of the 2006 IEEE/ACM Design Automation Conference*. New York, NY, USA: ACM Press, 2006, pp. 63–68.

[66] Y. Ye, S. Borkar, and V. De, "A new technique for standby leakage reduction in high-performance circuits," in *1998 Symposium on VLSI Circuits*, Honolulu, HI, June 1998, pp. 40–41.

[67] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 1, no. 1, pp. 25–35, Jan. 1982.

[68] L. Zhang, W. Chen, Y. Hu, and C. Chen, “Statistical static timing analysis with conditional linear max/min approximation and extended canonical timing model,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 8, pp. 1522–1533, Aug. 2007.