Circuit Timing and Leakage Analysis in the Presence of Variability

by

Khaled R. Heloue

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

Circuit Timing and Leakage Analysis in the Presence of Variability

Khaled R. Heloue

Doctor of Philosophy, 2010

Graduate Department of Electrical and Computer Engineering

University of Toronto

Abstract

Driven by the need for faster devices and higher transistor densities, technology trends have pushed transistor dimensions into the deep sub-micron regime. This continued scaling, however, has led to many challenges facing digital integrated circuits today. One important challenge is the increased variations in the underlying process and environmental parameters, and the significant impact of this variability on circuit timing and leakage power, making it increasingly difficult to design circuits that achieve a required specification. Given these challenges, there is a need for computer-aided design (CAD) techniques that can predict and analyze circuit performance (timing and leakage) accurately and efficiently in the presence of variability. This thesis presents new techniques for variation-aware timing and leakage analysis that address different aspects of the problem.

First, on the timing front, a pre-placement statistical static timing analysis technique is presented. This technique can be applied at an early stage of design, when

within-die correlations are still unknown. Next, a general parameterized static timing analysis framework is proposed, which supports a general class of nonlinear delay models and handles both random (process) parameters with arbitrary distributions and non-random (environmental) parameters. Following this, a parameterized static timing analysis technique is presented, which can capture circuit delay exactly at any point in the parameter space. This is enabled by identifying all potentially critical paths in the circuit through novel and efficient pruning algorithms that improve on the state of art both in theoretical complexity and runtime. Also on the timing front, a novel distance-based metric for robustness is proposed. This metric can be used to quantify the susceptibility of parameterized timing quantities to failure, thus enabling designers to fix the nodes with smallest robustness values in order to improve the overall design robustness.

Finally, on the leakage front, a statistical technique for early-mode and late-mode leakage estimation is presented. The novelty lies in the *random gate* concept, which allows for efficient and accurate full-chip leakage estimation. In its simplest form, the leakage estimation reduces to finding the area under a scaled version of the within-die channel length auto-correlation function, which can be done in constant time.

Acknowledgments

I would first like to thank Professor Farid Najm for his continuous guidance and support throughout the years of my graduate studies. I owe a great deal of this work to his contributions, patience, and unwavering trust. Working under his supervision has been a great and unique experience, one that will always bring back dear memories to my heart.

I thank Professors Jason Anderson, David Blaauw, Tony Chan Carusone, Paul Chow, and Jianwen Zhu for their constructive comments and review of this work.

During my Ph.D, I also had the chance to work and interact with excellent mentors who helped me bridge the "annoying" gap between theory and practice. I am grateful to Chandramouli Kashyap, Eli Chiprout, and Noel Menezes from Intel for their guidance, support, and direct contributions to several aspects of this work. I also want to thank Edwin Bender from AMD for his enthusiasm and interest during the last stages of my Ph.D. The many "2-minute" discussions we had helped me gain a better understanding of the bigger picture.

I am grateful to Navid Azizi for his direct contributions to Chapter 7 of this work. Thank you Navid for being a dedicated colleague and friend. I also want to thank Sari Onaissi, my partner in the "timing group", for his contributions to Chapter 5 of this dissertation, and for all the good times we spent in and out of the "arena".

I would like to thank Nahi Abdul Ghani for the many funny "moments" we shared, and for being a great friend and colleague. Of my friends at UofT, I also want to thank Hratch Mangassarian, Hayssam Dahrouj, Andrew Ling, Ian Kuon, Frank Plavec, Tom Czajkowski, Mehmet Avci, Ankit Goyal, and Meric Aydonat, who made my stay in the lab a pleasant one.

I also want to thank my mother Samar, father Reslan, sisters Karine and Reine,

and brother Majed for their immense love and support at every stage of my life. And to my wife Rasha, I say thank you for your patience and love, and above all for always being there for me.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivation

With the continued scaling of integrated circuits, the control over process and environmental variations has become increasingly difficult. As we move from one process technology to the next, variability has steadily increased and is now a major challenge facing digital integrated circuit design. The crux of the problem lies in that circuit performance, including timing and leakage power, is significantly impacted by the variability in the underlying physical and electrical parameters, to such extent that it is becoming increasingly difficult to design circuits that will achieve their target specifications with high yield. Based on predictions forecasted by the International Technology Roadmap for Semiconductors (ITRS), circuit timing variability is expected to reach 60% and leakage power variability is expected to reach 3× in the near future. These alarming levels call for new techniques for variation-aware performance analysis and verification, that can quantify accurately and efficiently the impact of variations on performance and guide circuit design and optimization to achieve more reliable and robust designs.

Process and environmental variations are not new, and taking care of their impact on timing is not a new problem. Traditionally, this has been taken care of in various ways. Application Specific Integrated Circuits (ASICs) are typically designed by making sure the chip passes the timing requirements at all *process corners*. This is done by setting certain transistor parameters at $\pm 3\sigma$ (typically, but could be higher) of their range, as defined in the process files, and running static timing analysis (STA) on the circuit. If these settings are too pessimistic, then designers are forced to waste time and effort optimizing a circuit using design conditions that are too stringent. Microprocessors

and FPGAs, on the other hand, use "frequency binning," so that a design that does not meet the worst-case corner specs is not necessarily failed. For microprocessors, it is typical to check circuit timing with *nominal* transistor files, and to specify some *timing margin* which should be left (as slack) to account for (some) process variations. In some cases, high-speed datapaths of microprocessors may be designed with *zero* timing margin, due to the difficulty of optimizing such paths any further and the fact that one will do frequency binning anyway. This is not to say, however, that timing variations are not relevant in microprocessor design. On the contrary, it turns out that within-die variations greatly impact microprocessor design because they cause unwanted mismatch (such as clock skew) which leads to timing violations. However, the ASICs versus microprocessors contrast highlights the two traditional methods of accounting for process variations by either using worst-case process files and running STA with a zero timing margin, or using nominal process files and running STA using a (possibly zero) timing margin.

However, today, and in future, this situation is changing in several ways. Firstly, it is not clear that we will be able to control process variations in future to the same extent that we have done in the past. This is especially true for threshold voltage variations arising from random dopant fluctuations. Thus, variations may be worse in future, so that we need to do a better job of choosing the timing margins for good design, avoiding over-pessimism. Secondly, within-die variations are becoming more significant, and they currently represent a larger component of the total parameter variation. For within-die variations, it becomes harder to figure out what exactly the timing margin should be and, overall, it becomes harder to predict the effect of variations on the design. Another trend is the increase in the number of parameters that impact performance. These include device parameters (such as channel length, threshold voltage, and oxide thickness) and interconnect parameters (such as wire height and width for various metal layers).

There has been considerable discussion in the literature that the traditional methods of using process corners or using a timing margin are breaking down. For ASICs, the number of corners is increasing, making it very expensive to explore all corners. Also, the corner-based methodology does not provide the user with any quantitative feedback

on the sensitivity of the design to variations; it is a pass/fail approach. Furthermore, this traditional approach cannot handle within-die variations. On the other hand, for microprocessors, where a timing margin needs to be left as slack, there are no easy ways to decide what the margin should be, to account for within-die variations.

In the past few years, a large body of research has focused on tackling the variation-aware timing problem statistically, and has led to what is known as statistical static timing analysis (SSTA). The basic idea is to start by modeling process parameters as random variables with known distributions. Then, by using a variational timing model for cell and wire delay, SSTA techniques propagate delay *distributions* instead of *deterministic* delay values in the timing graph. In this way, distributions for path delays and arrival times are determined, and ultimately the maximum circuit delay distribution, which is directly translated into a timing yield, is also determined. However, difficulties still remain in the adoption of SSTA. First, most proposed techniques require the existence of extensive process and placement information to operate, which makes them valid as final sign-off tools and unusable at an early stage of the design flow. Second, there is no easy way to handle environmental variations, such as supply voltage and temperature, in the context of SSTA frameworks; these types of variations, which depends on circuit operation and not on manufacturing variations, are not *random* but *uncertain*, and will have to be modeled differently.

On the leakage front, and as mentioned earlier, leakage power is expected to continue to increase and due to limited power budgets, it may affect the feasibility of future microprocessor and ASIC designs. In addition, with the drastic impact of variability on leakage power, as predicted by ITRS, it becomes clear why variation-aware leakage estimation techniques become increasingly important; these techniques will be needed both at an early stage for design planning, and at a late stage for final sign-off.

1.2 Thesis Contributions

This thesis focuses on the following three overarching themes:

1. Analysis and prediction of the impact of process and environmental variations on circuit timing.

2. Defining metrics for circuit timing robustness in the presence of variability.

3. Analysis and prediction of full-chip leakage current under process variations.

With respect to these themes, a number of different contributions are made, as summarized below:

Chapter 3 presents a statistical static timing analysis technique that can operate at an early stage of the design flow. Unlike current approaches to statistical timing analysis which operate post-placement and rely on the existence of complex within-die correlation models built from process data, the technique proposed in Chapter 3 operates pre-placement, at a stage where within-die spatial correlations are still unknown or unavailable. Starting with a simple variational delay model that requires minimal input from the user, the distribution of the maximum circuit delay is bounded, such that the bounds are valid for any arbitrary within-die correlation. An important contribution of this work is the use of these bounds to introduce the concept of *margin uncertainty*, resulting from the correlation uncertainty of the within-die variations. This margin range can be used by designers at an early stage of design to mitigate the impact of variations on timing. This work has been published in [4].

Chapter 4 proposes a general framework for *parameterized static timing analysis*, a term used to describe any variation-aware timing analysis technique in which delay is an explicit function of the underlying process and environmental parameters. The proposed framework is general, in the sense that it can handle statistical (random) process parameters with arbitrary distributions as well as non-random (uncertain) parameters such as supply voltage and temperature. In addition, the framework presented in Chapter 4 supports a general class of nonlinear variational delay models, including linear and quadratic models. When applied to random variations, this framework is competitive with existing statistical static timing analysis techniques, in that it allows for nonlinear delay models and arbitrary parameter distributions. When applied to uncertain non-random variations, the framework is competitive with existing multi-corner static timing

analysis techniques, in that it more reliably reproduces overall circuit sensitivity to variations. Crucially, this technique can also be applied to the mixed case where both random and uncertain variations are considered. This work has been published in [5].

Chapter 5 presents an efficient block-based parameterized static timing analysis technique that can accurately capture circuit delay at every point in the process and environmental parameter space, by reporting all the paths that can become critical at any parameter setting. The technique models parameters as variables specified in ranges; this is a more general model for parameters because it can represent random process parameters with arbitrary distributions and non-random (uncertain) environmental parameters (such as supply voltage and temperature) alike. An efficient pruning algorithm is proposed, where only those *potentially critical* paths are carried forward, while all other (non-critical) paths are discarded during propagation. This allows one to examine local sensitivities to parameters in different regions of the parameter space, not by considering differential sensitivity at a point but by knowledge of the paths that can become critical at nearby points in parameter space. Chapter 5 also gives a formal definition of this problem and proposes a technique for solving it that improves on the state of the art, both in terms of theoretical computational complexity and in terms of run time on various test circuits. This work has been published in [6].

Chapter 6 proposes a novel metric that can be used to quantify the timing *robustness* of designs to parameter variations. This type of robustness analysis can be used as a "post-variation" analysis step, that is, after the parameterized static timing analysis step is complete. The metric proposed in Chapter 6 works with parameterized static timing analysis techniques where parameters are modeled as uncertain variables specified as ranges, such as the techniques proposed in Chapter 4 and 5. The *distance to failure* is used as a measure for robustness; thus the analysis determines the minimum distance from the nominal point in the parameter space to any timing violation, and works under the assumption that parameters are specified as ranges rather than statistical distributions. In

addition to helping designers diagnose *if* and *when* different nodes can fail, this metric can guide optimization and can give insights on *what to fix*, by identifying nodes with small robustness values and proceeding to fix those nodes first. The usefulness of this distance-based robustness metric is demonstrated on circuit blocks extracted from a commercial 45nm microprocessor from Intel. This work has been published in [7].

Chapter 7 presents a probabilistic framework for full-chip leakage estimation in the presence of process variations. The proposed technique enables efficient prediction of the the mean and variance of the leakage current of a candidate design, while considering logic-structures and both die-to-die and within-die process variations, and taking into account the spatial correlation due to within-die variations. This framework can be used as either an *early* or a *late* estimator of leakage, with high accuracy. The full-chip leakage model is based on a novel *random gate* concept to capture high-level characteristics of a candidate chip design, which are sufficient to determine its leakage. These high-level characteristics include information about the process, the standard cell library, and expected design characteristics. Chapter 7 demonstrates empirically that, for large gate count, the set of all chip designs that share the same high level characteristics have approximately the same leakage, with very small error. In its simplest form, the full-chip leakage estimation reduces to finding the area under a scaled version of the within-die channel length auto-correlation function, which can be done in constant time. This work has been published in [8, 9].

1.3 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 covers the background material relevant to the research, including a description of process and environmental variations and their impact on performance (timing and leakage), an overview of current techniques for timing analysis under variability, and coverage of the dominant leakage current mechanisms that are impacted by variability.

The main research contributions, which are highlighted above, are presented in Chapters 3 through 7. For clarity, and owing to the range of topics considered, each chapter is self-contained; each chapter includes a description of the proposed technique, additional specific background material where applicable, and the experimental results. Chapter 8 presents concluding remarks and suggestions for future work.

2 Background

2.1 Introduction

Keeping up with Moore's Law, digital integrated circuits have continued to scale with every new technology node, and are now facing increased manufacturing process and environmental variations. This variability in the physical and electrical parameters greatly impacts circuit performance (timing and power) and calls for "variation-aware" performance verification techniques. In this chapter, a general background will be provided, covering the material that forms the basis for the research presented in later chapters. We first go over the different types of variations and cover some design for manufacturability techniques to reduce them, then we move to describe various methods for variation-aware timing verification, and finally present some background material on leakage power and how it is impacted by variability. Chapters 3 through 7 will also contain more background material and literature review that are specific to the topic of each chapter.

2.2 Process and Environmental Variations

Generally speaking, variations are deviations from the typical (intended) values of the electrical and physical parameters of a transistor or a circuit element, and are usually divided into two broad categories [10]: manufacturing process variations, or simply *process variations*, are permanent variations in device or wire parameters due to manufacturing tolerances caused by the lack of full control over the fabrication process. Examples of process parameters affected by manufacturing variations include transistor length and width, threshold voltage, and oxide thickness. On the other

Table 2.1: Examples of variations

Component	Form of variation
Channel length	dd, wds + wdr
Threshold Voltage	dd, wdr
Mean R and C differences between metal layers	dd
Vdd and temperature	wds

hand, *environmental variations* are operation dependent variations that can affect the circuit while it is functioning. This type of variation depends on the circuit's switching activity and modes of operation, and include variations in the supply voltage and temperature [10]. Collectively, process and environmental variations are referred to as process, voltage, and temperature variations, or simply *PVT variations*.

2.2.1 Components of Variations

A detailed description of process and environmental variations is presented in [11]. Variations can be broken down into two components. First, *die-to-die* variations, also known as *inter-die* variations, consist of differences in transistor parameters across different dies, irrespective of whether the dies are from different lots, wafers, or belong to the same wafer. These variations usually cause a shift in the mean value of a parameter over all the devices on a single die. Die-to-die variations are often caused by systematic effects across the wafer, however they can be modeled as random variations from one die to the next [10]. For a given die, these variations are global in the sense that they are shared by all devices; they are perfectly correlated and are often referred to as die-to-die systematic variations. Second, *within-die* variations, also known as *intra-die* variations, are local variations that occur spatially within a die, and that affect every

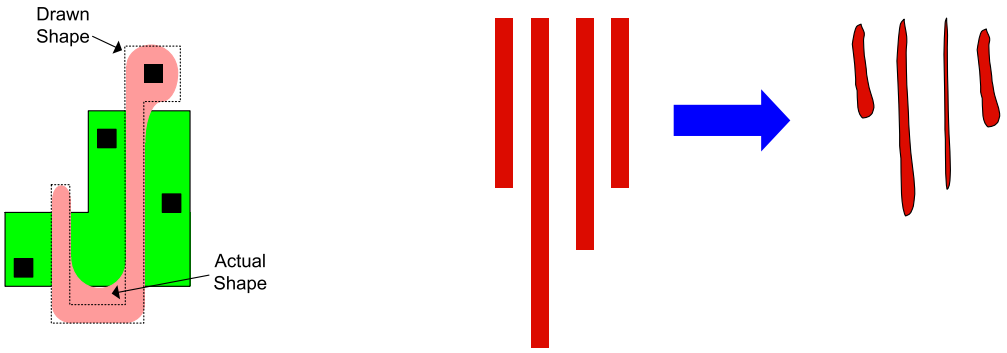


Figure 2.1: Effects of lithography limitations on drawn features [1]

transistor or circuit element differently. These variations are further broken down into a spatially correlated component known as *within-die systematic* variation, and a random (independent) component that is totally uncorrelated and independent of all else. These variations can be quite problematic since they can reduce the matched behaviour of different structures on the die needed to maintain correct circuit functionality [10].

Examples of die-to-die variations include channel length variation due to length of exposure, and variations between individual metal layers used for routing [11]. Within-die systematic variations come about because of layout-specific variations. These variations can be the result of semiconductor process methods or environmental differences that are seen across the design based on layout. Examples of within-die systematic variations include optical proximity effects that cause polysilicon feature sizes such as channel length to vary as a function of local layout. Also, they can be the cause of spatial variation of channel length due to lens aberration across the die. Finally, within-die random variations can be caused by any number of things including lithography, etching, polishing, and doping effects. An example of within-die random variation is the variation in device threshold voltage due to random dopant fluctuations in smaller silicon structures. Table 2.1 gives examples of process variations [11], where dd stands for die-to-die, wds for within-die systematic, and wdr for within-die random.

2.2.2 Sources of Increased Variations

The increase in variability is due to a variety of reasons [10–12]: First, nominal transistor geometries have become so small in deep sub-micron process technologies that even minute variations in physical features or dopants become pronounced and can cause significant impact on performance. At the same time, the wavelength of the light used in lithography to print the layout on silicon is not scaling at the same rate; for example, lights with 193nm and 157nm wavelengths are used to print 90nm and 45nm structures, respectively, on silicon [13]. With such a "thick pen", it is even harder to print fine and precise patterns on silicon, patterns that will continue to get only smaller relative to the wavelength of new light sources in the future [14]. This lithography limitation affects process parameters that are printed on silicon, such as channel length or wire width and spacing, to name a few. Fig. 2.1 shows that polysilicon corners drawn will not have the same sharp corners when they are created on silicon, causing the effective channel length to be different than what was intended. In addition, the non-uniformity in the surrounding area can cause patterns to be printed differently, as shown on the right side of Fig. 2.1. Variations in the transistor threshold voltage have also been increasing due to technology scaling. One important source of threshold voltage variation is random dopant fluctuation, which causes randomness in the location of dopants and does not allow a constant concentration of dopants in the channel, as seen in Fig. 2.2.

There are several Design for Manufacturability (DFM) techniques that try to reduce or even correct the effects of systematic variations; for example, techniques for resolution enhancement (RET) may include Optical Proximity Correction (OPC) [15] and phase shifting masks [16]. The idea behind OPC is to alter the layout by adding and adjusting shapes so that the printed feature on silicon looks more like the drawn feature. Other DFM techniques work by increasing the number of design rules, such as by increasing the spacing requirements between adjacent printed features or allowing only certain types of patterns to be used. While these DFM techniques have helped reduce some systematic effects that are well understood, the random effects especially random dopants fluctuations, and the other systematic effects that are difficult to understand and model have lead to an increase in variations, and will continue to do so in the future.

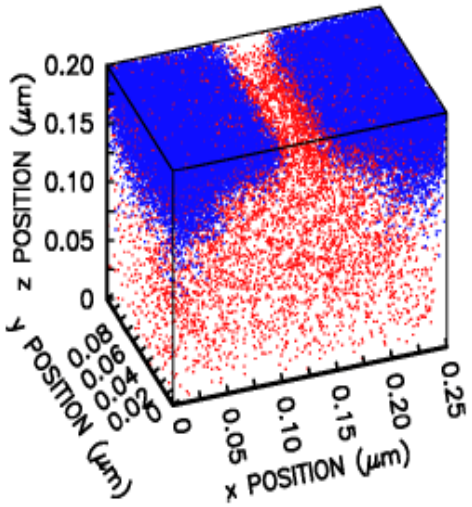


Figure 2.2: Uncertainty in the dopant location [2]

2.3 Circuit Performance Analysis under Variability

Process and environmental variations can have a significant impact on circuit performance metrics, particularly circuit timing and leakage power. Fig. 2.3 shows trends observed and predictions forecasted by the International Technology Roadmap for Semiconductors (ITRS) [3]. These predictions show that variations are either expected to remain at current levels (e.g. supply voltage and channel length variations), or to keep on increasing (e.g. threshold voltage variations due to random dopant fluctuations). Fig. 2.3 also shows that the impact of process and environmental variations on performance will reach alarming levels in the future, with circuit timing variability reaching 63% and leakage variability reaching 3× of their respective nominal values. It is therefore of crucial importance to devise techniques that are able to analyze the impact of variability on performance, and drive variation-aware circuit design and optimization. In the next sections, we cover different methods used to verify circuit timing under variability, and describe the dominant leakage current mechanisms that are impacted by variations.

Year of Production	2009	2010	2011	2012	2013	2014
% Vdd variability seen in on-chip circuits	10%	10%	10%	10%	10%	10%
% Vth variability for minimum size devices (doping only)	40%	40%	40%	58%	58%	81%
% Vth variability for minimum size devices (all sources)	42%	42%	42%	58%	58%	81%
% Vth variability for typical size devices (all sources)	20%	20%	20%	26%	26%	36%
% CD variability	12%	12%	12%	12%	12%	12%
% Circuit timing variability	49%	51%	60%	63%	63%	63%
% Circuit leakage power variability	186%	229%	255%	281%	287%	294%

Figure 2.3: Variability predictions and effects on performance [3]

2.3.1 Static Timing Analysis

Traditionally, Static Timing Analysis (STA), evolving from the early work of Kirkpatrick [17], has been used to verify that circuit timing meets the target constraints. We will not go into details about how STA works, but will mention only a few references [17–19] that the reader can explore to learn more about STA. Suffice it to say that the approach works by propagating signal arrival times in the circuit timing graph by adding cell and wire delays, and computes the maximum (late mode) and minimum (early mode) path delays. A simple comparison of these delays to the Setup and Hold timing constraints, that bound the minimum and maximum path delay respectively, allows one to determine if the circuit passes or fails timing.

2.3.2 Corner Analysis and Timing Margins

STA has been traditionally used to verify timing under variability. ASICs are typically designed by making sure the chip passes the timing requirements at all *process corners*, including nominal, worst, and best cases of device behavior. Such approach to

verification is known as corner analysis or worst-case files. A circuit is deemed to have *passed* the timing check if it meets the performance constraints for all "worst-case" files belonging to that process. In [20], corner-case files are generated in the following way. First, I-V curves are extracted from extensive measurements with SPICE, then parameters are deduced from measured data. Principal Component Analysis (PCA) is then applied to decorrelate SPICE parameters, and produce independent random variables. This leads to corner files at the transistor level. In [21], finding process corners is transformed into an optimization problem. Performance $g(X)$ (delay, for example) is assumed to be quadratic in terms of the process parameters vector $X = [X_1, \ldots, X_n]$:

$$g(X) = a + bX + X^T BX \tag{2.1}$$

where X_1, ..., X_n are either independent parameter variations, or their principal components. To find corners, the vector X that minimizes and maximizes $g(X)$ subject to performance constraints is found.

Corner case analysis suffers however from a number of limitations. One limitation is the fact that there are too many corners to verify, especially with the increase in the number of varying parameters; if n parameters are varying, 2^n verifications are needed, as one needs to check the two extremes of every parameter. With the ever increasing impact of within-die variations, predicted to grow from 35% to 60% of the total variation in channel length from 0.13μm to 0.07μm technologies, it is not clear how corner case analysis can handle the possible mismatch between devices at different locations. One can divide the die into smaller regions and apply region-based cornering, but this will only increase the number of corners even more! The bottom line is that corner case analysis cannot handle within-die variations.

For microprocessors, it is typical to check circuit timing with *nominal* transistor files, and to specify some *timing margin* (possibly zero) that should be left as slack to account for process variations. Frequency binning is a standard practice in microprocessors, however this is not to say that process variations are not a problem. In fact, within-die variations affect clock skew and cause significant timing failures if they are not accounted for.

There has been considerable discussion in the literature that the traditional methods

of using process corners or timing margins are breaking down. For one thing, we mentioned that the number of corners is increasing, making it very expensive to explore all corners. Also, the corner based method can be too conservative in some cases and does not provide the user with any quantitative feedback on the robustness of the design [20]; it is a pass/fail approach. Furthermore, this traditional approach cannot handle *within-die* statistical variations [22]. On the other hand, for microprocessors, where nominal process files are used and a timing margin needs to be left as slack, there is no easy way to decide what the margin should be, to account for within-die variations that have become more important in recent years [23].

2.3.3 Monte Carlo Analysis

The impact of process variations on circuit performance can also be predicted using Monte Carlo analysis [24,25]. This method is an iterative process where every iteration consists of two steps, namely *sampling* and *simulation*. In the sampling step, process parameters are sampled according to their distributions, thus generating a sampled value for every parameter. In the simulation step, the circuit is simulated or analyzed to obtain its performance at those particular process parameter settings obtained from the sampling step. After repeating this step multiple times, different performance samples are obtained, from which the distribution of circuit performance is computed. Monte Carlo analysis is very accurate in predicting the performance distribution of integrated circuits, however, it requires a large number of samples to converge, on the order of $10,000$. Given that a simulation step is required for every sample, this approach is computationally expensive, and is only practical for very small circuits.

2.3.4 Statistical Static Timing Analysis

Recently, due to the increased importance of within-die variations, there has been an increased interest in employing statistical techniques as part of the static timing analysis step. Statistical static timing analysis (SSTA), as it has been proposed, deals with circuit timing uncertainty by presenting an alternative to corner analysis and worst-case files. The basic idea is to start by modeling process parameters as random

variables with known distributions. Then, by using a variational timing model for cell and wire delay, these SSTA techniques propagate delay *distributions* instead of *deterministic* delay values in the timing graph. In this way, distributions for path delays and arrival times are determined, and ultimately the maximum circuit delay distribution, which is directly translated into a timing yield, is also determined. Unlike Monte Carlo analysis, which requires iteratively analyzing the circuit a large number of times to determine its delay distribution, SSTA techniques can do so in only one timing run and are thus applicable to large circuits.

Issues to consider in SSTA

There are many considerations that need to be addressed by almost any proposed SSTA technique, and these are summarized below:

- Choice to model one or all components of variations, including die-to-die, within-die systematic, and within-die random components.
- Choice of parameter distribution: SSTA models process parameters as random variables with specified distributions. The most popular distribution used by many techniques is the Normal or Gaussian distribution. However, some parameters are not normally distributed, or it may be difficult to determine their distribution at an early stage in the design flow. The most general SSTA technique must be able to handle arbitrary parameter distributions.
- Choice of variational timing model: A timing model is needed to capture the dependence of cell and wire delay variations on the underlying process and environmental parameters. The simple and most popular model is the first-order linear model, which treats delay variation as a linear (affine) function of PVT parameters. More complex models can be used, including nonlinear and quadratic models, which may add complexity to the SSTA techniques.
- Handling within-die spatial correlation: As mentioned earlier, within-die variations have a spatially correlated component expressing the fact that features that are close to each other are more likely to vary in the same way than features that

are further apart. Some SSTA techniques used correlation models, others used bounding schemes or even ignored spatial correlation altogether.

- Handling environmental variations: While process parameters can be modeled as *random variables* with distributions over the set of manufactured dies, environmental parameters, such as supply voltage and temperature, are not random in nature, and should be modeled as *uncertain variables*. In chapters 4 and 5, we present novel techniques that can handle this type of uncertain non-random variations.

In the past few years, a large body of literature was published on SSTA, each covering in its own merit one or more of the above considerations. We review below some of the earlier work that has been proposed, and later cover more recent work in the background sections of chapters 3 through 6. Broadly speaking, SSTA techniques fall into two groups, block-based and path-based techniques.

Block-Based Approaches

In block-based approaches, distributions of signal *arrival times* are propagated in the circuit timing graph to get the block delay distribution. Individual path delay distributions are only available indirectly, hence the name block-based.

Block-based techniques are linear in the size of the timing graph. Just as STA, their runtime is $O(V+E)$, where V and E are the number of vertices and edges in the timing graph respectively. They are also amenable to incremental analysis, since any change in arrival time distribution requires only propagating this change onwards. This property makes block-based approaches suitable for the inner loop of optimization tools. On the other hand, block-based approaches present some drawbacks, as they are not as accurate as path-based methods and they constitute a poor platform for capturing and propagating topological correlations.

In [26], random arrival times are captured via their Cumulative Distribution Functions (cdf) and random gate delays are captured via their Probability Density Functions (pdf). These are convolved to get the cdf of the arrival time when moving from an edge to a node in the timing graphs. Different arrival times are then Max-ed when they

converge to the same node to get the resulting cdf. Systematic within-die correlation is not taken into account, however correlation due to path sharing or reconvergent fanout is handled by propagating a dependency list that consists of all previous nodes that the current node depend on. Results show that ignoring path sharing is a conservative approach however the error is small and is not worth the overhead of creating dependency lists.

In [27] and [28], two similar models for timing quantities (arrival times and gate delays) are proposed, where the variations are considered Gaussian. They both handle within-die correlation and dependence on global sources of variations using the following decomposition.

$$A = a_0 + \sum_{i=1}^{n} a_i \Delta X + a_{n+1} \Delta R \tag{2.2}$$

where A is a random timing quantity with mean a_0. The first summation can either be interpreted as PCA decomposition of spatial correlation [27], or dependence on global sources of variations ΔX [28], and the last part accounts for independent random variations ΔR. Both works use analytical expressions of the Max function and propagate correlations by forcing the expression of the Max to be in the above form. This causes some errors to be induced, as the Max of two Gaussian random variables is not Gaussian. Also, spatial correlation is taken care of in [27] by dividing the die into a grid. Gates lying in the same region of the grid will have totally correlated within-die variations. It is not clear how the size of the grid is obtained, and how coarse or fine it should be.

In [29, 30], a conservative bound is obtained on the arrival time of all nodes if path reconvergence is ignored. However they present a way to handle path sharing similar to the dependency list of [26]. Arrival times are statistically "added" using convolution, and max-ed by simple multiplication of CDFs, in an implicit assumption of their independence. Finally, a method called quad-tree partitioning is proposed. It is based on assuming layers of correlations. The top layer is the global (die-to-die) and affects the whole timing quantities. As we go deeper, the partitioning increases four-fold, and becomes finer, allowing local correlation. It is not clear what number of layers to adopt, and how layout information would be extracted.

Path-Based Approaches

In path-based approaches, path delay distributions are expressed as functions of the underlying sources of variation. Gate delay distributions are added along a path to get its distribution. Once this is done for *all* paths in the block, circuit delay distribution is obtained from the joint probability of path delays by max-ing all paths. The general flow in path-based approaches is the following:

- Enumerate all critical paths
- Estimate path delay distributions
- Use multi-dimensional integration to combine all paths
- Estimate timing yield

Path-based approaches are more accurate than their block-based counterparts. Their analysis is more intuitive as they work on individual paths and they are in a better position to handle correlations. However, they suffer from a number of disadvantages. They are slower than block-based techniques, and face the path explosion problem since a large (exponential) number of paths needs to be considered as nominally non-critical paths may become critical under variability.

In [22], critical paths are enumerated using STA. The resulting timing report is augmented by adding path delay sensitivities to the various sources of variation. These sensitivities are obtained from wire and gate sensitivities by assuming a linear delay model and running SPICE simulations. A conservative yield is estimated by simply multiplying the yield (distribution) of every path, hence assuming independence of path delay variations.

In [31], the slack (difference between required time and arrival time) distribution is expressed as a function of the underlying global sources of variations. The timing yield is depicted as a multi-dimensional integration of the joint pdf of sources of variation over a *feasible region* defined by the performance constraints. Two integration algorithms are proposed. One fits a parallelepiped in the feasible region, and the other fits an ellipsoid. Both path sharing and dependence on global sources of variations, using decomposition, are handled.

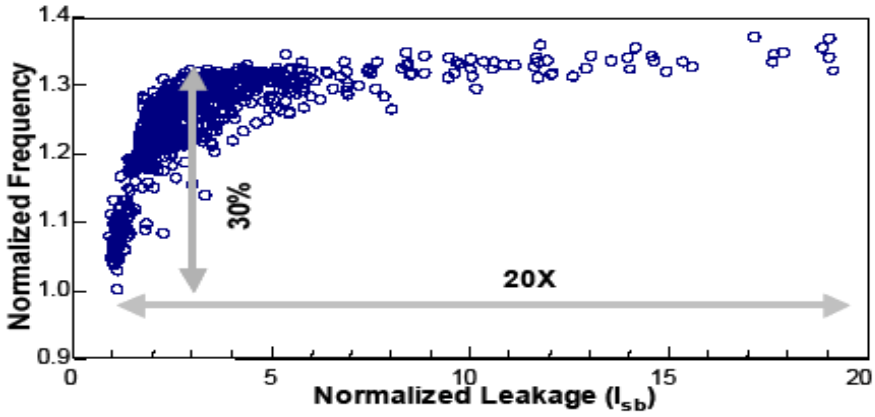


Figure 2.4: Impact of variations on leakage and frequency

In [32], bounds on circuit distribution are obtained using minimum and maximum covariance, and then these bounds are tightened using a technique based on stochastic majorization. Arbitrary gate (node) distributions are used, however, path distributions are assumed to be Gaussian due to the central limit theorem stating that the sum of independent RVs converges to a Gaussian. Die-to-die and within-die variations are also handled using decomposition.

In [33, 34], the effect of die-to-die and within-die variations on the maximum frequency (FMAX) distribution is assessed. The concept of "generic critical paths" is introduced, and interesting conclusions were obtained from experimental results stating that within-die variations impact the mean of the FMAX distribution, while die-to-die variations impact its variance.

2.3.5 Dominant Leakage Mechanisms

We now switch gears to cover the major types of leakage currents that are impacted by process and environmental variations. As was seen in Fig. 2.3, leakage power will continue to increase with technology scaling, as threshold voltage and oxide thickness have been reduced to improve performance. Fig. 2.4 presents published data from Intel, showing that the spread in leakage current can reach up to 20×, suggesting that leakage power is highly sensitive to variations, and that this impact on leakage power must be analyzed correctly.

There are many transistor leakage current mechanisms, including subthreshold leakage, gate tunneling leakage, edge-directed-tunneling (EDT), and reverse-bias p-n junction leakage, however, the dominant mechanisms in current CMOS technologies are due to subthreshold leakage and gate tunneling leakage [35]. Subthreshold leakage is the current that occurs between the drain and source of a transistor when the gate voltage is less than the transistor threshold voltage, i.e., when the transistor is supposedly turned OFF. This type of leakage current is exponentially dependent on threshold voltage and also very sensitive to temperature [36]. Gate tunneling leakage on the other hand refers to the leakage current that is tunneled or sunk through the gate terminal of a transistor. Traditionally, the gate was seen as an ideal capacitor with no leakage through it, however, the reduction of gate oxide thickness due to technology scaling has increased the tunneling of electrons through the oxide, resulting in gate leakage. This type of leakage is sensitive to oxide thickness reduction and to increased voltage differential across the gate, and unlike subthreshold leakage, it occurs when the transistor is both ON and OFF [35].

Given the high impact of process and environmental variations on leakage, it is important to be able to quantify leakage variability accurately during the analysis step. In chapter 7, we will cover in detail the background and related work on statistical leakage estimation.

2.4 Summary

Continuous scaling trends in digital integrated circuits have lead to an increase in the magnitude of process and environmental variations. Variability greatly impacts circuit performance, including timing and leakage power, and causes these performance metrics to fail to meet their target budgets. This calls for variation-aware performance analysis and verification techniques, not only to quantify the impact of variations, but also to drive circuit design and optimization.

3 Pre-placement Statistical Static Timing Analysis

3.1 Introduction

The ever increasing variability in process parameters, giving rise to circuit delay variations, presents an important challenge to the prediction and verification of circuit timing. As mentioned in Chapter 2, one of the considerations that needs to be addressed by SSTA proposals is the way to handle within-die spatial correlation. Some recent approaches to statistical static timing analysis rely on the existence of within-die correlation models; such models however are not readily and realistically available from the process, at least not at an early stage of the design. In this chapter, we present an early statistical timing analysis technique that can operate pre-placement, when within-die correlations are still unknown. Starting from a simple delay model that requires minimal input from the user, we will predict bounds on the distribution of the maximum circuit delay. Such bounds are valid for any arbitrary within-die correlation. We will use these bounds to introduce the concept of *margin uncertainty* and predict a margin range that can help designers at an early stage of the design flow.

3.2 Background

Process variations impact circuit delay, and can consequently cause timing yield loss. In Chapter 2, we have seen that process variations have been traditionally taken care of in various ways. In microprocessors, it is typical to check circuit timing with *nominal* transistor files, and to specify some *timing margin* that should be left as slack between

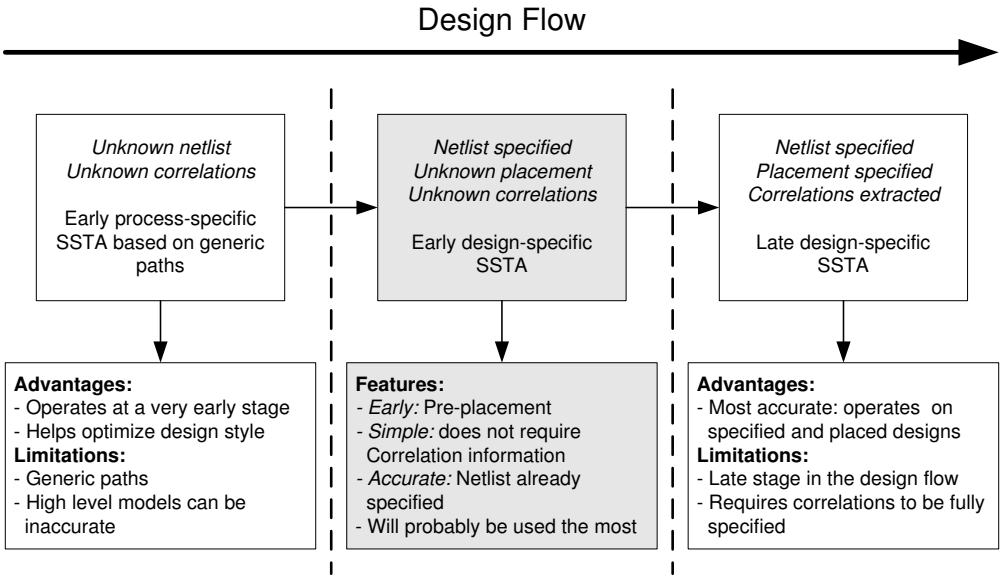


Figure 3.1: Types of SSTA

the nominal delays and the timing constraints, in order to account for process variations. In ASICs, the practice is to typically design circuits by making sure the chip passes the timing requirements at all *process corners*, including nominal, worst, and best cases of device behavior. If these settings are too pessimistic, then designers are forced to waste time and effort optimizing a circuit using design conditions that are too stringent. SSTA techniques offer a better alternative.

3.2.1 Types of SSTA

In Chapter 2, we have reviewed some related work on SSTA, focusing on the general outline of these approaches. In this chapter however, we describe how different SSTA techniques have handled within-die spatial correlations.

In earlier work [22,26,37,38], within-die variations were assumed to be totally uncorrelated. This assumption is generally not true as within-die variations have a spatially correlated component, however it is typically hard to express the correlations between within-die parameter variations with a model built from process data. Different attempts to model correlations have been proposed: In [27], spatial correlation is taken

care of by partitioning the die into a number of grids. Cells and wires lying in the same region of the grid will have highly or totally correlated within-die variations, and those lying in different grids will have low or zero correlation. It is not clear how the size of the grid is obtained, and how coarse or fine it should be. In addition, principal component analysis (PCA) is used to de-correlate variations onto a set of independent (uncorrelated) random variables. In [29], a similar partitioning scheme, known as quad-tree partitioning, is used to express a region-wise spatial correlation among within-die variations. This scheme is based on assuming layers of correlations. The top layer is the global (die-to-die) variation and affects all instances of cells and wires. As we go to the next layer, the partitioning increases four-fold, and becomes finer, allowing for local correlation. In [28], correlation is taken care of using a canonical model, where each variation is expressed in terms of global sources of variation obtained using PCA.

All these techniques rely on the existence of process data to build the correlation models, as well as placement information to determine the location of cells and wires in the grid. This information may not be readily available, at least not at an early stage of the design flow, therefore, these types of post-placement SSTA become final sign-off tools and cannot be used during early circuit design. In fact, we believe that there are three types of SSTA techniques that may be useful in practice, and these different types are listed in Fig. 3.1:

1. early process-specific SSTA based on generic paths. This can be applied early in the design flow, to establish timing margins for generic paths in the candidate technology, even before circuit design has started, and to possibly optimize the devices or the circuit style to reduce these margins. This is the type of approach adopted by [33, 34, 39, 40].

2. early design-specific SSTA based on a given design in a given process. This can be applied pre-placement, during the circuit design stage. This would be perhaps the most heavily used type of SSTA, based on present usage patterns of existing STA tools.

3. late design-specific SSTA, used post-placement for final sign-off.

Of course, the level of accuracy achieved and the level of physical detail that is taken into account will vary among the different types of SSTA. For instance, wire parasitics variations have an effect on delay (depending on the strengths of the driver, etc.), but can only really be taken into account post-placement. In a pre-placement scenario, they can often be ignored, or replaced by some safety factor.

Fig. 3.1 also highlights the advantages and limitations of each SSTA type in terms of chronology (early/late, pre-/post-placement) and accuracy. So far, only early process-specific (first type) and late post-placement design-specific (third type) SSTA techniques have been tackled in the literature. On one hand, the problem with the former type is the concept of the generic path, which might not capture a specific design very well; on the other hand, the problem with the latter SSTA type is its reliance on extensive correlation and placement information, which may not be available. In this chapter, we try to combine the best of both worlds by presenting the first early pre-placement SSTA. As defined in Fig. 3.1, our technique is *early*, *simple*, and *accurate* since it operates pre-placement on a specific netlist without requiring any correlation information. This does not mean, however, that we ignore correlations; on the contrary, we handle the lack of correlation information using bounds that are valid for any arbitrary correlation or placement.

3.2.2 Yield Specific Margins

As was mentioned earlier, the goal of statistical timing analysis is to verify timing under process variations, and not to simply predict the distribution of circuit delay. The real question is *how much margin* should one leave on top of nominal maximum delay to guarantee a desired timing yield. SSTA answers this question by predicting the distribution of circuit delay. Assume that a target yield of 99% is desired; using SSTA, the 99^{th} delay percentile is predicted, from which the 99% yield margin is deduced by simply subtracting the nominal circuit delay. The following equation shows how to determine the timing margin for a yield of α:

$$\tau_\alpha = D_\alpha - D_{\text{nom}} \tag{3.1}$$

where D_{nom} is the nominal maximum circuit delay, D_α is the α-percentile of delay, and τ_α is the timing margin specific to yield α. Once this *yield specific margin* is determined, timing is verified by simply "reducing" the nominal circuit delay by this much margin.

3.3 Overview

In late post-placement SSTA, the design has already been placed and correlations have been extracted; therefore, a *unique* margin is predicted to verify the timing of the design. However, our technique falls into early pre-placement SSTA where correlations are still unknown; this uncertainty in the correlations is translated into a *margin uncertainty* as our technique will predict a margin *range* to cover all possible correlation settings. Fig. 3.4 gives a "sneak preview" of the big picture, where a *min margin* (predicted from the delay distribution upper bound under best correlation setting), a *max margin* (predicted from the delay distribution lower bound under worst correlation setting), and a *margin uncertainty* (to account for correlations) are presented to designers at an early stage of the design flow; these margins can help them take early design decisions without having to wait until the design is placed. We also show that the margins predicted by post-placement SSTA for a specific correlation and placement fall within our margin range.

At this point, the reader is cautioned not to confuse the "min margin" that we propose with the margin that is needed to cover the variation in the *minimum* circuit delay (or hold check margin). In fact, both the min and max margins reported in this chapter are used to cover variations in the *maximum* circuit delay; it is the within-die correlation, being at its best or worst setting, that determines whether the margin is a *min* or *max* margin respectively. Our analysis can be easily extended to find a similar margin range to cover variations in the minimum circuit delay.

Starting from a simple parameter model broken down into die-to-die and within-die components, and assuming a linear delay model based on sensitivities to process parameters, we will construct a standard delay model and propagate it in the timing graph similarly to what was proposed in the literature [27, 28]. To account for the

unknown within-die correlations however, we assess the effect of correlation on the sum and max of random variables to produce bounds on the distribution of circuit delay, and prove that these bounds are valid for arbitrary correlations. Using these bounds, we present the concept of *margin uncertainty* that can help designers verify timing at an early stage of the design flow. We believe this approach would be a good addition to, and not a replacement for, current post-placement SSTA techniques; our margin range can guide designers during early optimization, while the final margin predicted by post-placement SSTA can be used for final sign-off to achieve timing closure.

3.4 Modeling Variations

In this section, we start by modeling variations at the level of process parameters. Then, using a first-order linear delay model, we express timing quantities such as gate delays and arrival times as a function of the underlying process variations.

3.4.1 Parameter Model

For a given circuit element or layout feature i, let $X_j(i)$, be a zero-mean Gaussian *random variable* (RV) that denotes the variation of a certain parameter j of this element from its nominal (mean) value. Thus, for example, $X_j(i)$ may represent channel length variations of transistor i. Notice, the Gaussian assumption is very common in the literature [27, 28, 31, 39, 40]. It is also standard practice [41] to express parameter variation by breaking it up into *die-to-die* and *within-die* components, as follows:

$$X_j(i) = X_{dd,j} + X_{wd,j}(i) \tag{3.2}$$

The die-to-die component $X_{dd,j}$ is an *independent*[1] zero-mean Gaussian RV that is *global* to the die as it takes the same value for all instances of this element on a given die, irrespective of location. The within-die component $X_{wd,j}(i)$ is a zero-mean Gaussian that can take different values for different instances of that element on the same die.

[1] Throughout this thesis, whenever an individual RV is described as "independent", this means that it is independent of all other RVs under consideration.

It is thus a *local* variation specific to every instance. Keep in mind that $X_{wd,j}(i)$ has some correlation due to systematic effects. We can rewrite (3.2) in the following way:

$$X_j(i) = \sigma_{dd,j}\, Z_{dd,j} + \sigma_{wd,j}\, Z_{wd,j}(i) \tag{3.3}$$

where $\sigma_{dd,j}$ and $\sigma_{wd,j}$ are the parameter's die-to-die and within-die standard deviations respectively, which can be obtained from the process for that specific parameter j. Note that $Z_{dd,j}$ and $Z_{wd,j}(i)$ are standard normal RVs with zero-mean and unit variance, and that $Z_{dd,j}$ is global whereas $Z_{wd,j}(i)$ is local with *some* correlation for different i's. For the scope of this chapter, the within-die correlation will be considered *unknown* or *unavailable*.

3.4.2 Gate Delay model

In general, there is a nonlinear relationship between gate delay and transistor parameters. Simple circuit simulations, however, reveal that this nonlinearity is not strong, especially for small transistor parameter variations. Therefore, we will simply assume that gate delay is linearly dependent on the process, and hence is Gaussian with mean equal to its nominal value. This assumption is also very common, and is used in all first-order path-based and block-based techniques [27, 31, 32, 37, 39].

Assume that p process parameters are varying; these can include channel length, threshold voltage, transistor width, and so on. For each gate, we can extract sensitivities to the different varying process parameters using circuit simulation; this can be done as part of library characterization. We can thus write the delay of gate i, $D(i)$, in the following way:

$$D(i) = \mu_i + \sum_{j=1}^{p} s_{ij} X_j(i) \tag{3.4}$$

where μ_i is the mean (nominal) delay, s_{ij} is the delay sensitivity of gate i to process parameter j, and $X_j(i)$ is the variation of process parameter j as defined in (3.3). Note that, for all the transistors within one logic gate i, we assume that the variations of process parameter j are captured with a *single* RV $X_j(i)$ and that $X_j(i)$'s are assumed to be independent for different j's, *i.e.*, variations of different process parameters are

independent. Replacing $X_j(i)$ with its value from (3.3) yields:

$$D(i) = \mu_i + \sum_{j=1}^{p} \alpha_{ij} Z_{dd,j} + \sum_{j=1}^{p} \beta_{ij} Z_{wd,j}(i) \tag{3.5}$$

where $\alpha_{ij} = s_{ij}\,\sigma_{dd,j}$ and $\beta_{ij} = s_{ij}\,\sigma_{wd,j}$. We can further group the within-die variations of different parameters into a single within-die *delay* component. This leads to the following expression:

$$D(i) = \mu_i + \sum_{j=1}^{p} \alpha_{ij} Z_{dd,j} + \beta_{wd,i} Z_{wd,i} \tag{3.6}$$

where $\beta_{wd,i} = \sqrt{\sum_{j=1}^{p} \beta_{ij}^2}$ since $Z_{wd,j}(i)$ are independent for different j. Note that $Z_{wd,i}$ is a standard normal RV that represents the within-die variation of delay $D(i)$. Also note that for different gates i and k, the within-die components of $D(i)$ and $D(k)$, *i.e.*, $Z_{wd,i}$ and $Z_{wd,k}$ will have some unknown correlation due to the correlation in process that is also considered unknown.

Generally, it is more accurate to specify a *timing arc delay* rather than a *gate delay*, since a gate can have different timing arcs, with different delays. Therefore, we will be expressing timing arc delays later on using the same delay model in (3.6).

3.4.3 Arrival Time model

Similarly, signal arrival times at the inputs and outputs of gates are modeled as normally distributed random variables using the same model for gate delays. Let A be a signal arrival time; then we can express A as follows:

$$A = a_o + \sum_{j=1}^{p} a_j Z_{dd,j} + a_{\mathrm{p+1}} Z_{wd,A} \tag{3.7}$$

where a_o is the mean of A, a_j's are the sensitivities to the (global) die-to-die components of the various process parameters, and $a_{\mathrm{p+1}}$ is the sensitivity to the (local) within-die component specific to A, *i.e.*, $Z_{wd,A}$. Note that similarly to gate delays, the within-

die components of different arrival times will have some unknown correlations that particularly depend on placement and circuit topology.

Notice that unlike gate delay $D(i)$ in (3.6), where μ_i, α_{ij}, and $\beta_{wd,i}$ are inputs determined by the user through characterization, arrival time's parameters a_o, a_j's, and $a_{\mathrm{p}+1}$ will be determined by our algorithm through propagation in the timing graph. From here onwards, we will refer to our delay model in (3.7) as the **standard delay model**; this model has a constant part equal to the mean delay, a die-to-die part based on a linear expansion over the global die-to-die components of process parameters, and a within-die part that is local to the particular timing quantity in hand, which has some unknown correlation among different timing quantities.

3.5 Effect of Correlation

As mentioned earlier, our approach does not require correlation information and is therefore valid before circuit placement. Having said this, we will look into ways to assess the effect of unknown correlation on the two timing operations that are used during propagation, *i.e.*, the **sum** and **max** operations.

3.5.1 Effect on the Sum of Two RVs

Let X and Y be two normally distributed random variables with means μ_X and μ_Y respectively, standard deviations σ_X and σ_Y respectively, and correlation coefficient ρ. Let $Z = X + Y$. Then Z is also normally distributed, with mean μ and variance σ^2 given by:

$$\mu = \mu_X + \mu_Y \tag{3.8}$$

$$\sigma^2 = \sigma_X^2 + \sigma_Y^2 + 2\sigma_{XY} \tag{3.9}$$

$$= \sigma_X^2 + \sigma_Y^2 + 2\rho\sigma_X\sigma_Y \tag{3.10}$$

where $\sigma_{XY} = \rho\sigma_X\sigma_Y$ is the covariance of X and Y.

This shows that the variance σ^2 of the sum of two random variables is an increasing

function of their correlation coefficient ρ. In other words, if ρ is unknown, we are sure that the variance of the sum lies between a minimum achieved when $\rho = \rho_{\min} = 0$ (X and Y are uncorrelated), and a maximum achieved when $\rho = \rho_{\max} = 1$ (X and Y are totally correlated). Let $\sigma^2_{\min}$ and $\sigma^2_{\max}$ be these minimum and maximum variances, respectively. Then:

$$\sigma^2_{\min} = \sigma^2_X + \sigma^2_Y \tag{3.11}$$

$$\sigma^2_{\max} = (\sigma_X + \sigma_Y)^2 \tag{3.12}$$

Since Z is a normally distributed RV, then we can write its distribution using the cumulative distribution function (cdf) of the standard normal, $\Phi(\cdot)$:

$$\mathcal{P}\{Z \leq x\} = \Phi\left(\frac{x-\mu}{\sigma}\right) \tag{3.13}$$

Note that $\Phi(\cdot)$ is non-decreasing, therefore for $x - \mu \geq 0$, we can write the following bounds on the CDF of Z:

$$\Phi\left(\frac{x-\mu}{\sigma_{\max}}\right) \leq \Phi\left(\frac{x-\mu}{\sigma}\right) \leq \Phi\left(\frac{x-\mu}{\sigma_{\min}}\right) \tag{3.14}$$

An important result can be drawn from the above equation:

Result 1 *the cdf of the sum of two normal RVs with unknown correlation ρ (ranging from 0 to 1) can be bound by two extremes: setting $\rho = 0$ will lead to a minimum variance and thus an upper bound, while setting $\rho = 1$ will lead to a maximum variance and thus a lower bound on the distribution.*

Note that the bounds are valid only beyond the mean of the sum, *i.e.*, $x \geq \mu$, which is translated to be above the 50% line for normal distributions; this is obviously the more interesting yield range. This will be validated by our results.

Note that we have assumed correlation to be positive, i.e., falling in the $[0, 1]$ range, whereas in general, the full correlation range is $[-1, 1]$. Some comments are in order with respect to this positivity assumption. The assumption of positive correlation is practical for many sources of variability. A physical variation that slows down a tran-

sistor, a gate, or a path, is likely to have the same effect on another that lies nearby. If the other device/gate/path is far away, then it would probably be independent anyway. In our work, the RVs that will be assumed to be positively correlated are timing quantities (gate delays, arrival times, path delays). Thus, we are *not* assuming that all process variables are positively correlated. By saying that two gates (or two paths) have positively correlated delay, we mean that when process variations cause the delay of one of them to increase, then the delay of the other does not decrease. Notice, however, that both gate and path delays are typically functions of *several* process variables. Therefore, this assumption is not as strong as requiring that corresponding sensitivities to the same process parameters be always of the same sign. To be sure, most commonly used process parameters, such as L and V_{th} have similar effects on gate and path delay: for example, an increase in channel length will almost always slow down a logic gate; such process variables lead to positive correlation. However, the fact that gate and path delay variations are an *aggregate* effect of dependence on a number of process variables is the best justification for our assumption; even if one or two pathological process variables, in advanced technology let's say, are such that they cause opposite effects, they are unlikely to dominate to such an extent so as to lead to overall negative correlation. Finally, in the case of path-to-path correlation, the fact that many paths share common sub-paths would also contribute to positive overall correlation between them.contribute to positive overall correlation between them.

3.5.2 Effect on the Max of Two RVs

Similarly, let X and Y be two jointly normally distributed RVs, with unknown correlation coefficient ρ. Let $Z = \max(X, Y)$. We are interested in assessing the effect of ρ on the distribution of Z. For this purpose, let $F_\rho(\cdot)$ be the **joint** distribution of X and Y, for a particular ρ. We can write the cdf of the max Z in the following way:

$$\begin{aligned} \mathcal{P}\{Z \leq a\} &= \mathcal{P}\{\max(X, Y) \leq a\} && (3.15) \\ &= \mathcal{P}\{X \leq a\,,\, Y \leq a\} && (3.16) \\ &= F_\rho(a) && (3.17) \end{aligned}$$

Therefore, the cdf of Z is equal to $F_\rho(a)$, and is thus correlation dependent. Using Slepian's theorem [42], we know that the joint distribution of two normal RVs X and Y, and consequently the distribution of their max Z, is an increasing function of their correlation coefficient ρ. Therefore we can draw our second important result:

Result 2 *The distribution of the max of two jointly normal RVs with unknown correlation ρ (ranging from 0 to 1) can be bound by two extremes; a lower bound on the distribution is achieved by setting $\rho = 0$, and an upper bound on the distribution is achieved by setting $\rho = 1$.*

3.6 Timing Analysis Operations

In this section, we present our statistical timing analysis technique and explain how it operates on a given circuit. To do that, we will first define its operation on a single gate, and then show how to repeatedly apply it in a block-based fashion on the whole timing graph.

Fig. 3.2 shows a gate with two inputs A and B and output C. The arrival times at inputs A and B are assumed to be known from previous stages, and are given in the standard delay form presented in Section 3.4.3. Also assume that D_1 and D_2 are gate delay arcs for inputs A and B respectively, and are also given in the standard delay form. The arrival time at C will be equal to:

$$C = \max\left[\left(A + D_1\right), \left(B + D_2\right)\right] \tag{3.18}$$

Thus, in order to determine the arrival time at the output of any gate, we need to perform two basic timing operations: a **sum** operation performed on a gate delay arc and an input arrival time, and a **max** operation performed on the two timing quantities resulting from the *additions*.

Since the correlation between the within-die variations of different timing quantities is unknown, we will use the results from Section 3.5 to derive bounds on the distribution of C. Applying our approach on every gate, and propagating these bounds in the timing graph will lead to bounds on the maximum circuit delay.

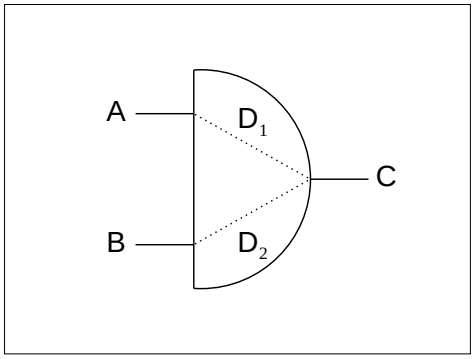


Figure 3.2: AND gate

3.6.1 Sum Operation

We will first show how to handle the sum operation given the lack of within-die correlation information. Assume that we have the gate in Fig. 3.2. Let $X = A + D_1$, and recall that both A and D_1 are expressed in the standard delay model:

$$A = a_o + \sum_{j=1}^{p} a_j Z_{dd,j} + a_{\mathrm{p+1}} Z_{wd,A} \tag{3.19}$$

$$D_1 = d_o + \sum_{j=1}^{p} d_j Z_{dd,j} + d_{\mathrm{p+1}} Z_{wd,D_1} \tag{3.20}$$

We are interested in writing X using the standard model for variation to be able to propagate it to later stages in the timing graph. Simple addition leads to,

$$X = (a_o + d_o) + \sum_{j=1}^{p} (a_j + d_j) Z_{dd,j} + [a_{\mathrm{p+1}} Z_{wd,A} + d_{\mathrm{p+1}} Z_{wd,D_1}] \tag{3.21}$$

The above equation is not given in the standard delay form presented in Section 3.4.3 since the expression in brackets is not yet resolved as required. To do that, let $W = a_{\mathrm{p+1}} Z_{wd,A} + d_{\mathrm{p+1}} Z_{wd,D_1}$, and recall that the correlation between $Z_{wd,A}$ and Z_{wd,D_1} is unknown. Using the result of Section 3.5.1, we want to bound the distribution of W and consequently the distribution of X. Let $Z_{wd,X_{lb}}$ be a standard normal RV. Then the variance of $(a_{\mathrm{p+1}}+d_{\mathrm{p+1}})Z_{wd,X_{lb}}$ is $(a_{\mathrm{p+1}}+d_{\mathrm{p+1}})^2$ and, therefore, the distribution

of $(a_{\mathrm{p}+1}+d_{\mathrm{p}+1})Z_{wd,X_{lb}}$ bounds the distribution of W from below via $\sigma^2_{\max}$ as in (3.12) and (3.14). Similarly, let $Z_{wd,X_{ub}}$ be a standard normal RV. Then the variance of $\sqrt{a^2_{\mathrm{p}+1}+d^2_{\mathrm{p}+1}}Z_{wd,X_{ub}}$ is $(a^2_{\mathrm{p}+1}+d^2_{\mathrm{p}+1})$ and, therefore, the distribution of $\sqrt{a^2_{\mathrm{p}+1}+d^2_{\mathrm{p}+1}}Z_{wd,X_{ub}}$ bounds the distribution of W from above via $\sigma^2_{\min}$ as in (3.11) and (3.14). Now that we have derived bounds on W, we use them to bound the distribution of X. This is done by replacing the term in brackets, *i.e.*, W, in (3.21) by the two bounds. This gives us the two bounds on X:

$$X_{lb} = (a_o + d_o) + \sum_{j=1}^{p}(a_j + d_j)Z_{dd,j} + (a_{\mathrm{p}+1} + d_{\mathrm{p}+1})Z_{wd,X_{lb}} \tag{3.22}$$

$$X_{ub} = (a_o + d_o) + \sum_{j=1}^{p}(a_j + d_j)Z_{dd,j} + \sqrt{a^2_{\mathrm{p}+1} + d^2_{\mathrm{p}+1}}Z_{wd,X_{ub}} \tag{3.23}$$

where X_{lb} and X_{ub} are two RVs whose distributions bound the distribution of X from below (lower-bound) and from above (upper-bound) respectively. Note that the within-die component of X_{lb} corresponds to the case where $Z_{wd,A}$ and Z_{wd,D_1} are assumed to be totally correlated ($\rho = 1$), which led to a maximum standard deviation of $(a_{\mathrm{p}+1}+d_{\mathrm{p}+1})$ as explained in Section 3.5.1. Similarly, the within-die component of X_{ub} corresponds to the case where $Z_{wd,A}$ and Z_{wd,D_1} are assumed to be uncorrelated ($\rho = 0$), which led to a minimum standard deviation of $\sqrt{a^2_{\mathrm{p}+1}+d^2_{\mathrm{p}+1}}$.

Note that the within-die components of X_{ub} and X_{lb}, *i.e.*, $Z_{wd,X_{ub}}$ and $Z_{wd,X_{lb}}$ have lost any dependence on $Z_{wd,A}$ and Z_{wd,D_1}. This is not a problem, since our approach does not keep track of within-die correlation, but always considers it unknown. Hence, the correlation between $Z_{wd,X_{ub}}$ (or $Z_{wd,X_{lb}}$) and any other within-die component of other arrival times will be considered unknown. This is important to account for any possible correlation.

At this point, we have handled the sum operation of a gate delay arc and an arrival time using bounds. For sake of clarity, we will refer to the process of generating a lower bound on the distribution of the sum, which was described above as **LB-sum**. Similarly, we will refer to the process of generating an upper bound on the distribution of the sum as **UB-sum**. The next section will show how we perform a max operation while keeping the standard delay model and handling unknown correlations.

Table 3.1: Within-die correlation settings for *sum* and *max*

	Lower Bound	**Upper Bound**
Sum	$\rho = 1$	$\rho = 0$
Max	$\rho = 0$	$\rho = 1$

3.6.2 Max Operation

It is well known that the max operator is not linear, which means that the maximum of two normally distributed RVs is not necessarily normal. Nevertheless, analytical expressions for the mean and variance of the maximum of two normally distributed RVs were determined by Clark in [43]. These expressions were presented and used in [27, 28], and we will review them next. Let $Z = \max(X, Y)$, then:

$$\mu_z = \mu_x T + \mu_y(1-T) + \theta\phi\left(\frac{\mu_x - \mu_y}{\theta}\right) \tag{3.24}$$

$$\sigma_z^2 = (\mu_x^2 + \sigma_x^2)T + (\mu_y^2 + \sigma_y^2)(1-T) + (\mu_x + \mu_y)\theta\phi\left(\frac{\mu_x - \mu_y}{\theta}\right) - \mu_z^2 \tag{3.25}$$

where $\phi(\cdot)$ is the probability density function (pdf) of the standard normal, and θ and T are given by:

$$\theta = \sqrt{\sigma_x^2 + \sigma_y^2 - 2\rho\sigma_x\sigma_y} \tag{3.26}$$

$$T = \Phi\left(\frac{\mu_x - \mu_y}{\theta}\right) \tag{3.27}$$

The main idea is that, given the **means**, **variances**, and **correlation** coefficient of two normally distributed RVs X and Y, the *exact* mean and variance of the max Z can be determined. Once the mean and variance are obtained, the max distribution is "cast" into a normal distribution with the same mean and variance, thus preserving only the first two moments of the exact max distribution and ignoring the higher moments; this is known as Clark's approximation, and is the standard method used in first-order SSTA [27, 28] to resolve the max operator. Clark's approximation works well in practice, capturing the max distribution with small error as reported in the

literature on SSTA. In this section, we will use a similar approach and show how we handle the unknown within-die correlation.

Looking back at the gate in Fig. 3.2, assume that at this point we have performed the sum operation $X = A + D_1$ and $Y = B + D_2$, and that we need to *max* the timing quantities resulting from these two additions. For this purpose, let $C = \max(X, Y)$; recall that X and Y are already in the standard delay form:

$$X = x_o + \sum_{j=1}^{p} x_j Z_{dd,j} + x_{\mathrm{p+1}}\, Z_{wd,X} \tag{3.28}$$

$$Y = y_o + \sum_{j=1}^{p} y_j Z_{dd,j} + y_{\mathrm{p+1}}\, Z_{wd,Y} \tag{3.29}$$

where $Z_{wd,X}$ and $Z_{wd,Y}$ are the within-die components of X and Y with unknown correlation ρ. We will use the result of Section 3.5.2 to derive bounds on the distribution of $C = \max(X, Y)$. Let C_{ub} and C_{lb} be two RVs defined as follows:

$$C_{ub} = \max(X, Y) \mid \rho_{X,Y} = \rho_{\max} \tag{3.30}$$

$$C_{lb} = \max(X, Y) \mid \rho_{X,Y} = \rho_{\min} \tag{3.31}$$

where $\rho_{X,Y}$ is the correlation coefficient between X and Y, and $\rho_{\max}$ ($\rho_{\min}$) is the maximum (minimum) possible correlation achieved by $\rho_{X,Y}$. Using the result from Section 3.5.2, which states that the distribution of the max of two RVs is an increasing function of their correlation coefficient, it is easy to see that the distributions of C_{ub} and C_{lb}, as defined above, will bound the distribution of C from above and below, respectively.

To determine $\rho_{\max}$ and $\rho_{\min}$, we first express $\rho_{X,Y}$ in terms of the sensitivities of X and Y:

$$\rho_{X,Y} = \frac{\sum_{j=1}^{p} x_j y_j + \rho\, x_{\mathrm{p+1}} y_{\mathrm{p+1}}}{\sqrt{\sum_{j=1}^{p} x_j^2 + x_{\mathrm{p+1}}^2}\,\sqrt{\sum_{j=1}^{p} y_j^2 + y_{\mathrm{p+1}}^2}} \tag{3.32}$$

Note that $\rho_{X,Y}$ is also a function of the unknown within-die correlation ρ, and it can be shown that $\rho_{X,Y}$ is maximum when $\rho = 1$ and is minimum when $\rho = 0$. Therefore

we can rewrite (3.30) and (3.31) in the following way:

$$C_{ub} = \max(X, Y) \mid \rho = 1 \tag{3.33}$$

$$C_{lb} = \max(X, Y) \mid \rho = 0 \tag{3.34}$$

Now that we have defined C_{ub} and C_{lb}, we express these timing quantities in the standard delay model to allow further propagation in the timing graph. To do this, we first use Clark's expressions to determine the exact mean and variance of C_{ub} (with $\rho = 1$) and C_{lb} (with $\rho = 0$). Then, we use the approach of [27, 28] to expand C_{ub} and C_{lb} in the standard delay model. This is done by preserving the dependence on the *global* RVs, *i.e.*, $Z_{dd,j}$, and then computing the within-die part by matching the total variance to the exact variance determined from Clark's expressions. This approach has become the standard way to approximate the max operation while preserving the correlation due to the global die-to-die variables. In the following, we will refer to the process of generating an upper/lower bound on the distribution of the max, *i.e.*, C_{ub}/C_{lb} as **UB/LB-max**.

3.6.3 Combining Sum and Max

In sections 3.6.1 and 3.6.2, we presented ways to find upper and lower bounds on the distributions of the sum and max of two RVs that are represented in our standard delay form. Recall that we are interested in deriving bounds on the distribution of C, the arrival time at the output of a gate. We know that:

$$C = \max\left[(A + D_1), (B + D_2)\right] \tag{3.35}$$

so that C can be determined by a series of two additions, $A + D_1$ and $B + D_2$, and one max involving the results of these additions. Therefore, to determine a lower bound on the distribution of C, we need to perform the sum and the max in the *lower bound mode*, *i.e.*, using LB-sum and LB-max. Similarly, to determine an upper bound on the distribution of C, we need to perform our analysis in the *upper bound mode*, *i.e.*, using UB-sum and UB-max successively.

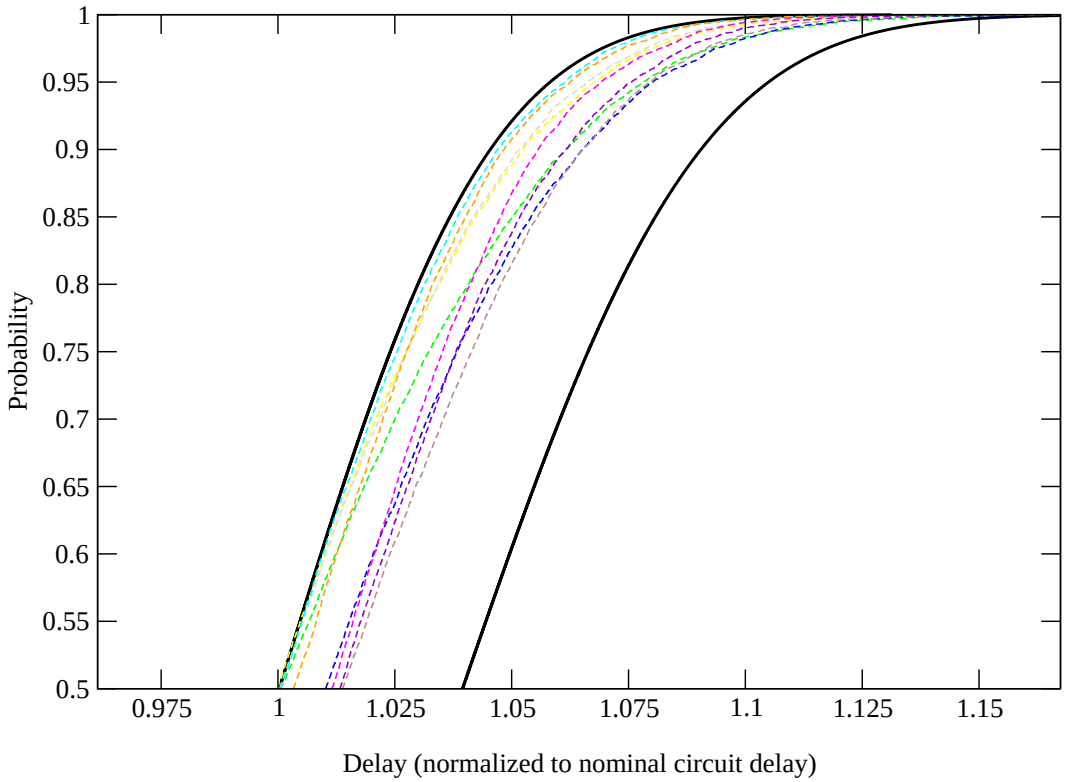


Figure 3.3: Upper/Lower bounds vs. MC distributions for circuit c499

Table 3.1 gives a summary of the different settings of the unknown within-die correlation ρ that should be used for the sum and max operations to get a lower/upper bound on the distribution of delay at the output node of a given gate.

3.6.4 Block-based Propagation

The rest of our approach is similar to deterministic STA, except: the deterministic sum is replaced by the UB and LB sums as described in Section 3.6.1 and the deterministic max is replaced by the UB and LB max as described in Section 3.6.2. In this way, and only in **a single pass**, we produce an upper and a lower bound on the distribution of delay at every node in the timing graph, particularly at the sink node (the circuit primary output). We are specifically interested in the delay of the sink node since it is equal to the maximum circuit delay.

Recall that the bounds produced are valid for any within-die correlation and consequently any circuit placement, since the analysis was performed under unknown correlation. This allows designers to be at an advantage, as one is able to predict, at an early stage of the design flow, the extent of circuit delay variations without having access to placement information.

3.7 Results

To test our SSTA technique on realistic circuits, we have implemented it as part of an existing static timing analyzer using C/C++. A 90nm CMOS library of gates was characterized using HSPICE in order to determine the sensitivities of gate delay to various process parameters. We have chosen to vary channel length (L_n and L_p), and threshold voltage (V_{tn} and V_{tp}) of n-mos and p-mos transistors using the ranges specified in the technology files, and have determined the sensitivities of gate delays to these process parameters. Also, in all experiments, we have assumed equal within-die and die-to-die process variation, *i.e.*, the within-die and the die-to-die variance of each process parameter is equal to 50% of the total parameter variance:

$$\sigma_{dd}^2 = \sigma_{wd}^2 = \frac{1}{2}\sigma^2 \tag{3.36}$$

This assumption is in line with [12], which predicts that within-die variation is between 40% to 65% of the total variation. We ran our SSTA analyzer on all of the ISCAS85 benchmark circuits [44], and computed bounds on the distribution of the maximum circuit delay. To validate these bounds, we performed several Monte Carlo (MC) tests under arbitrary within-die correlations, generated using the grid model [27] for spatial correlation. We varied the number of grids to increase/decrease the correlation, and also generated cases of *biased* correlations where we increased correlations across some paths (to maximize variance of delay) or decreased correlations across primary inputs (to increase the mean of delay).

The results for circuit c499 are shown in Fig. 3.3, where the solid curves represent the upper and lower bounds on the distribution of delay, and the dotted curves represent the distributions of delay generated through MC analysis under different arbitrary and

biased correlations. It is clear that all MC generated distributions fall between our bounds. Note that for higher percentiles, the bounds get tighter. Also note that, as stated at the end of Section 3.5.1, our bounds are only valid above the 50% line. This is not a problem, since we are mainly interested in high delay percentiles in order to design for a high yield.

As was mentioned in Section 3.2.2, the purpose of SSTA is to predict a *yield specific timing margin*, that is, the margin that should be left on top of the nominal maximum circuit delay to meet a certain target yield. Using the bounds on the circuit delay distribution, we can predict a min margin (from upper bound) and a max margin (from lower bound) by subtracting the nominal circuit delay from the desired delay percentiles predicted by the bounds as was explained in (3.1). Fig. 3.4 represents a plot to scale of these margins at a 99% target yield for all ISCAS circuits. It also shows the margin uncertainty caused by the unknown correlations that needed to be accounted for. Note that to guarantee the target yield, one needs to use the max margin since it is the one predicted from the lower bound on the distribution of delay, which accounts for "worst" possible correlations. The min margin is also useful to check if the design is feasible; recall that since it accounts for the "best" possible correlation, it is the smallest possible margin for the current design. If the min margin turned out to be unacceptable (too large) for designers, then the whole design needs to be adjusted. One conclusion to draw here is that, in the absence of any correlation information, and at a stage where the circuit has not yet been placed, it is a powerful asset to have an idea about the range of margins needed to achieve a desired yield. This allows designers to take early design decisions, and makes room for early optimization.

Table 3.2 lists, for all ISCAS circuits, the max margin as a percentage of nominal maximum circuit delay, at a 99% target yield. The largest max margin is about 14%. We have also listed, for sake of comparison, the margin predicted from "worst case" analysis on every gate. What we mean by "worst case" analysis is to set the within-die and die-to-die variations of every process parameter to their maximum (3σ) and perform deterministic STA. It is clear that the "worst case" margin is more pessimistic, which shows the need for statistical techniques.

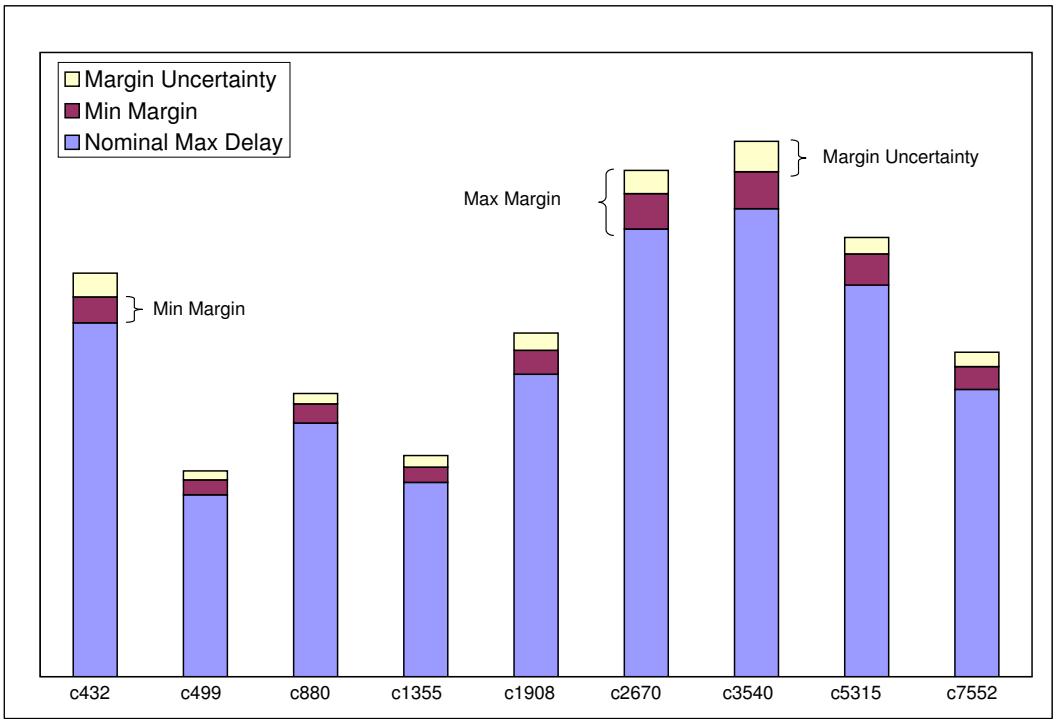


Figure 3.4: 99% yield margins

Table 3.2: Margin comparison as % of nominal max delay

ISCAS 85 **Circuit**	**Max Margin at** 99% **yield**	**"worst case" Margin**
c432	14.0%	24.0%
c499	13.2%	27.0%
c880	11.7%	24.4%
c1355	13.8%	24.2%
c1908	13.6%	25.4%
c2670	13.1%	26.6%
c3540	14.4%	26.4%
c5315	12.2%	26.1%
c7552	13.0%	25.3%

3.8 Incomplete correlation information

The analysis presented so far assumes *unknown* within-die correlations, and hence tries to account for this lack of information by selecting extreme correlation values for each timing operation as specified in Table 3.1. Under such a *blind* setting, *i.e.*, completely unknown correlations, the extremes are clearly $[0, 1]$. However, it is not unusual that designers have *some* raw information about within-die correlations, either through access to floorplanning information, or through pure historical observations; these cases will be referred to as "cases of incomplete correlation information". We are mainly interested in lowest ($\rho_{\min}$) and highest ($\rho_{\max}$) *observed* correlations, which, if available, can be safely used in lieu of the current pair of extremes $[0, 1]$, and the rest of the analysis remains the same.

3.9 Summary

In this chapter, we have presented a pre-placement statistical timing analysis technique that can operate under unknown within-die correlations. Starting from a simple delay model based on sensitivities to process parameters, we construct a standard delay model for arrival times, and propagate this model in the timing graph in a block-based fashion. To account for unknown correlations, we use bounds on each timing operation (sum and max) by taking the correlation coefficient to its extremes. We prove that these bounds are valid for any within-die correlation and placement, which makes our analysis valid pre-placement. We believe that this technique is useful since it can predict the range of margins needed to achieve a target yield at a very early stage of the design flow, prior to the access to correlation information.

4 General Framework for Parameterized Static Timing Analysis

4.1 Introduction

Many recent techniques for timing analysis under variability, in which delay is an *explicit* function of underlying parameters, may be described as *parameterized static timing analysis* techniques. The "*max*" operator, used repeatedly during block-based timing analysis, causes several complications during parameterized timing analysis. In this chapter, we introduce bounds on, and an approximation to, the *max* operator that allow us to develop a general, accurate, and efficient framework for parameterized timing, which can handle either uncertain or random variations, and a general class of nonlinear variational delay models. Applied to random variations, this approach is competitive with existing statistical static timing analysis techniques, in that it allows for nonlinear delay models and arbitrary parameter distributions. Applied to uncertain (non-random) variations, the method is competitive with existing multi-corner STA techniques, in that it more reliably reproduces overall circuit sensitivity to variations. Crucially, this technique can also be applied to the *mixed* case where both random and uncertain variations are considered.

4.2 Background

With the traditional approaches to timing verification becoming too expensive and unable to handle local variations, new alternatives have emerged in recent years, most of which can be described under the heading of *parameterized static timing analysis*.

Essentially, timing quantities are "parameterized" as explicit functions of the underlying process and environmental parameters, allowing one to assess the effect of these parameters on circuit delay, which can be useful in determining the robustness of the design and its sensitivity to variations. While manufacturing process variations can be modeled *statistically* as random variables (RVs), environmental variations such as supply voltage and temperature cannot. These are non-random in nature and must be treated as *uncertain* parameters. It is worth noting that both types of variations must be taken care of in the context of a general parameterized static timing analysis framework.

As mentioned in Chapter 2, one of the major trends in parameterized static timing analysis is block-based statistical static timing analysis (SSTA) [27, 28, 45–50], where parameters are modeled as RVs. In the past few years, several SSTA techniques have been proposed. In [27, 28], linear (first-order) delay models and Gaussian distributions for process parameters were used, which allows the use of tightness probability to resolve the *max* operation efficiently. It is expected, however, that nonlinearities will increase with technology scaling. On one hand, as the magnitude of process variations is increasing, first-order delay models may no longer be accurate. Also, there is evidence that delay is highly nonlinear in low-voltage modes. On the other hand, process variations are not necessarily Gaussian. For example gate length has an asymmetric non-Gaussian distribution due to variation in depth of focus (DOF).

To address these concerns, several attempts were made to generalize SSTA to handle nonlinear delay models and/or random variables with non-Gaussian distributions. In [46, 47], despite the use of quadratic models, Gaussian distributions were forced on process parameters. In [49], non-Gaussian process parameters were addressed, but first-order linear models were used. More recently, both non-Gaussian parameters and nonlinear models were addressed simultaneously, with varying degrees of success. In [48], a sampling (Monte Carlo) based regression is used to handle the *max* operation, which increases the overall runtime; [45] uses expensive multi-dimensional integration to handle the nonlinear non-Gaussian terms, which hinders the complexity of the approach; [50] proposes an efficient nonlinear non-Gaussian SSTA technique with linear complexity. This technique uses Fourier series and moment matching to approximate

the *max* operation efficiently. The drawback of this approach is that it requires subdividing the region of variation of every variation source, then for every sub-region, the Fourier transform of every possible distribution used is pre-computed. Although this is done as a pre-processing step, it can introduce undesired complications, especially if these distributions are unknown. In addition, by using moment matching and being distribution dependent, all these SSTA techniques fail to handle uncertain non-random parameters, which is a requirement in general parameterized static timing.

Another type of parameterized static timing analysis is linear-time multi-corner static timing analysis (STA), which was introduced in [51]. The idea is to propagate *hyperplanes*, i.e., affine linear functions of the process and environmental parameters, in the timing graph. The *max* operation is resolved by raising the hyperplanes in a way to always follow the maximum corner delay. In this way, the circuit maximum corner delay is estimated accurately. Although the approach can handle both random and uncertain parameters, it has some drawbacks: while trying to be always accurate at the maximum corner delay, the approach loses accuracy at the other corners. Consequently, the sensitivity to process variations is lost and this approach cannot be used for optimization because the true spread of the circuit delay is not accurately estimated. In addition, the analysis is restricted to linear delay models.

In this chapter, we propose an efficient and general parameterized static timing analysis framework that can handle nonlinear delay models and account for delay variability due to both random process parameters with arbitrary (and possibly unknown) distributions and uncertain non-random parameters (that typically depend on circuit operation). Our framework is based on a novel and efficient method of resolving the nonlinear *max* operator by either bounding or approximating it by a linear model, while preserving the inherent nonlinearity of the delay model itself. We have tested our technique within two timing verification frameworks, namely multi-corner static timing analysis and nonlinear non-Gaussian SSTA, and have shown that, at least when using quadratic delay models, the complexity of the approach is linear in both the number of process and environmental parameters and the circuit size. Our results show that the spread of the maximum circuit delay is accurately captured, whereby, on average, the maximum and minimum corner delays are predicted with less than 2% error for

multi-corner analysis. As for nonlinear non-Gaussian SSTA, all timing characteristics are predicted with less than 1% average error.

4.3 General Parameterized Timing

In general, the delay of gates and interconnects is a nonlinear function that can be approximated using a *delay model* that depends on the underlying process and environmental variations. As stated earlier, linear (first-order) and quadratic (second-order) delay models have been previously used in the literature. Nevertheless, for the time being, we will work under the assumption of an arbitrary delay model; we will show later that our approach can be used with a general class of nonlinear delay models. We can write the delay D of gates or interconnects as follows:

$$D = f_D(X_1, \cdots, X_p) \tag{4.1}$$

where $f_D(\cdot)$ is an arbitrary delay model, and X_i's represent process and environmental variations. Recall that not all variations can be modeled as random variables. While some parameters are indeed *random* such as channel length or threshold voltage, others are not statistical but rather *uncertain* parameters, such as temperature or supply voltage. These parameters typically depend on the operating environment. We will show that our analysis is indifferent to whether parameters are random or uncertain and thus the approach can be applied for both types of variations.

The benefit of parameterized timing analysis is in elevating the explicit delay dependence on process and environmental variations, i.e., the *delay model*, from the stage level to the circuit level, allowing one to assess the effect of these parameters on the total circuit delay. In block-based timing analysis, this is done by propagating timing quantities in the timing graph in topological order, using a sequence of basic operations, such as *add* operations on arrival times and arc delays, and *max* operations on the timing quantities resulting from those additions to get the output arrival time. If these basic operations do not distort the delay model at hand, then circuit delay can be represented in the same delay model. Unfortunately, it is known that the *max* is

nonlinear. The novelty of this work is in resolving the nonlinear *max* operator using a linear model, while preserving the inherent nonlinearity of the delay model.

In the following section, we propose two methods to efficiently resolve the *max*: first by using upper and lower bounds, and second by using an approximation that minimizes the square of the error. Our methods operate irrespective of the delay model used provided the model falls in a general class of nonlinear functions; they also handle random parameters with arbitrary distributions, as well as uncertain parameters.

4.4 Mathematical Framework

Let $\mathcal{F}$ be a general class of (possibly) nonlinear functions that obey the following three properties,

1. $\mathcal{F}$ is closed under linear (and/or affine) operations
2. $\forall f(\cdot) \in \mathcal{F}$, $f(\cdot)$ is bounded
3. $\forall f(\cdot) \in \mathcal{F}$, $f(\cdot)$ can be maximized and minimized efficiently

and assume that all timing quantities are represented using delay models that are in $\mathcal{F}$. For example, timing quantity A is modeled as follows:

$$A = f_A\left(X_1, \cdots, X_p\right) = f_A(X) \in \mathcal{F} \tag{4.2}$$

where $X = [X_1, \cdots, X_p]$ represent process/environmental parameters.

The first property essentially states that linear operations (such as addition or subtraction) will result in functions that belong to $\mathcal{F}$; in other words, $\mathcal{F}$ "survives" linear operations, that is, if $A = f_A(X) \in \mathcal{F}$ and $B = f_B(X) \in \mathcal{F}$, then $C = aA+bB+c$, where $[a, b, c] \in \mathcal{R}$, will be such that $C = f_C(X) \in \mathcal{F}$. Note that all polynomial models satisfy this property. The second and third properties imply that the delay model, and consequently any timing quantity expressed using the model, is bounded by a minimum and a maximum value over the space of parameters, i.e, $A_{\min} \leq A \leq A_{\max}$, where:

$$A_{\min} = \min_X f_A(X) \quad \text{and} \quad A_{\max} = \max_X f_A(X) \tag{4.3}$$

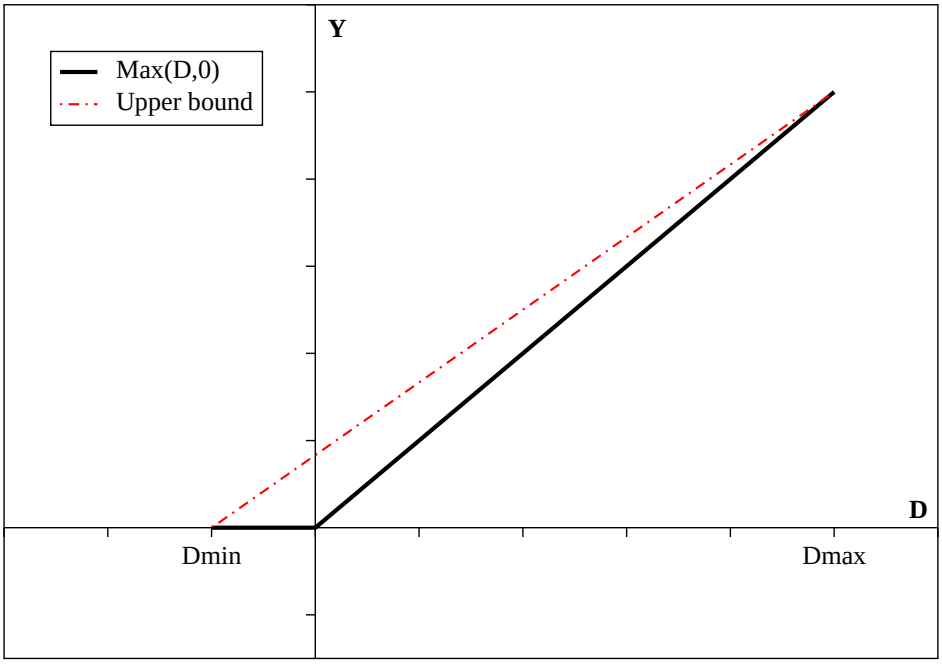


Figure 4.1: Upper bound on Y

Note that the second property is trivial if we recall that all physical process and environmental parameters are bounded, which implies that delays are also bounded. The third property is only forced to guarantee that our approach, which requires maximizing and minimizing the delay model to operate, is computationally efficient. For instance, if maximizing/minimizing the delay model is linear (in the process/environmental parameters), then the overall complexity of our approach is linear in the circuit size and process/environmental parameters. By property 1, it is clear that the *add* operation, being linear, will maintain their membership in $\mathcal{F}$ as delay models are propagated during timing analysis. However, the *max* operation, being nonlinear in general, is the crux of the problem, and we now focus on it.

4.4.1 Max Operation

Let A and B be two timing quantities expressed using delay models in $\mathcal{F}$. Let $C = \max(A, B)$ be the maximum of A and B. Since the maximum operator is, in general,

nonlinear, then C is not necessarily expressible using a delay model in $\mathcal{F}$. We are interested however in finding (1) two bounds on C, C_l and C_u, and (2) an approximation C_a, which can all be expressed using functions in $\mathcal{F}$, and such that:

$$C_l \leq C \leq C_u \tag{4.4}$$

$$C_a \approx C \tag{4.5}$$

where $C_l = f_{C_l}(X) \in \mathcal{F}$, $C_u = f_{C_u}(X) \in \mathcal{F}$, $C_a = f_{C_a}(X) \in \mathcal{F}$. Notice that:

$$C = \max(A, B) = B + \max([A - B], 0) \tag{4.6}$$

$$= B + \max(D, 0) = B + Y \tag{4.7}$$

where $D = A - B$ and $Y = \max(D, 0)$.

Recall from property 1 that $\mathcal{F}$ survives linear operations (including subtraction), which means that $D = f_D(X) \in \mathcal{F}$. By properties 2 and 3, D is bounded and varies between $[D_{\min}, D_{\max}]$ where $D_{\min} \leq D_{\max}$. Depending on the signs of these extreme values of D, we can identify two cases in which the *max* operator is either linear or nonlinear. If $D_{\min} \geq 0$ then $D \geq 0\ \forall X$, and $Y = D$. In this case, $C = A$ since A completely dominates B. The converse happens when $D_{\max} \leq 0$; in this case, $C = B$, since B completely dominates A. The more interesting case is when the *max* is nonlinear, which can be identified when $D_{\max} \geq 0$ and $D_{\min} \leq 0$. In this case, which we will refer to by saying that A and B are *co-dominant*, $Y = \max(D, 0)$ cannot be expressed using a delay model from $\mathcal{F}$. We will now show how we can bound and approximate Y using functions that belong to $\mathcal{F}$.

4.4.2 Bounding the Max

Upper bound

Fig. 4.1 shows a broken solid line representing a plot of $Y = \max(D, 0)$ between $D_{\min}$ and $D_{\max}$, the extreme values of D. We are interested in finding a linear function of D that is guaranteed to upper bound Y (recall, since $D \in \mathcal{F}$, then any linear function

of D is also a member of $\mathcal{F}$). The dashed line represents an affine function of D which upper bounds Y and is exact at $D_{\max}$ and $D_{\min}$. Note that the bound is closer to the exact *max* around $D_{\min}$ and $D_{\max}$ where either A or B dominates. The equation for Y_u, the upper bound on Y, can be expressed as follows:

$$Y_u = \frac{D_{\max}}{D_{\max} - D_{\min}}(D - D_{\min}) \tag{4.8}$$

By replacing Y with Y_u in (4.7), we get an upper bound C_u on C:

$$\begin{aligned} C_u = B + Y_u &= B + \frac{D_{\max}}{D_{\max} - D_{\min}}([A - B] - D_{\min}) \\ &= \left(\frac{D_{\max}}{D_{\max} - D_{\min}}\right) A - \left(\frac{D_{\min}}{D_{\max} - D_{\min}}\right) B - \frac{D_{\max} \cdot D_{\min}}{D_{\max} - D_{\min}} \end{aligned} \tag{4.9}$$

Note that $C_u = f_{C_u}(X) \in \mathcal{F}$ since it is a linear combination of A and B (property 1). To gain a more intuitive understanding of the above relationship, let us define the following terms:

- $S = D_{\max} - D_{\min}$ to be the "spread" of D
- $S_A = D_{\max}$ to be the "strength" of A, i.e. the region where A dominates B ($D \geq 0$)
- $S_B = -D_{\min} = |D_{\min}|$ to be the "strength" of B, i.e. the region where B dominates A ($D \leq 0$)
- $\alpha = \frac{S_A}{S}$ is the fraction of space where A dominates B
- $(1 - \alpha) = 1 - \frac{S_A}{S} = \frac{S_B}{S}$ is the fraction of space where B dominates A

Then, using the above notations, we can rewrite C_u as follows:

$$C_u = \alpha A + (1 - \alpha)B + \alpha(1 - \alpha) \cdot S \tag{4.10}$$

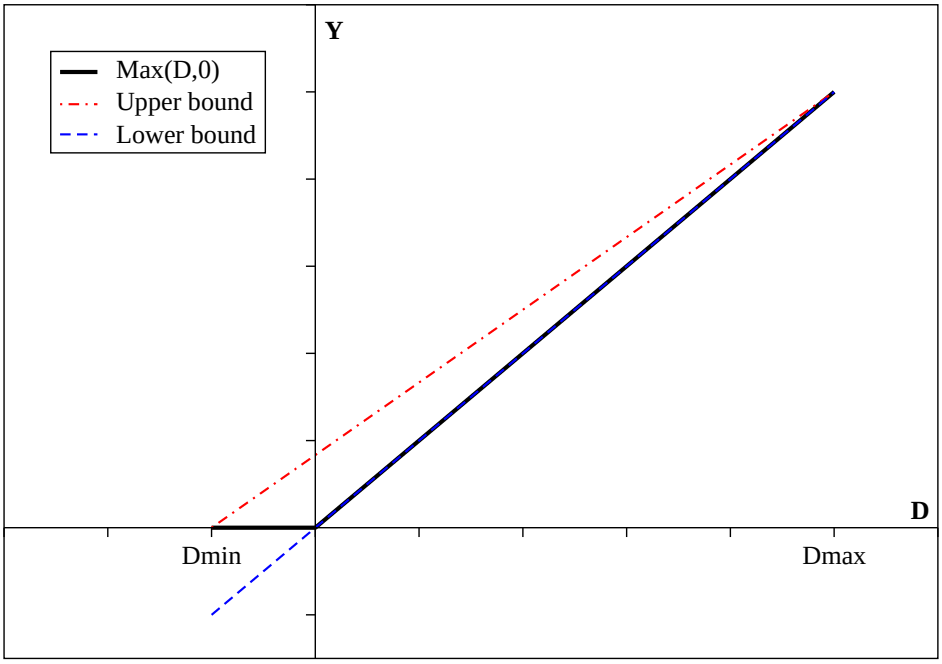


Figure 4.2: Lower bound $Y_l = D$

where A and B are both weighted by their "extent of dominance", so to speak, and the last term accounts for the region where both A and B are dominant, hence the product of α and $(1 - \alpha)$.

Lower bound

Similarly, we would like to find a lower bound on Y to find a lower bound $C = \max(A, B)$. Looking back at Fig. 4.1, it is easy to see that any function in the form $Y_l = aD$, where $0 \leq a \leq 1$, is a valid lower bound on $Y = \max(D, 0)$ and can be expressed using a function in $\mathcal{F}$ (property 1). In practice, we have found that limiting the choice of Y_l to one of three functions depending on the values of $D_{\min}$ and $D_{\max}$ is sufficient. Figures 4.2-4.4 depict these three cases. In addition to showing Y as a solid line and the upper bound Y_u as a dashed line, these figures show the lower bound Y_l to be one of the following:

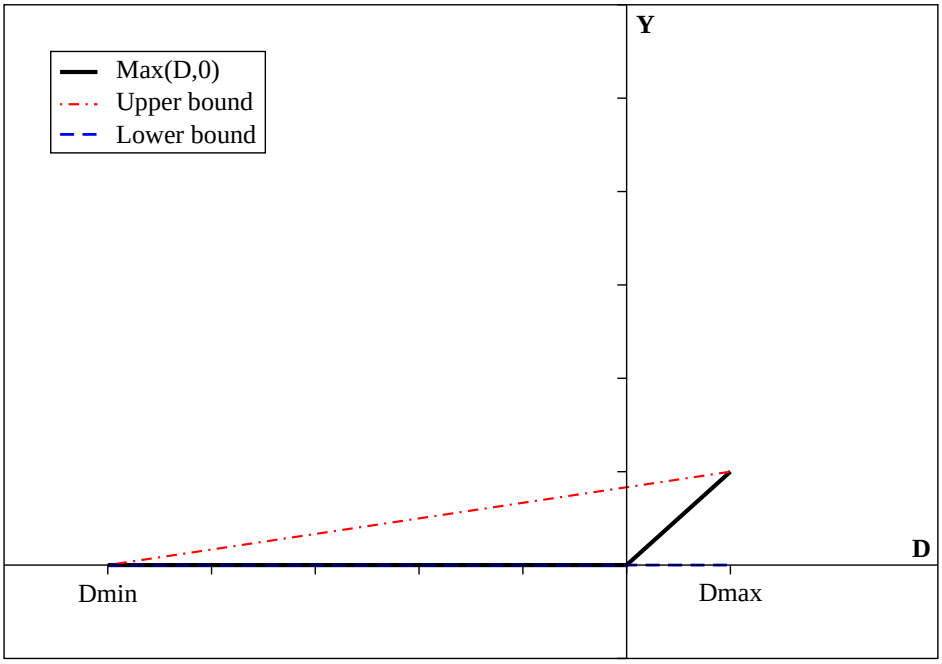


Figure 4.3: Lower bound $Y_l = 0$

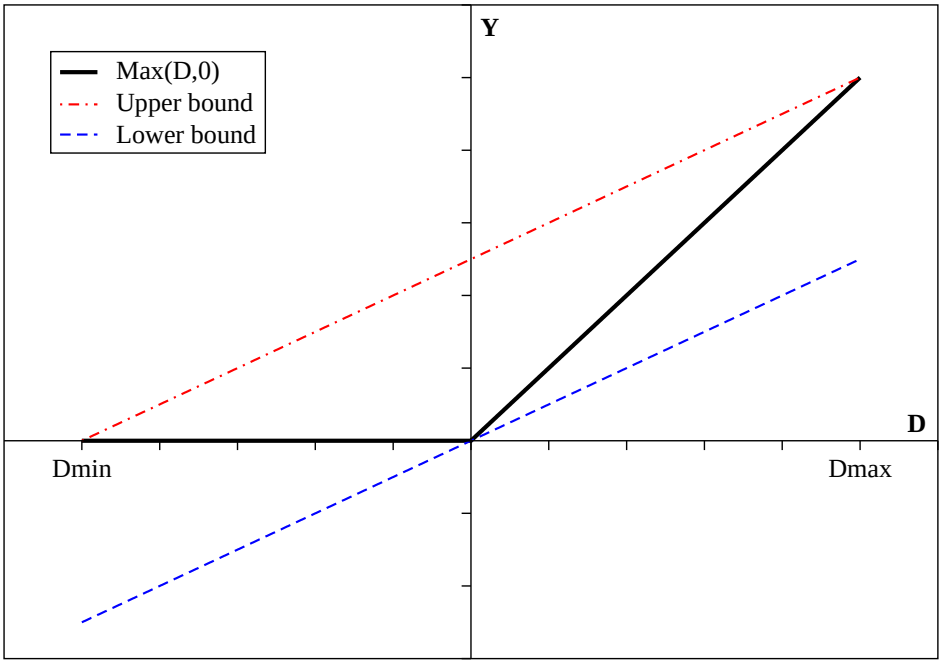


Figure 4.4: Lower bound on Y

$$Y_l = \begin{cases} D & \text{if } |D_{\max}| \gg |D_{\min}| \\ 0 & \text{if } |D_{\max}| \ll |D_{\min}| \\ \left(\frac{D_{\max}}{D_{\max} - D_{\min}}\right) D & \text{otherwise} \end{cases} \tag{4.11}$$

where the slope of Y_l in the third case is equal to that of the upper bound Y_u. Note that $\gg$ means "much larger than"; in our simulations, we have found that setting $\gg$ to be at least four times larger gives tighter lower bounds.

Replacing each case of the above in (4.7) gives us C_l, the lower bound on C:

$$C_l = \begin{cases} A & \text{if } |D_{\max}| \gg |D_{\min}| \\ B & \text{if } |D_{\max}| \ll |D_{\min}| \\ \alpha A + (1-\alpha) B & \text{otherwise} \end{cases} \tag{4.12}$$

where α is as defined in (4.10). Therefore, $C_l = f_{C_l}(X) \in \mathcal{F}$ since it is a linear combination of A and B (property 1).

4.4.3 Least Squares Max Approximation

In the previous sections, we have shown how we can determine upper and lower bounds on $C = \max(A, B)$ using a carefully chosen linear combination of A and B. We are now interested in finding an expression C_a that approximates $C = \max(A, B)$ and can be expressed using a function that belongs to $\mathcal{F}$. To do that, it suffices to approximate $Y = \max(D, 0)$ in (4.7) by a linear approximation Y_a in the form $Y_a = aD + b$. We will choose a and b in such a way to minimize the sum of the squares of the error inside the interval $[D_{\min}, D_{\max}]$; we call this approximation the *least squares max* approximation. It is understood that this approximation is applied only when the *max* is nonlinear, i.e., in the case of co-dominance.

Let E be the total error incurred by the above approximation. Then we can express E as follows:

$$E = \int_{D_{\min}}^{D_{\max}} \left(aD + b - \max(0, D)\right)^2 \, \mathrm{d}D \tag{4.13}$$

We now determine the parameters a and b that minimize E. To do that, we first determine the partial derivatives of E, $\frac{\partial E}{\partial a}$ and $\frac{\partial E}{\partial b}$, with respect to a and b, then we set the derivatives to zero and solve for a in b. This gives us the following two equations in a and b:

$$2\left(D_{\max}^3 - D_{\min}^3\right)a + 3\left(D_{\max}^2 - D_{\min}^2\right)b = 2D_{\max}^3 \tag{4.14}$$

$$\left(D_{\max}^2 - D_{\min}^2\right)a + 2\left(D_{\max} - D_{\min}\right)b = D_{\max}^2 \tag{4.15}$$

Finally, a and b can be easily obtained by solving the system of equations (4.14) and (4.15), which results in:

$$a = \frac{D_{\max}^2(D_{\max} - 3D_{\min})}{(D_{\max} - D_{\min})^3} \quad \text{and} \quad b = \frac{2D_{\max}^2 D_{\min}^2}{(D_{\max} - D_{\min})^3} \tag{4.16}$$

C_a can be easily obtained by replacing Y with $Y_a = aD + b$ in (4.7). It is easy to see that $C_a = f_{C_a}(X) \in \mathcal{F}$ since it is a linear combination of A and B which are in $\mathcal{F}$ (property 1). Fig. 4.5 shows the true *max*, $Y = \max(D, 0)$ in solid line, and the least squares *max* approximation, $Y_a = aD + b$, in dashed line.

To summarize, we have presented a mathematical framework that resolves the nonlinear *max* operator in two ways; either by using bounds, or by using an approximation that minimizes the error with the true *max*. This framework is simple and general, in the sense that it can be applied to a general class of nonlinear delay models, and it uses simple linear operations, allowing to keep and propagate the same delay model to later stages. In addition, the analysis is indifferent (1) to whether the variation sources are modeled as random variables or uncertain variables, or (2) to the types of distributions used, when the variations are random. In spite of its apparent simplicity, we will now demonstrate that this approach is extremely effective and competitive in dealing with two difficult application areas: multi-corner timing analysis as well as nonlinear non-Gaussian SSTA. Crucially, this approach can also deal with the "mixed" case, which is typical of practical situations, where some variables are random while others are simply uncertain.

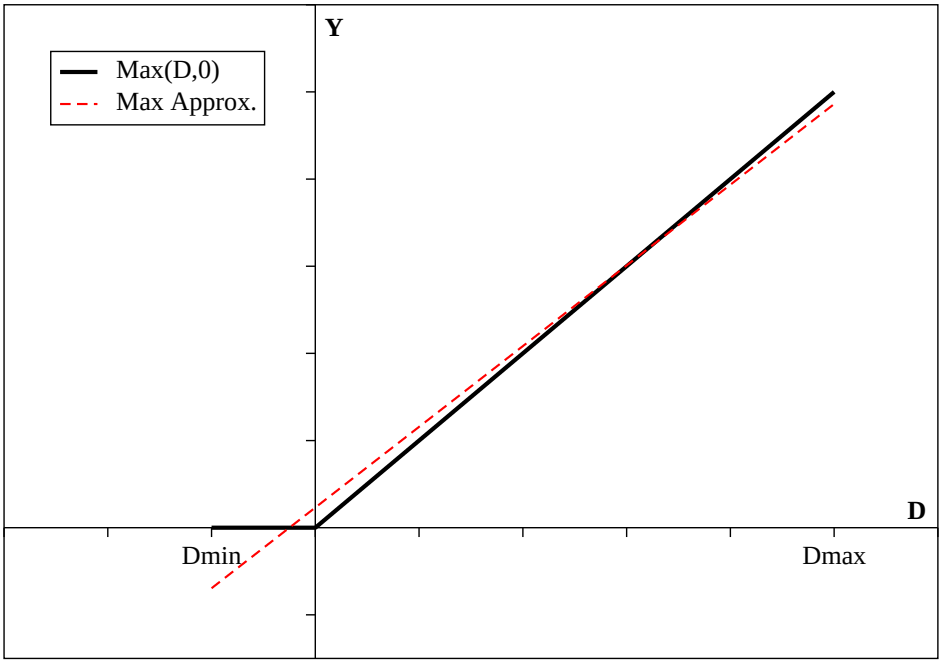


Figure 4.5: Least squares max approximation

4.5 Multi-corner STA

In corner case analysis, circuit timing must be checked at various process/environmental corners, which are typically extreme values of process/environmental parameters. This approach to timing verification is exponential in the number of varying parameters as multiple runs of STA are needed to cover all possible corners to determine the maximum and minimum corner delays. Instead, in practice, using parameterized timing, where timing quantities are expressed using a variational delay model that depends on process and environmental parameters, one hopes to do this task with only one traversal of the timing graph; this is what we call linear-time multi-corner STA.

In this section, we demonstrate how our simple framework can handle this otherwise complex analysis in a simple and elegant fashion. To our knowledge, and unlike [51], where the approach is restricted to linear delay models, we are the first to handle, in linear-time, multi-corner STA with linear and nonlinear delay models alike. For our framework to hold, all we need to show is that the three properties of Section 4.4 are

satisfied. For illustration, we will demonstrate our approach for linear and quadratic models.

4.5.1 Linear and Nonlinear Models

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be the sets of linear and quadratic delay models respectively. Let A be a timing quantity such that:

$$A = \begin{cases} a_o + \sum_{i=1}^{p} a_i X_i & \text{if } A \in \mathcal{F}_1 \\ a_o + \sum_{i=1}^{p} (a_i X_i + \hat{a}_i X_i^2) & \text{if } A \in \mathcal{F}_2 \end{cases} \tag{4.17}$$

where a_o is the nominal delay of A, and a_i and $\hat{a}_i$ are first-order and second-order sensitivities to X_i. Note that X_i's are arbitrary variation sources (random/uncertain variables) that are bounded, and without loss of generality, assume that $-1 \leq X_i \leq 1$.

It can be easily shown that property 1 holds, i.e., both $\mathcal{F}_1$ and $\mathcal{F}_2$ survive linear operations. In fact, if $[A, B] \in \mathcal{F}_1$, then $C = aA + bB + c$ is also $\in \mathcal{F}_1$. The same applies for $\mathcal{F}_2$. Property 2 also holds, since the X_i's are bounded. As for property 3, both the linear and the quadratic model can be easily maximized and minimized in $\mathcal{O}(p)$ time, where p is the number of process/environmental parameters; if $A \in \mathcal{F}_1$, then:

$$A_{\min} = a_o - \sum_{i=1}^{p} |a_i| \quad \text{and} \quad A_{\max} = a_o + \sum_{i=1}^{p} |a_i| \tag{4.18}$$

whereas, if $A \in \mathcal{F}_2$, then maximizing and minimizing A over the space of variations requires maximizing and minimizing the individual quadratic terms, $g(X_i) = a_i X_i + \hat{a}_i X_i^2$. Below, we show how this can be done analytically depending on whether $g(X_i)$ is monotone in $[-1, 1]$ or not.

The quadratic function is of the form, $g(x) = ax + bx^2$, where $-1 \leq x \leq 1$, and $[a, b] \in \mathcal{R}$. Recall that the minimum or maximum of a parabola occurs at the vertex $x_v = -a/(2b)$. If x_v falls outside $[-1, 1]$ then $g(x)$ is monotone in $[-1, 1]$, so that its maximum and minimum occur at the domain boundaries:

$$\begin{aligned} \max_x g(x) &= \max(g(-1), g(1)) = |a| + b \\ \min_x g(x) &= \min(g(-1), g(1)) = -|a| + b \end{aligned}$$

On the other hand, if x_v falls in $[-1, 1]$, then $g(x)$ is not monotone in $[-1, 1]$, i.e., the maximum and minimum can be either at the vertex or at the boundaries of the domain:

$$\begin{aligned} \max_x g(x) &= \max(g(-1), g(1), g(x_v)) = \max(|a| + b\ ,\ -a^2/4b) \\ \min_x g(x) &= \min(g(-1), g(1), g(x_v)) = \min(-|a| + b\ ,\ -a^2/4b) \end{aligned}$$

All these operations above can be done in constant time, therefore maximizing and minimization the timing quantity A is linear in the number of parameters, since there are p quadratic terms.

At this point, the reader is reminded that our approach is not restricted to the sets of linear and quadratic delay models, $\mathcal{F}_1$ and $\mathcal{F}_2$, but can also be used with more general nonlinear models. These models must (at the very least) meet properties 1 and 2 listed at the start of Section 4.4. However, property 3, stating that the model should be maximized and minimized "efficiently", is not a requirement and is only used to achieve good overall runtime. In the case of linear and quadratic models, property 3 can be achieved in linear complexity as shown previously. For higher-order models, it may be required to formulate a possibly complex optimization problem to maximize and minimize the model.

Given that all the required properties listed in Section 4.4 are satisfied by $\mathcal{F}_1$ and $\mathcal{F}_2$, the methods described in Sections 4.4.2 and 4.4.3 for resolving the *max* operation can be applied to demonstrate multi-corner STA with linear and nonlinear delay models. The timing graph is traversed topologically, and our bounding/approximation scheme is applied to handle the *max* operation at every stage. This results in expressing the arrival time at the sink node, i.e., the maximum circuit delay, using the delay model at

hand. Once this expression is obtained, it can be easily maximized and minimized as shown in this section to bound/approximate the maximum and minimum corner delays of the maximum circuit delay.

4.5.2 Results

We have tested our approach by first characterizing a 90nm CMOS library using HSPICE to determine the sensitivities of gate delay to various process parameters. We have chosen to vary channel length (L_n and L_p), and threshold voltage (V_{tn} and V_{tp}) of NMOS and PMOS transistors, and have determined the sensitivities of gate delays to these process parameters. The variation information was specified in the technology files. Our technique was then implemented using C/C++ and was tested on the ISCAS85 benchmark circuits. For each circuit, the maximum circuit delay is determined at various corners using exhaustive corner case analysis, and the true maximum and minimum corner delays are recorded. Our framework is then applied, using (1) the lower bound technique, (2) the upper bound technique, and (3) the least squares approximation, to predict the maximum and minimum corner delay for every circuit; the bounds and the approximation are propagated together in the circuit timing graph in one timing run. The results are summarized in Table 4.1, where the values shown are normalized to the true minimum and maximum corner delays respectively, determined by corner case analysis. For example, the second column gives a lower bound on the minimum (over all corners) circuit delay, and the last column gives an approximation of the maximum (over all corners) circuit delay. The closer the values are to 1 the more accurate they are. Note that, unlike [51] where the method is only accurate in predicting the circuit delay at the maximum corner, we can accurately estimate the maximum circuit delay at both the minimum and the maximum corners, so that the spread of the maximum circuit delay is well-captured. In addition, our approach is not restricted to linear models. The last row of the table shows the average percent error for every approach; the maximum and minimum corner delays can be approximated within 0.7% and 1.8% respectively.

Table 4.1: Our methods compared to corner analysis

Circuit	Lower bound		Upper bound		LS Approx	
	Min	Max	Min	Max	Min	Max
c432	0.941	0.959	1.067	1.032	1.006	0.999
c499	0.930	0.966	1.026	1.022	0.982	0.996
c880	0.935	0.961	1.103	1.059	1.018	1.012
c1355	0.928	0.946	1.153	1.097	1.041	1.030
c1908	0.945	0.960	1.122	1.077	1.031	1.016
c2670	0.929	0.951	1.037	1.013	0.983	0.982
c3540	0.932	0.951	1.070	1.037	1.001	0.993
c5315	0.936	0.949	1.115	1.057	1.022	0.999
c6288	0.912	0.932	1.183	1.120	1.051	1.031
c7552	0.942	0.953	1.153	1.082	1.042	1.012
Avg Error	-6.7%	-4.7%	10.3%	5.9%	1.8%	0.7%

4.6 Nonlinear Non-Gaussian SSTA

We now demonstrate nonlinear non-Gaussian SSTA, using a general quadratic delay model that depends on process variables modeled as RVs with arbitrary distributions, as well as uncertain non-random parameters. As has become typical in the SSTA literature [27], we assume that one can deal with within-die correlations using some simple scheme by which correlation is resolved based on the physical location of a cell on the layout surface. As a result, when it comes to those variations that are random variables (as opposed to the uncertain non-random parameters), we assume that we are only dealing with global variations and purely random variations. Before proceeding with the details, we first show how the three properties of the delay model are satisfied for our framework to hold.

4.6.1 Delay Model

Assume that gate delay D is expressed using a quadratic model, $\mathcal{F}$, as follows:

$$D = d_o + \sum_{i=1}^{p} (d_i X_i + \hat{d}_i X_i^2) + d_r X_{D_r} \tag{4.19}$$

where d_o is the nominal delay, d_i and $\hat{d}_i$ are first-order and second-order sensitivities to X_i, and d_r is the sensitivity to the purely independent random variation X_{D_r} specific to D.

Because our framework is indifferent to whether parameters are random or uncertain, there is no restriction on X_i's, which can be either RVs or uncertain non-random parameters alike. If it is random, X_i is modeled as an independent zero mean RV with an arbitrary distribution, such that $-1 \leq X_i \leq 1$. This can be any distribution, including common cases such as a truncated Gaussian (normal) distribution, a uniform distribution, etc. If it is uncertain, X_i is simply assumed to vary in $[-1, 1]$. Note that, in both case, X_i's are global variation sources that are shared by all gate delays and timing quantities. As for the purely independent random variation X_{D_r}, we will assume that it can be modeled as a standard normal distribution with zero mean and unit variance. This is because X_{D_r} are typically the result of various independent random effects that add up and converge to a normal distribution by the central limit theorem.

Add Operation

The *add* operation and generally any linear operation can be easily performed without destroying the above quadratic model. Assume that $[A, B] \in \mathcal{F}$:

$$A = a_o + \sum_{i=1}^{p} (a_i X_i + \hat{a}_i X_i^2) + a_r X_{A_r} \tag{4.20}$$

$$B = b_o + \sum_{i=1}^{p} (b_i X_i + \hat{b}_i X_i^2) + b_r X_{B_r} \tag{4.21}$$

then $C = aA + bB + c$ can be expressed using the quadratic model as follows:

$$\begin{aligned} C &= (aa_o + bb_o + c) + \sum_{i=1}^{p} \left[(aa_i + bb_i) X_i + (a\hat{a}_i + b\hat{b}_i) X_i^2 \right] \\ &+ (aa_r X_{A_r} + bb_r X_{B_r}) \\ &= c_o + \sum_{i=1}^{p} (c_i X_i + \hat{c}_i X_i^2) + c_r X_{C_r} \end{aligned}$$

where $c_r = \sqrt{(aa_r)^2 + (bb_r)^2}$, since X_{A_r} and X_{B_r} are independent standard normal RVs, and X_{C_r} is an independent standard normal RV specific to C. Therefore, $\mathcal{F}$ satisfies property 1.

Max Operation

As explained in Section 4.4, the *max* operation $C = \max(A, B)$ is resolved by first subtracting A and B, i.e., $D = A - B$ and then bounding/approximating C using linear combinations of A and B; recall that the coefficients of the linear combinations are functions of the minimum and maximum values of the difference D, i.e., $D_{\min}$ and $D_{\max}$.

We have already shown how to perform linear operations using the quadratic model. Hence, to apply the results of Section 4.4, it remains to show how the quadratic model is bounded and can be efficiently maximized and minimized over the space of variations (Properties 2 and 3). Assume that we have already performed the subtraction $D = A - B$, and D is given by:

$$D = d_o + \sum_{i=1}^{p} (d_i X_i + \hat{d}_i X_i^2) + d_r X_{D_r} \tag{4.22}$$

Since X_i's and X_{D_r} are all independent, then the maximum $D_{\max}$ and minimum $D_{\min}$ of D are achieved when all variations are maximized and minimized separately. In other words, we need to maximize and minimize the quadratic terms and the independent random term.

The quadratic terms $g(X_i) = d_i X_i + \hat{d}_i X_i^2$ can be analytically maximized and minimized as was described previously in Section 4.5.1. As for the purely independent random variation X_{D_r}, recall that it is modeled as an RV with standard normal distribution, which suggests that X_{D_r} can take values in $-\infty$ to $+\infty$. This, however, is not realistic, and it is usually the case that standard normal distributions are truncated between $[-k, k]$, where k represents multiple standard deviations. As an example, setting $k = 3$ will cover 99.73% of the standard normal distribution. As a result, the maximum and minimum values contributed by the independent random component are kd_r and $-kd_r$ respectively. Hence, by combining the above two contributions, $D_{\min}$ and $D_{\max}$

can be easily determined. Having shown that the three properties of Section 4.4 are satisfied by $\mathcal{F}$, we use the approaches in Sections 4.4.2 and 4.4.3 to resolve the *max* operation.

4.6.2 Complexity Analysis

We have shown that both *add* and *max* operations are resolved using simple linear operations involving the delay model; in addition, the coefficients of the linear combination needed for the *max* operation involve a subtraction and a maximization/minimization performed on the model. If p is the total number of variation sources, then performing an addition or a subtraction is $\mathcal{O}(p)$; also, performing a maximization/minimization of the model is $\mathcal{O}(p)$. This means that both *add* and *max* operations are linear in the number of variation sources. Therefore, the overall complexity of a block-based SSTA technique using the above operations is $\mathcal{O}(pn)$, where n is the circuit size.

4.6.3 Results

We have implemented using C/C++ our nonlinear non-Gaussian block-based SSTA technique that propagates upper and lower bounds on, and an approximation to the maximum circuit delay, and have tested it on the ISCAS85 benchmark suite. Similar to [50], and as a proof of concept, we have generated the variation information randomly. We have chosen the coefficients of the quadratic delay model in such a way that every variation source causes 10% to 20% deviation in the nominal delay. In addition to the purely random variation that follows a truncated Gaussian distribution, four global variation sources X_i's were used, each following either a truncated Gaussian, a uniform, or a triangular distribution as shown in Fig. 4.6. The accuracy of our technique is compared to Monte Carlo analysis with $10,000$ runs. All delays reported are normalized to the nominal circuit delay. Fig. 4.7 shows a plot of the cumulative distribution functions (CDF) of the maximum circuit delay for benchmark c1355 for the case of variations with a uniform distribution, generated using our SSTA technique (upper/lower bounds and approximation) and compared to Monte Carlo simulation. Note that the bounds are valid and accurate, and the least squares (LS) approxima-

Table 4.2: Least-squares SSTA vs Monte Carlo analysis for Gaussian, Uniform, and Triangular distributions

Circuit	Gaussian Distributions						Uniform Distributions						Triangular Distributions					
	LS-SSTA			Monte Carlo			LS-SSTA			Monte Carlo			LS-SSTA			Monte Carlo		
	95% -tile	99% -tile	σ/μ %	95% -tile	99% -tile	σ/μ %	95% -tile	99% -tile	σ/μ %	95% -tile	99% -tile	σ/μ %	95% -tile	99% -tile	σ/μ %	95% -tile	99% -tile	σ/μ %
c432	1.20	1.30	10.7	1.20	1.30	10.6	1.37	1.45	18.4	1.37	1.45	18.4	1.26	1.36	13.2	1.26	1.36	13.3
c499	1.22	1.31	9.35	1.19	1.28	9.18	1.38	1.48	15.6	1.36	1.46	15.9	1.27	1.37	11.4	1.25	1.34	11.4
c880	1.20	1.28	8.98	1.19	1.27	9.07	1.35	1.46	15.1	1.34	1.44	15.1	1.25	1.34	11.0	1.23	1.32	11.0
c1355	1.17	1.24	7.86	1.16	1.23	7.76	1.30	1.39	13.4	1.30	1.38	13.6	1.21	1.29	9.67	1.21	1.28	9.78
c1908	1.19	1.28	9.95	1.19	1.27	9.75	1.35	1.46	16.8	1.35	1.46	16.7	1.24	1.34	12.2	1.25	1.34	12.1
c2670	1.22	1.32	11.7	1.21	1.32	11.5	1.39	1.49	19.8	1.40	1.48	19.8	1.28	1.39	14.3	1.28	1.39	14.2
c3540	1.22	1.32	9.52	1.20	1.29	9.64	1.37	1.45	16.2	1.36	1.43	16.4	1.28	1.37	11.7	1.25	1.35	11.7
c5315	1.22	1.33	11.6	1.22	1.32	11.6	1.39	1.48	19.8	1.40	1.48	19.9	1.28	1.39	14.3	1.27	1.38	14.1
c6288	1.21	1.30	9.56	1.19	1.28	9.50	1.37	1.46	16.3	1.36	1.45	16.5	1.26	1.35	11.8	1.25	1.35	12.0
c7552	1.22	1.33	11.7	1.22	1.33	11.7	1.40	1.48	19.8	1.39	1.48	19.7	1.28	1.38	14.3	1.27	1.38	14.2
AvgErr%	-0.80	-0.80	-0.51	-	-	-	-0.37	-0.54	0.55	-	-	-	-0.68	-0.59	0.04	-	-	-

tion is very close to the true distribution. Table 4.2 compares the following timing metrics: the 95 and the 99 delay percentiles, and the ratio of standard deviation to the mean σ/μ, generated using the least squares approximation (LS-SSTA) to Monte Carlo analysis, for Gaussian, uniform, and triangular distributions. The results show that our approach is consistently accurate for all metrics and different distributions, with an average error less than 1%.

In the mixed case, where one is dealing with both random variables and uncertain variables, the overall (random) circuit delay and the interesting statistical metrics (like mean, variance, percentiles) are effectively functions of the uncertain variables. If this functional dependence is complex, then the desirable worst-case values (of the interesting statistical metrics) must be found by a process of search or optimization, which is beyond the scope of this thesis. For now, the above results were obtained at a specific setting (the nominal) of the uncertain variables. This does not diminish the value of the results, because one strength of our existing approach is that it provides an *explicit* dependence of the total circuit delay on the uncertain parameters, so that one does not simply have to repeat the overall SSTA for different settings.

4.7 Summary

In this chapter, we proposed a general parameterized static timing analysis technique that can handle nonlinear delay models and account for delay variability due to both *random* process parameters with arbitrary distributions and *uncertain* non-random parameters such as supply voltage and temperature which depend on circuit operation. Central to this technique is a novel and efficient method to resolve the *max* operator by bounding it and approximating it using linear models, while preserving the inherent nonlinearity of the delay model itself. Two applications of this framework were proposed, namely multi-corner timing analysis and nonlinear non-Gaussian SSTA, and have shown that the complexity of the approach is linear in both the number of process and environmental parameters and the size of the circuit. Our results show that, on average, circuit delay is predicted with less than 2% error for multi-corner analysis, and less than 1% error for SSTA.

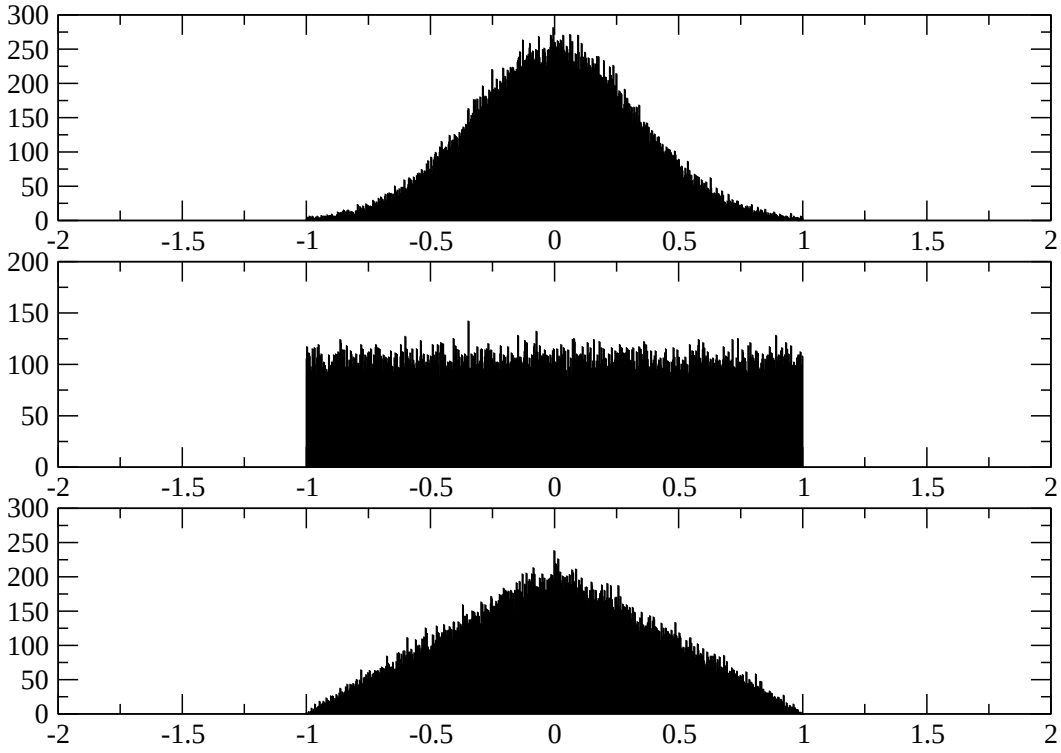


Figure 4.6: Truncated Gaussian, uniform, and triangular distributions

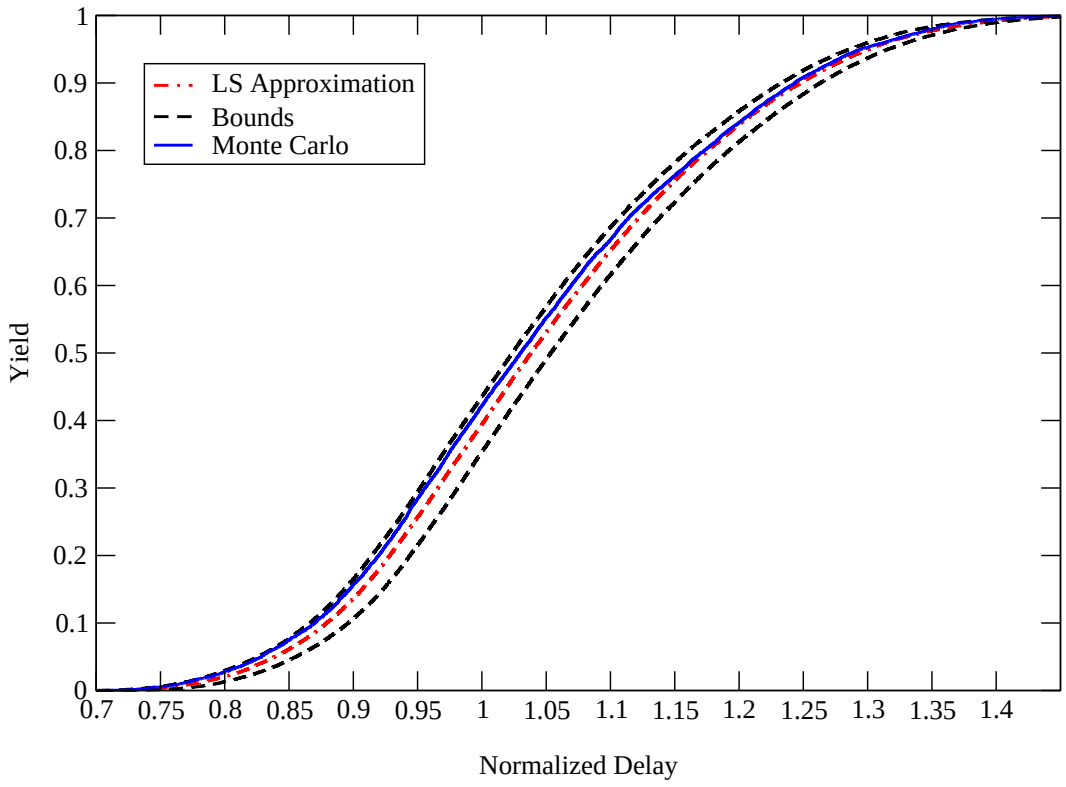


Figure 4.7: CDF comparison for c1355

5 Parameterized Static Timing Analysis Covering All Potentially Critical Paths

5.1 Introduction

A limitation of many recent variability-aware timing analysis techniques is that, while they report delay distributions, or verify multiple corners, they do not provide the required guidance for re-design. In this chapter, we propose an efficient block-based parameterized static timing analysis technique that can accurately capture circuit delay at every point in the parameter space, by reporting all the paths that can become critical. Using an efficient pruning algorithm, only those *potentially critical* paths are carried forward, while all other paths are discarded during propagation. This allows one to examine local robustness to parameters in different regions of the parameter space, not by considering differential sensitivity at a point (which would be useless in this context) but by knowledge of the paths that can become critical at nearby points in parameter space. We give a formal definition of this problem and propose a technique for solving it that improves on the state of the art, both in terms of theoretical computational complexity and in terms of run time on various test circuits.

5.2 Background

We have seen in Chapter 2 that signal and clock path delays in integrated circuits are subject to variations arising from many sources, including (manufacturing) process variations, (supply/ground) voltage variations, and temperature variations. These are collectively referred to as PVT variations. During design, one accounts for the delay

variability by either "padding" the path delays with a *timing margin*, so that the chip would yield well at all process corners in spite of the variations, or by "binning" the resulting chips at different frequencies. As mentioned in earlier chapters, the scale of the problem has increased recently, because *i)* an increasing number of circuit parameters have significant variability, causing an increase in the number of corners, and *ii)* within-die variations are becoming more significant, and they cannot be handled by the traditional corner-based approach. In Chapter 4, we have classified the variables or parameters under study into two types: many transistor and metal line parameters are directly related to underlying statistical process variables, so they may be modeled as *random variables*, with certain distributions; on the other hand, the supply/ground voltage and temperature are not random, and must be modeled as simply unknown or *uncertain variables*, within known bounds.

Given the two types of variables under study, two types of solution techniques have emerged: statistical static timing analysis (SSTA) and multi-corner static timing analysis (MCSTA). SSTA models parameters as random variables, assuming that their distributions and correlations are known a priori [27, 28, 37], and provides the distribution of circuit delay, from which the timing yield can be estimated. On the other hand, MCSTA models the PVT parameters as uncertain variables, within given bounds, and attempts to verify the timing at all corners in a single timing run [51]. All these techniques consider the circuit delay to be dependent on a number of PVT parameters, be they random or uncertain. Therefore, one can describe the required overall solution to this problem as *parameterized static timing analysis* (PSTA). In Chapter 4, we proposed a general PSTA framework where the max operator is bounded and approximated (essentially linearized), allowing us to maintain the same delay model during propagation. In this chapter however, we approach the problem in a different way.

5.3 Overview

The motivation for the work in this chapter is the simple notion that, for the results of timing analysis to be useful, they must provide guidance on how the circuit may be improved so as to fix any reported timing problems. To understand the need for

PSTA in general, consider the simple case where delay is linear in the variational (PVT) parameters. In a circuit, the delay of any input-output path becomes a linear expression in terms of the parameters, or what we refer to as a *hyperplane*. At the nominal PVT point, the hyperplane corresponding to the path with the largest delay (under nominal conditions) is *dominant* (over all others). As we move around in PVT space, some other path may become critical, and, correspondingly, another hyperplane may become dominant. Overall, across the whole PVT space, the total circuit delay follows some piece-wise planar (PWP) surface. This surface is defined by all the hyperplanes which can become dominant at *some* point in PVT space. We refer to these hyperplanes as *potentially dominant* and to their corresponding paths as *potentially critical*.

Suppose we are at some operating point in PVT space, and we are interested in the *robustness* of the circuit at that point. In other words, we are interested in the impact of variations on overall circuit delay around that point. What information would be useful to the designer in this case? One could consider providing the *sensitivity* of delay, at that point, to the various PVT parameters, such as by means of the partial derivatives of delay to each of the parameters. However, because of the PWP nature of the delay surface, such point metrics are actually useless. One may find the derivatives to have low values at that point, yet one may be very close to a "break point" in the surface where another hyperplane with much larger sensitivities suddenly becomes dominant. Instead, one must be able to quickly discover what paths (i.e., hyperplanes) become dominant in a certain neighborhood around the point of interest. Given a list of problematic paths in the neighborhood, when working on fixing some path, one avoids being "blind-sighted" to the criticality of other paths. Thus, for the results of timing analysis to be useful, we believe that the whole PWP surface is required. It is not enough to give the user the worst-case corner; that does not provide a full picture of what needs to be fixed. Also, simply providing the timing yield, as is done in SSTA, or simply providing a list of a large number of paths, with a failure probability for each, does not give sufficient insight for what paths need to be fixed around the operating point. Instead, a PWP surface (for the total circuit delay) allows one to examine the local neighborhood to see which parameters and paths may be problematic (so that one can focus on them as part of redesign). It should be mentioned that the

"broken" nature of the delay surface is not due to the linearity assumption. Instead, it is actually due to the *max* function which is implicit in the problem of timing verification of setup constraints (a similarly broken delay surface results from the *min* function, in the similar problem of verifying hold constraints). If one assumes a non-linear, say polynomial, delay dependence, one simply ends up with a piece-wise polynomial surface, which presents the same sort of problems.

To faithfully represent the PWP surface for the total circuit delay, we must include (during propagation in the timing graph) all the hyperplanes that can become dominant *somewhere* in PVT space. Simply carrying along *all* paths can be problematic due to possible path count explosion; hence, an efficient *pruning strategy* is needed, whereby redundant paths that cannot become dominant *anywhere* in PVT space are identified and *pruned* during the propagation. This problem was studied in [52], where an exact pruning algorithm and a sufficient condition for pruning were proposed, and where it was found that indeed the number of potentially dominant paths is manageable and does not explode. In that work, the exact algorithm (as we will see) has time complexity $\mathcal{O}(p^2n^2)$, where p is the number of PVT parameters and n is the number of hyperplanes to be pruned, and the sufficient condition is $\mathcal{O}(pn^2)$. In this chapter, we propose: (1) a more efficient exact solution to the pruning problem that takes $\mathcal{O}(p^2mn)$ time, where m is the number of potentially dominant hyperplanes at the circuit outputs, and (2) a sufficient condition for pruning that is $\mathcal{O}(pn)$. We will see that the resulting improvements in run-time can be significant for *hard* circuits.

5.4 Preliminaries

In this section, we first review some basic terminology covering timing modeling and propagation. Then, we describe the problem formulated by the authors of [52] and briefly review their approach.

5.4.1 Modeling and Propagation

In block-based timing analysis, timing quantities are propagated in the timing graph in topological order, through a sequence of basic operations, such as *add* operations on input arrival times and arc delays, and *max* operations on the timing quantities resulting from those additions. In this way, the output arrival time is determined and is then propagated to subsequent stages. This is shown in Fig 5.1, where the arrival time C at the output of the AND gate is computed as the *max* of the sums of arrival times and arc delays at the inputs of the gate. In other words:

$$C = \max\left(A + D_1, B + D_2\right) \tag{5.1}$$

where A and B are input arrival times, and D_1 and D_2 are timing arc delays. This can be easily generalized to gates with more than two inputs.

Since variability in the process and environmental (PVT) parameters affects transistor performance, gate delays should be represented in such a way to highlight their dependence on these underlying parameters. First-order linear delay models have been extensively used in the literature, and they generally capture well this dependence. In this chapter, we assume that gate delay is a linear function of process and environmental parameters, such as channel length L, threshold voltage V_t, supply voltage V_{dd}, and temperature T. These parameters are assumed to vary in specified ranges, however, without loss of generality, we can easily normalize these ranges to $[-1, 1]$, similarly to what was done in [51]. Hence, gate delay D, can be expressed as follows:

$$D = d_o + \sum_{i=1}^{p} d_i X_i, \quad -1 \leq X_i \leq 1 \quad \forall i \tag{5.2}$$

where d_o is the nominal delay, X_i's are the normalized PVT parameters, and d_i's are the delay sensitivities to these parameters. Since D is a linear function of p parameters, then it is referred to as a delay *hyperplane*.

The delay of a path is simply the *sum* of arc delays of all gates on that path. Since arc delays are expressed as hyperplanes, so will be the path delay; in the rest of the chapter, when we refer to the delay of a path, it is understood that we mean *path delay*

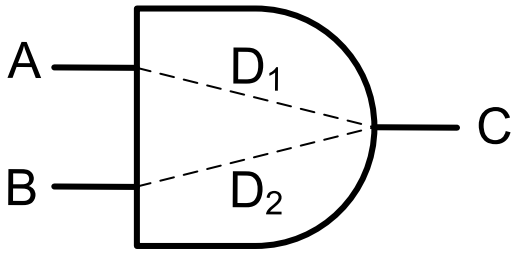


Figure 5.1: Propagation for a single gate

hyperplane. Although this is true for a path, the arrival time at a node, which is the *max* of all path delay hyperplanes in the fan-in cone of that node, is not necessarily a hyperplane. This is shown in Fig. 5.2, where four paths, $P_1 - P_4$, converge at a node. The arrival time, A, at that node is given by:

$$A = \max(P_1, P_2, P_3, P_4) \tag{5.3}$$

Shown as the broken dashed line, A is a piece-wise planar (PWP) surface because either P_1, P_2, or P_3 can become the maximum (or dominant) hyperplane, depending on which region of the parameter space is under consideration. Note that P_4 is always *covered* by another hyperplane, and therefore does not show up in the PWP surface. Paths, such as $P_1 - P_3$, which can become dominant are referred to as *potentially critical* or *non-redundant* paths, whereas paths, such as P_4, which cannot become critical, are referred to as *redundant* or *prunable* paths. We will formally define these terms in the next section. Ideally, during analysis, only those potentially critical paths (or non-redundant hyperplanes) must be propagated to subsequent stages, while all other hyperplanes must be discarded or pruned.

5.4.2 The Pruning Problem

Let D_j be the delay hyperplane of path j in a set of n paths converging on a node, so that D_j is given by:

$$D_j = a_{oj} + \sum_{i=1}^{p} a_{ij} X_i, \quad j = 1, \ldots, n \tag{5.4}$$

The hyperplane D_j is said to be redundant or prunable if and only if:

$$\max(D_1, \ldots, D_n) = \max(D_1, \ldots, D_{j-1}, D_{j+1}, \ldots, D_n), \quad \forall X_i \tag{5.5}$$

In this case, no matter where we are in the parameter space, D_j will never show up as the maximum hyperplane, as other hyperplanes will be dominating it. An example of this is path P_4 in Fig 5.2; such a redundant hyperplane can be pruned from the set without affecting the shape of the piece-wise planar surface representing the max. On the other hand, if (5.5) is not satisfied, then D_j is a non-redundant hyperplane and must be kept in the set. An example of this are paths $P_1 - P_3$, which show up in the PWP surface.

Formally, the pruning problem can be stated as follows. Given a set $\mathcal{P}$ of n hyperplanes D_j, find the set $\mathcal{Q} \subseteq \mathcal{P}$, such that $\mathcal{Q}$ is an irreducible set of m non-redundant hyperplanes $\tilde{D}_j$, where $m \leq n$, and such that:

$$\max(D_1, \ldots, D_n) = \max(\tilde{D}_1, \ldots, \tilde{D}_m), \quad \forall X_i \tag{5.6}$$

Only those m non-redundant hyperplanes are needed to describe the shape of the PWP surface defined by the max. This pruning problem was studied by the authors of [52] who proposed two techniques for pruning. We now review these techniques and describe some of their limitations.

Pairwise Pruning

The first technique is based on pairwise comparisons between hyperplanes, to check if any hyperplane can prune another hyperplane, as follows. Let D_1 and D_2 be two

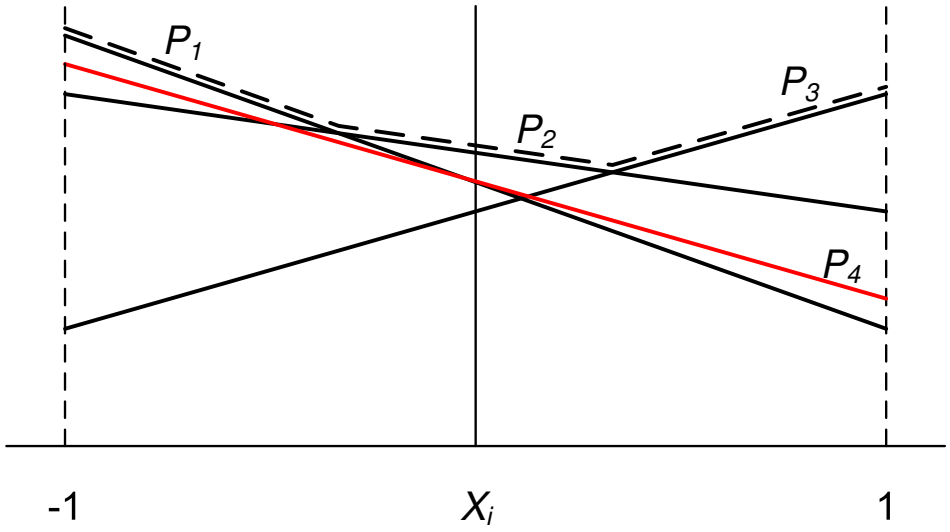


Figure 5.2: MAX of path delay hyperplanes

hyperplanes:

$$D_1 = a_{o1} + \sum_{i=1}^{p} a_{i1} X_i \tag{5.7}$$

$$D_2 = a_{o2} + \sum_{i=1}^{p} a_{i2} X_i \tag{5.8}$$

If $D_1 - D_2 \leq 0$ for all values of X_i, then D_1 is pruned by D_2, denoted by $D_1 \prec D_2$. Since $-1 \leq X_i \leq 1$, then $D_1 \prec D_2$ if and only if:

$$a_{o1} - a_{o2} + \sum_{i=1}^{p} |a_{i1} - a_{i2}| \leq 0 \tag{5.9}$$

which can be easily checked.

The pairwise pruning procedure [52] is shown in Algorithm 1, where we have preserved the same flow as in [52] for clarity. It has two nested loops that cover all pairs of hyperplanes, checking if $D_j \prec D_i$. Note that this algorithm is only a *sufficient con-*

dition for pruning and is not an exact solution for the pruning problem. In fact, the resulting set $\mathcal{Q}$ is not necessarily an irreducible set. Going back to Fig. 5.2, PAIRWISE will fail to identify P_4 as a redundant hyperplane since $P_1, \cdots, P_4$ are pairwise non-prunable. In addition, the complexity of PAIRWISE is $\mathcal{O}(pn^2)$, where p is the number of PVT parameters and n is the number of hyperplanes. This quadratic complexity can be problematic, particularly if a large number of redundant hyperplanes, which are identified as non-redundant by PAIRWISE, are propagated to subsequent stages, potentially causing a blow-up in the number of hyperplanes, as reported by [52] on one of the test circuits.

Algorithm 1 PAIRWISE

Input: $\mathcal{P} = \{D_1, \ldots, D_n\}$;
Output: $\mathcal{Q} \supseteq \{\tilde{D}_1, \ldots, \tilde{D}_m\}$;

```
1: Mark all hyperplanes in P as non-redundant;
2: for i = 1 : n do
3:    if (D_i is marked redundant) then
4:       continue;
5:    end if
6:    for j = 1 : n do
7:       if (D_j is marked redundant) then
8:          continue;
9:       end if
10:      if (D_j ≺ D_i) then
11:         Mark D_j as redundant;
12:      end if
13:   end for
14: end for
15: Add all non-redundant hyperplanes to Q;
```

Feasibility Check

The second pruning technique is a *necessary and sufficient* condition for pruning. It is therefore an exact solution for the pruning problem, which guarantees that the resulting set $\mathcal{Q}$ is an irreducible set of non-redundant hyperplanes. The idea is to perform a *feasibility check* for every hyperplane D_j by searching for a point in the space of X_i's

where D_j is dominant over *all* other hyperplanes. If this is feasible, then D_j is non-redundant, otherwise, D_j is redundant and can be pruned from the set. Thus, D_j is non-redundant if and only if the following system of inequalities has a feasible solution:

$$\begin{aligned} D_j &\geq D_k, \quad k = 1, \ldots, n,\ k \neq j \\ -1 &\leq X_i \leq 1, \quad i = 1, \ldots, p \end{aligned} \tag{5.10}$$

Algorithm 2 describes FEASCHK, where a feasibility check is formulated for every hyperplane in the starting set $\mathcal{P}$ (line 6). If there is a feasible solution, then the hyperplane is non-redundant and is added to $\mathcal{Q}$. Otherwise, it is marked as redundant and is pruned from the set.

Algorithm 2 FEASCHK

Input: $\mathcal{P} = \{D_1, \ldots, D_n\}$;
Output: $\mathcal{Q} = \{\tilde{D}_1, \ldots, \tilde{D}_m\}$;

```
1:  Mark all hyperplanes in P as non-redundant;
2:  for j = 1 : n do
3:     if (D_j is marked redundant) then
4:        continue;
5:     end if
6:     Formulate (5.10) for D_j excluding redundant hyperplanes;
7:     if (feasible) then
8:        Add D_j to Q;
9:     else
10:       Mark D_j as redundant;
11:    end if
12: end for
```

Note that the feasibility check in (5.10) consists of solving a Linear Program (LP) with p variables and $(n+p)$ constraints, which has a complexity of $\mathcal{O}\left(p^2(n+p)\right)$, if an interior-point based LP solver is used [53]. Therefore, the complexity of FEASCHK, which requires n feasibility checks to determine the irreducible set of non-redundant hyperplanes is $\mathcal{O}\left(p^2(n+p)n\right)$, which is $\mathcal{O}(p^2n^2)$ if $p \leq n$, which is usually the case. Given that this pruning algorithm would potentially be applied at every node in the timing graph, its $\mathcal{O}(n^2)$ behavior in the number of hyperplanes can be expensive.

In the following sections, we present a more efficient method for solving the pruning problem. By transforming this problem into a standard problem in *computational geometry*, we present an exact pruning algorithm which is $\mathcal{O}(p^2mn)$, where n is the number of hyperplanes in the initial set $\mathcal{P}$, and m is the number of non-redundant hyperplanes in the final irreducible set $\mathcal{Q}$. We also propose a sufficient condition for pruning that can be used as a pre-processing step, and which is $\mathcal{O}(pn)$.

5.5 Problem Transformation

In this section, we show how we map our parameterized timing pruning problem into a standard problem in computational geometry.

5.5.1 From Computational Geometry

The field of computational geometry deals with the study of algorithms to solve problems stated in terms of geometry. Typical problems include Convex Hull, Vertex/Facet enumeration, and Voronoi diagrams, to name a few [54]. We have identified two standard problems that can be related to the pruning problem: *Enumeration of Extreme Points of a Convex Hull* and its equivalent (dual) problem of *Minimal Representation of a Polytope*. We first review these problems and show how the pruning problem can be transformed into a standard problem.

Extreme Points Enumeration

To understand this problem, let us start by defining the following terms:

Definition 1 (Convex Hull) *The convex hull of a set P of n points, denoted as $conv(P)$, is the smallest convex set that contains these points.*

Definition 2 (Extreme Points) *Given a set P of n points in d dimensions, the minimal subset E of P for which $conv(P) = conv(E)$ is called the set of extreme points. In other words, if point $e \in E$, then $conv(P \setminus \{e\}) \neq conv(P)$.*

The *Extreme Points Enumeration* problem can be stated as follows. Given a set P of n points, determine the minimal subset E of m extreme points, where $m \leq n$. This is shown graphically in Fig. 5.3a where the shaded region is the convex hull, and points 1 through 4 are the set of extreme points. Note that points 5 and 6 do not contribute to the convex hull and can thus be removed.

Minimal Polytope Representation

We begin by defining some terms that will help us introduce this standard problem:

Definition 3 (Hyperplane) *It is the set* $\{x \mid a^T x = b\}$*, where* $a \in \mathbb{R}^d$*,* $a \neq 0$ *and* $b \in \mathbb{R}$*. It is the solution set of a nontrivial linear equation among the components of* x*. A hyperplane divides* $\mathbb{R}^d$ *into two half-spaces.*

Definition 4 (Half-space) *It is the set* $\{x \mid a^T x \leq b\}$*, where* $a \in \mathbb{R}^d$*,* $a \neq 0$ *and* $b \in \mathbb{R}$*. It is the solution set of one nontrivial linear inequality.*

Definition 5 (Polyhedron/Polytope) *A polyhedron is the set* $P \subseteq \mathbb{R}^d$*, such that:*

$$P = \{x \mid a_j^T x \leq b_j, \quad j = 1, \ldots, n\} \tag{5.11}$$

It is therefore the intersection of a finite number of half-spaces. A bounded *polyhedron is called a polytope. A polyhedron/polytope can be written in matrix form as follows:*

$$P = \{x \mid Ax \leq b\} \tag{5.12}$$

where A *is an* $n \times d$ *matrix, and* $b \in \mathbb{R}^n$*. Note that* A *is not necessarily the minimal representation of* P*.*

Definition 6 (Supporting Hyperplane) *If one of the two closed half-spaces of a hyperplane* h *contains a polytope* P*, then* h *is called a supporting hyperplane of* P*. Note that every row in the matrix representation of the polytope* P *corresponds to a supporting hyperplane.*

For example, the shaded region in Fig. 5.3b is a polytope defined as the intersection of six half-spaces, each bounded by a hyperplane, and all six hyperplanes are *supporting hyperplanes*.

Definition 7 (Bounding Hyperplane) *A hyperplane that is spanned by its intersection with a polytope P is called a bounding hyperplane of P. Those rows in the matrix representation of P, which can be satisfied with equality for some values of x, correspond to bounding hyperplanes of P.*

For example, in Fig. 5.3b, only hyperplanes 1 through 4 are *bounding hyperplanes*; they appear at the boundary of the polytope.

With the above definitions, the problem of *Minimal Polytope Representation* can be stated as follows. Given a polytope P with n supporting hyperplanes, find all m bounding hyperplanes of the polytope, where $m \leq n$. This will correspond to the minimal representation of P. In other words, if P is defined as $Ax \leq b$, where A is an $n \times d$ matrix, and $b \in \mathbb{R}^n$, find a reduced $\tilde{A}$ and $\tilde{b}$ such that:

$$P = \{x \mid Ax \leq b\} = \{x \mid \tilde{A}x \leq \tilde{b}\} \tag{5.13}$$

where $\tilde{A}$ and $\tilde{b}$ are the rows of A and b that correspond to the m bounding hyperplanes of P. Referring again to the example in Fig. 5.3b, if hyperplanes 5 and 6 were removed, it would not affect the shape of the polytope.

The above two problems have obvious similarities; in fact, these two problems are equivalent, as one is the dual of the other. This can be explained by the *point-hyperplane* duality in computational geometry [54]. Point-hyperplane duality is a common transformation whereby a point p at distance r from the origin O is associated with the hyperplane normal to Op at distance $1/r$ from the origin. Under this transformation, extreme points enumeration and minimal polytope representation are two equivalent problems. This is shown in Fig 5.3, where the extreme points 1 to 4 of the convex hull are transformed to the bounding hyperplanes 1 to 4 of the polytope; also, the points 5 and 6 on the interior of the convex hull are transformed to hyperplanes 5 and 6, which do not appear in the minimal polytope representation. Therefore, an algorithm that can solve one problem efficiently can also be used to solve the other problem,

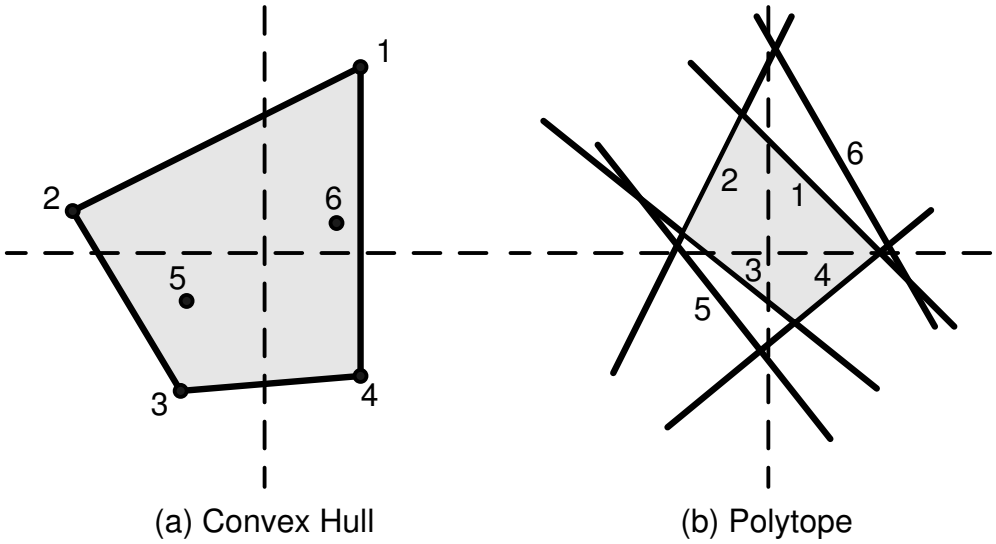


Figure 5.3: (a) Extreme points of convex hull (b) Minimal polytope representation (dual problem)

and vice-versa. We have identified an efficient algorithm in [55], which solves extreme points enumeration. The same algorithm can be used to solve the dual problem of minimal polytope representation. In the next section, we show how the pruning problem, defined in Section 5.4.2, can be transformed to minimal polytope representation problem. Once this is established, we can adapt the algorithm in [55] to solve the pruning problem efficiently.

5.5.2 To Parameterized Timing

Recall the pruning problem, where given a set of n delay hyperplanes D_j, we want to determine every hyperplane that can become the maximum hyperplane, for some setting of the PVT parameters. These hyperplanes are referred to as non-redundant, whereas other hyperplanes that cannot become dominant are referred to as redundant hyperplanes, and should be pruned. Let $D_{\max}$ be the piece-wise planar maximum of all n hyperplanes, i.e., $D_{\max} = \max(D_1, \cdots, D_n), \forall X_i$, such as the broken line in Fig. 5.2.

As a result, the following condition holds:

$$D_{\max} \geq D_j = a_{oj} + \sum_{i=1}^{p} a_{ij} X_i, \quad -1 \leq X_i \leq 1, \ \forall i, \ \forall j \tag{5.14}$$

Note that if D_j is a non-redundant hyperplane, then the above inequality will be satisfied with equality, i.e., $D_{\max} = D_j$, when D_j becomes the maximum hyperplane for some setting of X_i's. Otherwise, if D_j is redundant, then $D_{\max} > D_j$ for all X_i's.

By rearranging (5.14) so as to include $D_{\max}$ in the parameters, we get the following:

$$\sum_{i=1}^{p} a_{ij} X_i - D_{\max} \leq -a_{oj}, \quad j = 1, \ldots, n \tag{5.15}$$

Let $x = [X_1 \ X_2 \ \cdots \ X_p \ D_{\max}]^T$, $h_j = [a_{1j} \ a_{2j} \ \cdots \ a_{pj} \ -1]^T$, and $b_j = -a_{oj}$. Then, we can write the above inequality as:

$$h_j^T x \leq b_j, \quad j = 1, \ldots, n \tag{5.16}$$

Finally, if $b = [b_1 \ b_2 \ \cdots \ b_n]^T$ and $H = [h_1 \ h_2 \ \cdots \ h_n]^T$. Then, we can write the above inequalities in matrix form:

$$Hx \leq b, \quad -1 \leq X_i \leq 1 \tag{5.17}$$

This defines a polytope $\mathcal{H}$ in $p+1$ dimensions, where p is the number of the PVT parameters. Now if we were to find the minimal representation of $\mathcal{H}$, this would result in determining all the rows that correspond to bounding hyperplanes, that is, the rows that can be satisfied with equality, as explained in Definition 7. If row j is a bounding hyperplane, then $h_j^T x = b_j$ is satisfied for some parameter setting. By rearranging this equality in terms of $D_{\max}$, we get $D_{\max} = D_j$, which is the condition for which a delay hyperplane is non-redundant in the pruning problem, as observed above. Therefore, determining the minimal representation of $\mathcal{H}$ would actually solve the pruning problem and determine the set of non-redundant delay hyperplanes.

5.6 Pruning Algorithm

In this section, we present two pruning algorithms: (1) an exact solution to the pruning problem, which is a modified version of the algorithm in [55], and (2) a sufficient condition for pruning that is linear in the number of hyperplanes, and which can be used to speed up the pruning algorithm by reducing the number of calls of the exact algorithm during propagation in the timing graph.

5.6.1 Exact Algorithm

Description

In the minimal polytope representation problem, one needs to identify which rows of the defining polytope matrix correspond to bounding hyperplanes. Let $\mathcal{H}$ be a polytope defined by the system of inequalities $Hx \leq b$, and let $h_j^T x \leq b_j$ be a row in that system. To test whether $h_j^T x \leq b_j$ corresponds to a bounding hyperplane, we need to check if $h_j^T x = b_j$ is satisfied for some value of x; this can be tested using the following LP:

$$\begin{aligned} \text{maximize} \quad &: \quad v = h_j^T x \\ \text{such that} \quad &: \quad Hx \leq b \end{aligned} \tag{5.18}$$

If the solution is $v^* < b_j$, then the hyperplane $h_j^T x = b_j$ is not a bounding hyperplane and can be removed (pruned) from the system; this means that other inequalities are acting in such a way that $h_j^T x \leq b_j$ is never "pushed to its boundary." Otherwise, if the solution is $v^* = b_j$, then $h_j^T x = b_j$ is a bounding hyperplane of the polytope and should be kept in the system.

The LP in (5.18) is formulated in Procedure 3, Check_Redund(), below. This procedure takes as inputs a set of delay hyperplanes $\mathcal{B}$, and a hyperplane $D \in \mathcal{B}$ that we're trying to prune. The delay hyperplanes are first transformed from the parameterized timing domain to the computational geometry domain, where a polytope $Hx \leq b$ is created. Then, $h_j^T x \leq b_j$ is checked to see if it corresponds to a bounding hyperplane of the polytope by formulating the LP in (5.18). If so, then pruned=FALSE, and D is non-redundant, otherwise, pruned=TRUE and D is redundant. The procedure

also returns x^*, which is a *solution witness* generated by the LP solver, i.e., it is the value of x at which the LP maximum v^* is found. Recall that the complexity of an LP is linear in the number of constraints [53]. Therefore, the complexity of Check_Redund() is linear in the size of $\mathcal{B}$; specifically, it is $\mathcal{O}(p^2|\mathcal{B}|)$.

Procedure 3 Check_Redund($D, \mathcal{B}$)

Inputs: Hyperplane D, and a set of hyperplanes $\mathcal{B}$ including D;
Outputs: pruned={TRUE, FALSE}, x^* solution witness;

1: $D \Rightarrow h_j^T x \leq b_j$ and $\mathcal{B} \Rightarrow Hx \leq b$; {transform inputs from delay domain to polytope domain as described in Section 5.5.2}
2: Formulate the LP in (5.18) and get x^* as solution witness;
3: **if** $(h_j^T x^* = b_j)$ **then**
4: pruned = FALSE;
5: **else**
6: pruned = TRUE;
7: **end if**
8: **return** (pruned, x^*)

Algorithm 4 describes the exact pruning algorithm PRUNE, which uses Check_Redund(). PRUNE takes a set of delay hyperplanes $\mathcal{P}$ and determines the set $\mathcal{Q} \subseteq \mathcal{P}$ of non-redundant hyperplanes. The algorithm starts by determining a small subset of non-redundant hyperplanes by calling a procedure Get_Initial_NR(), shown in Procedure 5. Get_Initial_NR() probes the delay hyperplanes at a predefined set of points in the parameter space X_i, to determine which delay hyperplane is maximum at every point. Those hyperplanes that show up as maximum hyperplanes are non-redundant, and are therefore added to the initial set. In addition to the nominal probing point, $X_i = 0$ $\forall i$, $2p$ probes are chosen such that $X_j = \pm 1,\ X_i = 0\ \forall i \neq j,\ j = 1, \ldots, p$, which makes Get_Initial_NR() linear in the number of hyperplanes and the number of probes; specifically, it is $\mathcal{O}(pn)$.

Once this initial set of non-redundant hyperplanes is determined, PRUNE creates the set of remaining hyperplanes $\mathcal{P}'$ (line 2), and starts a loop until $\mathcal{P}'$ is empty (line 20). In every run of the loop, a hyperplane D is removed from $\mathcal{P}'$, and is first checked for redundancy against the set $\mathcal{Q}$ of non-redundant hyperplanes that were discovered so far, by calling Check_Redund() (line 6). If $\mathcal{Q}$ prunes D, then D is definitely pruned

Algorithm 4 PRUNE

Input: Set of hyperplanes $\mathcal{P} = \{D_1, \ldots, D_n\}$ of size n;
Output: Set of all non-redundant hyperplanes $\mathcal{Q}$ of size $m \leq n$;

$\mathcal{Q}$ = Get_Initial_NR($\mathcal{P}$);
$\mathcal{P}' = \mathcal{P} \setminus \mathcal{Q}$;
repeat
  Let D be the next hyperplane in $\mathcal{P}'$;
  Remove D, $\mathcal{P}' = \mathcal{P}' \setminus \{D\}$;
  [pruned, x^*] = Check_Redund($D, \mathcal{Q} \cup \{D\}$); {run a small LP}
  if (pruned = TRUE) **then**
    D is redundant and is not added to $\mathcal{Q}$;
  else
    [pruned, x^*] = Check_Redund($D, \mathcal{Q} \cup \mathcal{P}' \cup \{D\}$); {run a large LP}
    if (pruned = FALSE) **then**
      D is non-redundant;
      Add it to set, $\mathcal{Q} = \mathcal{Q} \cup \{D\}$;
    else
      D is redundant and is not added to $\mathcal{Q}$;
      Use witness x^* to get a set $\mathcal{W}$ of non-redundant hyperplanes containing x^*;
      Add $\mathcal{W}$ to set, $\mathcal{Q} = \mathcal{Q} \cup \mathcal{W}$;
    end if
  end if
until $\mathcal{P}' = \{\}$

by the bigger set $\mathcal{P}$, and is therefore discarded as a redundant hyperplane. Otherwise, if D is found to be non-redundant against $\mathcal{Q}$, then we cannot claim that it is non-redundant in $\mathcal{P}$. Hence, Check_Redund() is called again (line 10) where D is checked against the bigger set $\mathcal{D} \cup \mathcal{P}'$. If D could not be pruned (line 11), then D is a non-redundant hyperplane and is added to $\mathcal{Q}$. Otherwise, D is pruned and discarded as a redundant hyperplane. Recall that Check_Redund() formulates the LP in (5.18) and returns a solution witness x^*. Although D is identified as redundant (line 15), the solution witness x^* can be used to check which constraints of the LP were satisfied with equality at x^*; those satisfied would correspond to bounding hyperplanes of the polytope. Hence a set $\mathcal{W}$ of non-redundant delay hyperplanes can be identified (line 16), which are added to $\mathcal{Q}$. The authors of [55] prove that at least 1 new non-redundant hyperplane is discovered in this step. The reader is referred to [55] for more details about the proof of correctness of this algorithm.

Procedure 5 Get_Initial_NR($\mathcal{P}$)

Input: Set of hyperplanes $\mathcal{P} = \{D_1, \ldots, D_n\}$;
Output: Subset $\mathcal{Q}$ of non-redundant hyperplanes;
1: $\mathcal{Q} = \{\}$;
2: Find D_j with maximum nominal delay a_{oj};
3: $\mathcal{Q} = \mathcal{Q} \cup \{D_j\}$;
4: Set X_i to 0, $\forall i$;
5: **for** i=1:p **do**
6: Set X_i to 1;
7: Find D_j with maximum value at X_i;
8: $\mathcal{Q} = \mathcal{Q} \cup \{D_j\}$;
9: Set X_i to -1;
10: Find D_j with maximum value at X_i;
11: $\mathcal{Q} = \mathcal{Q} \cup \{D_j\}$;
12: Reset X_i to 0;
13: **end for**

Complexity

The execution time of PRUNE is dominated by the time to solve the LPs formulated in Check_Redund() at lines 6 and 10. Line 6 is called for all hyperplanes in $\mathcal{P}'$, which

is $\mathcal{O}(n)$ times. The LP in line 6 has at most p variables and $\mathcal{O}(m)$ constraints, because m is the largest possible size of $\mathcal{Q}$. So in total, this would take $\mathcal{O}(np^2m)$. As for the second LP formulated by Check_Redund() in line 10, notice that every time it is called, a new non-redundant hyperplane is discovered, either explicitly (as in lines 11-12), or through the use of the solution witness x^* (line 16). Therefore, this LP, which has at most p variables and n constraints, is solved at most m times, which is the total number of non-redundant hyperplanes, so that its complexity $\mathcal{O}(mp^2n)$. And since Get_Initial_NR() is $\mathcal{O}(pn)$, then the overall complexity of PRUNE is $\mathcal{O}(p^2mn)$, which is an improvement over the $\mathcal{O}(p^2n^2)$ approach of [52], particularly when many hyperplanes are redundant, i.e. when $m \ll n$.

5.6.2 Sufficient Condition

Applying the exact algorithm at every node in the timing graph can be expensive. Because we are only interested in the non-redundant hyperplanes at the primary outputs, it makes sense to use a *faster* sufficient condition for pruning at the internal nodes, provided that the number of hyperplanes remains under control. Once the primary outputs are reached, the exact algorithm is applied to determine the non-redundant hyperplanes, which correspond to the potentially critical paths in the circuit. We propose a sufficient condition for pruning based on the following idea. Recall that when a set of hyperplanes $\{D_1, \ldots, D_k\}$ prunes a hyperplane D, the following condition, from (5.5), is satisfied:

$$\max(D_1, \ldots, D_k, D) = \max(D_1, \ldots, D_k), \quad \forall X_i \tag{5.19}$$

which can be written as,

$$\max(D_1, \ldots, D_k) \geq D, \quad \forall X_i \tag{5.20}$$

and which can be checked using an LP.

Now assume one can find efficiently a hyperplane D_{lb}, which acts as a lower bound on $\max(D_1, \ldots, D_k)$. Then a sufficient condition for pruning D would be to check if

D_{lb} prunes D. If so, then $\max(D_1, \ldots, D_k)$ would also prune D, because:

$$\max(D_1, \ldots, D_k) \geq D_{lb} \geq D, \quad \forall X_i \tag{5.21}$$

In Chapter 4, we showed how we can determine a lower bound on the maximum of two timing quantities while maintaining the same timing model. We can use the same technique here to find a hyperplane that acts as a lower bound on the maximum of a set of hyperplanes. Let A and B be two delay hyperplanes; let $C = \max(A, B)$ be the maximum of A and B, and assume that either hyperplane can become dominant. We are interested in finding a hyperplane, C_{lb}, that acts as a lower bound on C. Based on Section 4.4.2, C_{lb} will have the following form:

$$C_{lb} = \begin{cases} A & \text{if } |D_{\max}| \gg |D_{\min}| \\ B & \text{if } |D_{\max}| \ll |D_{\min}| \\ \alpha A + (1-\alpha)B & \text{otherwise} \end{cases} \tag{5.22}$$

where $D_{\min}$ and $D_{\max}$ are the extremes of the difference hyperplane $D = A - B$, and α is defined as $\frac{D_{\max}}{D_{\max} - D_{\min}}$. For more detailed information, the reader can refer to Chapter 4 Section 4.4.2. Note that C_{lb} is a hyperplane since it is a linear combination of A and B in either of the three cases.

Although the above analysis is restricted to finding a lower bound on the maximum of two hyperplanes, it can be recursively applied to find a lower bound on the maximum of $n \geq 2$ hyperplanes. It is easy to show that the complexity of doing this for n hyperplanes is $\mathcal{O}(pn)$.

Algorithm description

Algorithm 6 describes our lower bound based sufficient condition for pruning, PRUNE_LB. It takes as input a set $\mathcal{P}$ of n hyperplanes and returns a reduced set $\mathcal{Q} \subseteq \mathcal{P}$. Similarly to PRUNE, PRUNE_LB starts by determining an initial set of non-redundant hyperplanes by calling Get_Initial_NR(), which takes $\mathcal{O}(pn)$ time. Next, a lower bound D_{lb}

on the maximum of all hyperplanes in this set is determined as shown in the previous section. This takes $\mathcal{O}(p^2)$ time since the size of the initial set is $\mathcal{O}(p)$. Then, for every hyperplane D in the remaining set $\mathcal{P}'$, the lower bound is used to test whether D is prunable or not. If so, then D is redundant in $\mathcal{P}$, otherwise, D is added to $\mathcal{Q}$; the cost of this loop is $\mathcal{O}(np)$. Thus, the runtime of PRUNE_LB is $\mathcal{O}(p(p+n))$, which in practice is really $\mathcal{O}(pn)$ because one expects that it's always the case that $p < n$. This represents an improvement over the $\mathcal{O}(pn^2)$ sufficient condition of [52].

Algorithm 6 PRUNE_LB

Input: Set of hyperplanes $\mathcal{P} = \{D_1, \ldots, D_n\}$ of size n;
Output: Reduced set $\mathcal{Q} \subseteq \mathcal{P}$;

```
1: Q = Get_Initial_NR(P);
2: Find a lower bound D_lb on the maximum of all hyperplanes in Q;
3: P' = P \ Q;
4: repeat
5:    Choose an arbitrary D ∈ P' and remove it from P';
6:    Check if D_lb prunes D, i.e. D ≺ D_lb;
7:    if (D ≺ D_lb) then
8:       D is redundant in P {sufficient condition is able to prune};
9:    else
10:      Add D to Q {sufficient condition fails, and D is not pruned};
11:   end if
12: until P' = {}
```

5.7 Results

To verify the accuracy and speed of our pruning techniques, we have tested this approach on a number of circuits from the ISCAS-85 benchmark suite, mapped to a commercial 90nm library. The timing engine was implemented in C++, with an interface to the commercial optimization package MOSEK [56], which was used to solve the LPs in the PRUNE and FEASCHK algorithms. In our tests, the cell library was not characterized for sensitivities to any specific process parameters. Instead, and to allow us to test the approach under some extreme conditions, we have assumed that cell de-

lay depends on a set of 10 arbitrary parameters that are normalized to vary in $[-1, 1]$. In addition, the delay sensitivities to these parameters were generated randomly such that every cell exhibits a total of $\pm 20\%$ deviation in its nominal delay as a result of parameter variability. As to the signs of these sensitivities, they were set at random, again to better test the limits of our approach (as the sensitivity signs are made less correlated, one would expect to see more non-redundant hyperplanes).

Two of the ISCAS-85 circuits were excluded from the analysis for the following reasons. Given our extreme settings of the sensitivities, we have encountered unrealistically large dominant hyperplane counts in c1355 and c6288, and we exclude these circuits from the results. Given more realistic (less extreme) settings of the sensitivities, even these circuits would be manageable. Indeed, for c1355, if the sensitivity signs are all set to be the same, then the analysis completes easily with 174 dominant hyperplanes at the circuit primary outputs. For c6288, if the range of variations is reduced to 5% and the sensitivities are made equal, then again the analysis completes quickly, with only 68 non-redundant planes at the output. In any case, we now present the results, under the extreme settings for sensitivity, on all the other circuits.

We test our approach using the following flow: run PRUNE_LB on every node in the timing graph and then apply PRUNE at the primary output to determine the exact number of non-redundant hyperplanes. For comparison, we also test the equivalent flow from [52]: run PAIRWISE on all nodes and then apply FEASCHK at the primary output. Table 5.1 shows the results, where we report the number of hyperplanes reported at the primary output by the sufficient condition (PRUNE_LB and PAIRWISE), the number of non-redundant hyperplanes found at the primary outputs after exact pruning (PRUNE and FEASCHK), and the total run-times. For example, for circuit c432, the number of hyperplanes propagated by PRUNE_LB to the primary output is 1481, from which only 639 hyperplanes are found to be non-redundant by PRUNE. These 639 hyperplanes make up the piece-wise planar surface of the maximum circuit delay and correspond to the set of all potentially critical paths in the circuit. The overall runtime is found to be 129sec, compared to 338sec for PAIRWISE+FEASCHK, i.e., about $3\times$ speed up.

We can draw several conclusions from Table 5.1. First, it is clear that the proposed

Table 5.1: Summary of hyperplanes at primary output and Run-times for (1) PRUNE_LB + PRUNE and (2) PAIRWISE + FEASCHK

Circuits	**PRUNE_LB Result**	**PRUNE Result**	**Runtime (sec)**	**PAIRWISE Result**	**FEASCHK Result**	**Runtime (sec)**
c432	1481	639	129	1114	639	338
c499	1918	520	156	1539	520	230
c880	16	12	0.62	16	12	0.57
c1908	63	37	2.09	46	37	2.24
c2670	556	126	18.3	270	126	18.9
c3540	60	30	2.80	56	30	3.55
c5315	18	12	2.22	12	12	2.12
c7552	14	9	3.67	11	9	4.57

approach is practical and offers the hope that the timing of the circuit across the *whole* PVT space *can* be provided for use by downstream tools. Secondly, note that what determines whether a circuit is *harder* or *easier* to analyze is not the number of gates in the design, but the number of hyperplanes that are non-redundant, i.e., the number of potentially critical paths; this depends on circuit topology. While c7552 is much larger than c432, we find that the latter takes more time to analyze as the resulting number of non-redundant hyperplanes is much larger. Notice that even under the extreme sensitivity settings, described above, most circuits have a reasonably small number of non-redundant hyperplanes at their outputs, which is in-line with the observations in [52] where the number of potentially critical paths was shown to be manageable for most circuits. For easy circuits, the performance of our method is comparable to [52], while for harder circuits, such as c432 and c499, our approach becomes faster.

Another metric to look at is the quality of pruning by the sufficient condition PRUNE_LB, and a comparison of that to PAIRWISE of [52]. As explained earlier, both algorithms are applied on every node in the test circuits until the primary outputs are reached. Table 5.1 shows a comparison of the two pruning techniques in terms of how many hyperplanes they report at the primary outputs, as shown in the second and fifth column. For example, PRUNE_LB and PAIRWISE report 1481 and 1114 non-redundant hyperplanes at the primary output of circuit c432, respectively, while the exact number of non-redundant hyperplanes is 639. Notice that the PAIRWISE

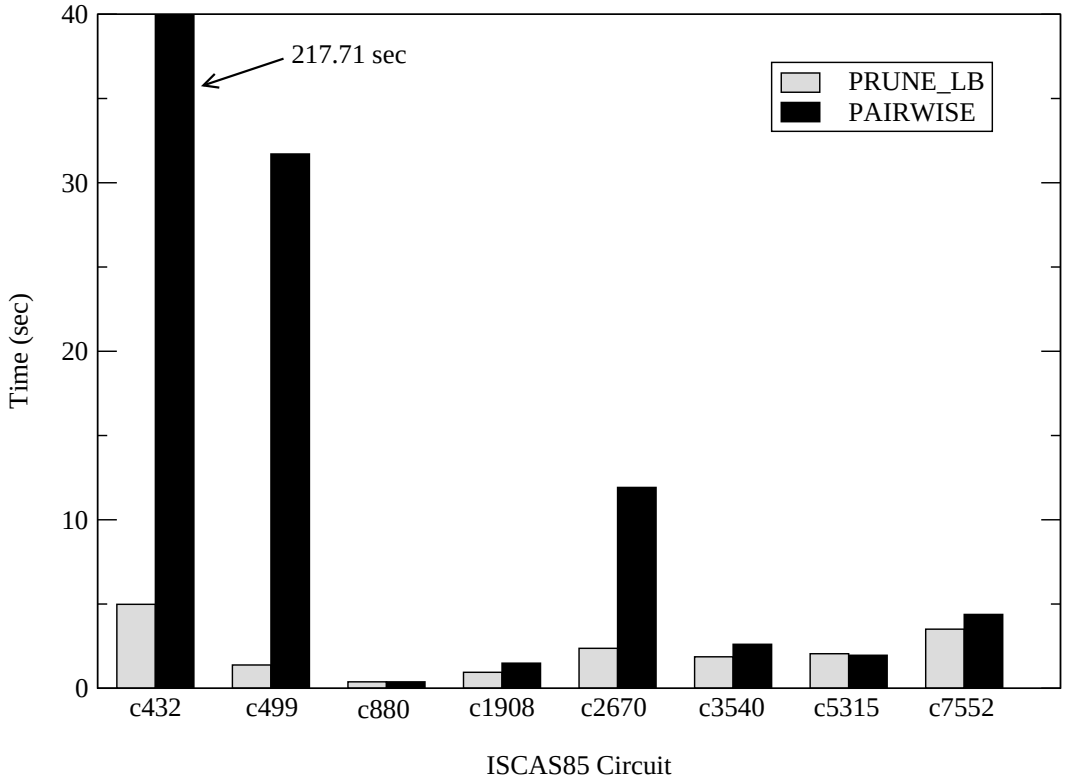


Figure 5.4: Comparison of PRUNE_LB and PAIRWISE algorithms

algorithm typically yields results that are closer to the exact solution. However this advantage of PAIRWISE comes at a runtime disadvantage when compared to PRUNE_LB, as can be seen in Fig. 5.4. This is particularly true for c432, c499, and c2670, which are *harder* circuits compared to the rest of the test circuits. For example, although the number of hyperplanes reported by PRUNE_LB for circuit c432 is slightly larger than that reported by PAIRWISE, the actual speed up is about 44×, which can be calculated from Fig. 5.4. Also notice that the performance of the two methods, in terms of both runtime and quality of pruning, is comparable for all other *easy* cases where the number of hyperplanes seen at the output is smaller.

Finally, we draw the reader's attention to circuit c2670 in Table 5.1, where the total run-time for both flows is comparable. This is due to the fact that PRUNE_LB predicted 556 hyperplanes to be reduced by PRUNE, whereas PAIRWISE predicted

only 270 hyperplanes to be reduced by FEASCHK. In a sense, the speed up achieved by our sufficient condition, shown in Fig 5.4, is lost since our exact algorithm had to reduce a larger set of hyperplanes. In this case, an alternative approach may be to apply PRUNE_LB on every node, then to apply PAIRWISE for additional pruning at the primary output before calling PRUNE to determine the exact number of non-redundant hyperplanes.

5.8 Summary

In this chapter, we have proposed an efficient block-based parameterized static timing analysis technique that can accurately capture circuit delay at every point in the parameter space, by reporting all paths that can become critical. Using efficient pruning algorithms, only those *potentially critical* paths are carried forward, while all other paths are pruned during propagation. After giving a formal definition of this problem, we have proposed (1) an exact algorithm for pruning, and (2) a fast sufficient condition for pruning, that improve on the state of the art, both in terms of theoretical computational complexity and in terms of run time on various test circuits. The work in this chapter has also established a link between the pruning problem in the parameterized timing domain, and two standard problems in computational geometry.

6 Robustness Metrics in Parameterized Static Timing Analysis

6.1 Introduction

In this chapter, we propose a new post-variational analysis metric that can be used to quantify the (timing) *robustness* of designs to parameter variations. In addition to helping designers diagnose *if* and *when* different nodes can fail, this metric can guide optimization and can give insights on *what to fix*, by identifying nodes with small robustness values and proceeding to fix those nodes first. Inspired by the rich literature on *design centering, tolerancing, and tuning* (DCTT), we use *distance* as a measure for robustness. Our analysis thus determines the minimum distance from the nominal point in the parameter space to any timing violation, and works under the assumption that parameters are specified as ranges rather than statistical distributions. We demonstrate the usefulness of this distance-based robustness metric on circuit blocks extracted from a commercial 45nm microprocessor from Intel.

6.2 Background

Many variation-aware timing analysis techniques have focused on assessing the impact of parameter variations on timing by using variational models where delay is assumed to depend on a number of PVT parameters. These techniques can be collectively described under the heading of *parameterized static timing analysis* (PSTA). Statistical static timing analysis (SSTA) is one example of PSTA, in which parameters are modeled as random variables with known distributions and correlations [27, 28], and timing yield

is estimated from the corresponding distribution of circuit delay. In practice, however, the statistical distributions and correlations of some PVT parameters may be unknown or unavailable. Also, some parameters, such as supply voltage or temperature, are not truly random and are better modeled as simply unknown or uncertain variables. Thus, some alternative PSTA techniques have also been proposed, such as multi-corner static timing analysis (MCSTA). MCSTA models parameters as uncertain variables within given or known bounds, and attempts to verify circuit timing at all corners in a single timing run [5, 51]. Although the circuit delay is captured accurately at the worst case corner, the same cannot be said about other points in the parameter space. Note that the PSTA framework proposed in Chapter 4 is applicable to both SSTA and MCSTA.

Recently, some PSTA techniques [6,52] have addressed this limitation by proposing to capture, in a single timing run, circuit delay *exactly* at all points in the PVT space. This can be done by propagating in the timing graph all the paths that can become dominant (critical) at any setting of the PVT parameters, and pruning all other redundant paths. The technique proposed in Chapter 5 falls under this latter PSTA class. In any case, the end result of all PSTA techniques is to provide designers with parameterized timing quantities (arrival times and/or slacks) which are expressed as functions of PVT parameters.

6.3 Overview

While a large body of research has focused on the *analysis* step, very few proposals have presented clear answers for how to interpret and utilize the results of PSTA in design. The crux of the matter, and this is what motivated PSTA in the first place, is that the goal is to produce a *safe* design, in the sense that it must be *robust* to variations. To do that, an essential requirement is to be able to *measure* the safety or robustness of a given design, i.e., its *susceptibility to timing failure* due to variations. But how does one formally define robustness? How does one quantify the susceptibility to failure and determine how far a nominally safe design is from the "edge of the cliff"? Finally, what does one need to do to improve the robustness of a given design? One way to quantify robustness, which is used in SSTA, is to use the notion of

yield to assess the safety of timing quantities. Indeed, timing yield is one measure of robustness - it provides designers with the probability of meeting/violating the timing constraints. However, yield analysis via SSTA requires that all parameters be modeled as random variables with known distributions and correlations. As we noted earlier, the distributions and correlations of PVT parameters may not always be available or fully specified. Hence, for those PSTA techniques where parameters are either *i)* modeled as uncertain (non-random) variables in specified ranges, or *ii)* where distributions are unknown or unavailable, we need to define some other metric that can be used to assess the robustness of timing quantities resulting from these methods. One possible way of doing that is to use the *volume* of the feasible region as a measure of robustness. As part of their work on yield prediction, the authors of [31] have proposed two techniques for approximating the volume of the feasible region, the *parallelepiped method* and the *ellipsoid method*. However, it was found that, while both techniques are accurate, they may not scale well with the number of PVT parameters or the number of paths. Thus, this approach can be costly on large designs.

In this chapter, we hope to answer some of the above open questions as we present a new metric that can be used to *quantify* the (timing) robustness of a design to variations in the case where parameters are given as ranges rather than fully specified distributions. Our method *processes* the parameterized timing quantities resulting from PSTA so as to extract useful information about the susceptibility to timing failures. We will define robustness as the minimum *distance*, from the nominal point in the PVT parameter space, to any other point where a timing violation occurs. Such distance-based metrics have been used in a different context in the realm of design centering, tolerancing, and tuning (DCTT) [57, 58]. In DCTT, optimal nominal values for some *designable parameters* are selected so that the *distance* from the nominal point (center point of the design) to the boundary of the *acceptability region* is maximized in the hope of maximizing the process yield. Traditional DCTT operates in the space of design parameters, whereas our distance metric is measured in the PVT parameter space. It is also important to note that design centering has only been traditionally applied to small, typically analog, circuits, not to large digital integrated circuits; this is because it relies on expensive statistical simulations to determine the acceptability

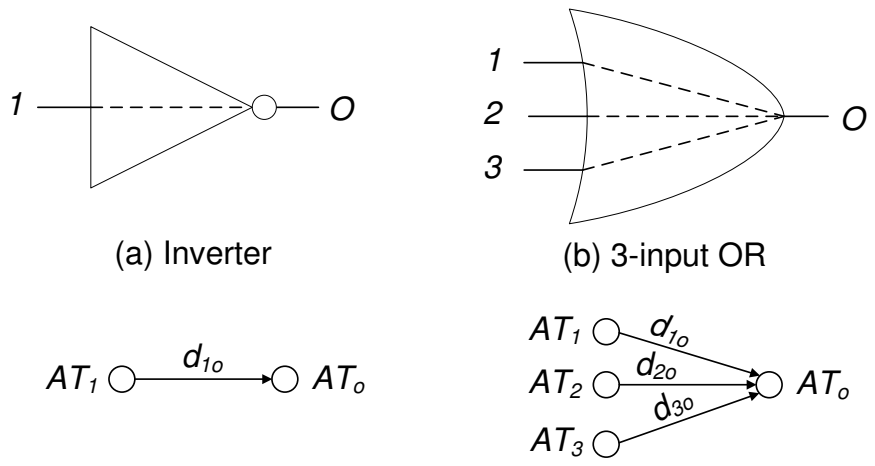


Figure 6.1: Timing graphs for (a) Inverter, and (b) 3-input OR gate

region of the design parameters, not to mention that the number of those parameters increases with circuit size. We will see that, in our use of a distance metric in PVT space, these complications do not arise and the resulting approach is computationally efficient.

Thus, the novelty of the work in this chapter is in its proposed application, where we extend the use of such distance metrics to the timing verification of large logic circuits, which, to our knowledge, was never done before. Using this new distance-based robustness metric, designers can not only diagnose if and when different nodes can fail timing, but also get insight on *what to fix*. In fact, our robustness metric can be used to evaluate design quality and to guide optimization by ranking different nodes according to their robustness, thus identifying the least robust nodes and proceeding to fix those nodes first. We also show that our robustness analysis can handle parameterized timing quantities resulting from either exact [6,52] (Chapter 5) or bounded/approximate PSTA techniques [5, 51] (Chapter 4). Also, our metric is computed efficiently and scales well (linear complexity) with the number of PVT parameters and the number of paths.

6.4 Preliminaries

We will review the static timing analysis (STA) terminology and describe how STA is extended to handle PVT variations as part of parameterized static timing analysis (PSTA).

6.4.1 Nominal Static Timing Analysis

In static timing analysis, the circuit under study is represented as a timing graph by creating a graph node for every electrical net in the circuit (primary input, output, or internal node) and a graph edge for every *timing arc* (logic gate input/output pair). The weight of every edge corresponds to the delay value from that input pin to the output pin. The *arrival time* at the output of a gate is computed first by *adding* the input arrival times to their corresponding timing arc delays (edge weights), and then taking the *max* over the result of those additions. This procedure is repeated while topologically traversing the timing graph and computing the arrival times at every node.

Fig. 6.1 shows the timing graphs for two simple logic gates, an inverter and a 3-input OR gate. The edge weight, d_{io}, corresponds to the arc delay from input i to the output. Since the inverter has one input, the arrival time (AT) at its output is simply:

$$AT_o = AT_1 + d_{1o} \tag{6.1}$$

For the OR gate, its output arrival time is the maximum of the sum of its three input arrival times and their corresponding arc delays:

$$AT_o = \max_{i=1}^{3}(AT_i + d_{io}) \tag{6.2}$$

While arrival times are computed during forward propagation in the timing graph, *required times* (RT), which are defined as the latest acceptable arrival times that would not violate the timing constraints, must be computed based on the information downstream, and thus require a backward propagation from the primary outputs. For a node to pass timing, its arrival time must not exceed its required time. The concept of

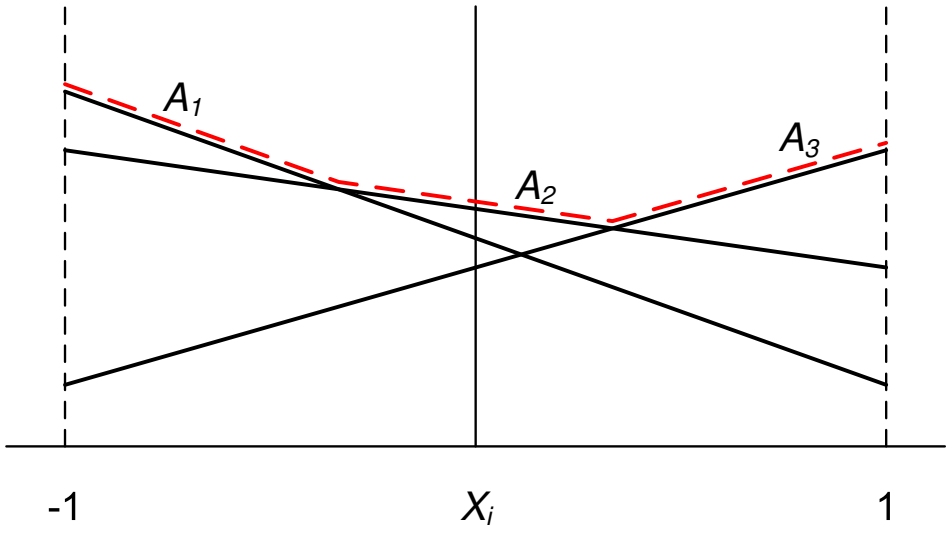


Figure 6.2: Parameterized arrival time

slack, which is the difference between the required time and the arrival time at a node, is generally used as a measure of how close a node is to violating its timing constraint. In general, we require the slack S to be positive:

$$S = RT - AT \geq 0 \tag{6.3}$$

In nominal static timing analysis, all the above timing quantities (AT, RT, S) are computed under the assumption that process and environmental parameters - and consequently timing arc delays, which depend on these parameters - are fixed, typically at either their *nominal* or corner values. However, due to the increasing significance of variability, parameterized static timing analysis (PSTA) techniques have emerged, with the goal of handling parameter variations as part of the timing analysis step.

6.4.2 Parameterized Static Timing Analysis

A key component of any PSTA technique is the *delay model* that captures the dependence of gate/interconnect delays on the underlying process and environmental parameters. First-order linear delay models are often used in the literature, and they generally capture well this dependence. Under such a linear model, the delay D of a timing arc is expressed as:

$$D = d_o + \sum_{i=1}^{p} d_i X_i \tag{6.4}$$

where d_o is the nominal delay value and d_i is the (first-order measure of) sensitivity to parameter X_i. Note that X_i can represent the variation of any parameter, such as channel length, supply voltage, or temperature. Also note that d_o and the d_i's are determined during library characterization.

Another component of PSTA is the model for the PVT parameters. As noted above, while SSTA techniques [27, 28] use random variables with known probability distributions (e.g. Gaussian) to model the parameters, other PSTA techniques, such as [51, 52] and the techniques that were presented in Chapters 4 and 5 model them as *uncertain* variables that are specified in given ranges. We will adopt this more general model, based on uncertain parameters, because the distributions and correlations of some PVT parameters may be unknown or unavailable in practice. For simplicity, and without loss of generality, we will assume that the variation range of every uncertain parameter X_i is normalized to $[-1, 1]$. Following standard terminology, the linear model in (6.4) will be referred to as a delay *hyperplane*.

Because path delay is the sum of gate and interconnect delay hyperplanes on that path, it is also modeled as a hyperplane. However, arrival times are not simply hyperplanes because, when different paths converge at a node, a (nonlinear) *max* operation must be performed to determine the arrival time at that node. For example, consider Fig. 6.2, where A_1, A_2, and A_3 represent path delay hyperplanes. For purpose of illustration, a single parameter X_i is considered, so that the hyperplanes are simply straight line segments. The dashed piecewise linear function (in general piecewise planar) resulting from the max operation corresponds to the exact representation of the arrival

time as:

$$AT = \max(A_1, A_2, A_3) \tag{6.5}$$

where the A_j's have the form in (6.4).

A similar piecewise planar function representing the *min* operation arises when one is dealing with parameterized slacks. For example, consider the circuit in Fig. 6.3 having two primary outputs at registers R_1 and R_2. The arrival time AT_1 at the data input of register R_1 is represented by a piecewise planar surface, say $AT_1 = \max(A_1, A_2, A_3)$. Note however, that the required time RT_1 at the data input of R_1 is not a max surface but a (single) hyperplane. This is because we are assuming that the arrival time at the clock input, r_1, is the delay hyperplane corresponding to a clock tree path. Hence, the parameterized slack S_1, at the input of register R_1 will be given as a minimum of a set of hyperplanes, as follows:

$$\begin{aligned} S_1 &= RT_1 - AT_1 \\ &= RT_1 - \max(A_1, A_2, A_3) \\ &= RT_1 + \min(-A_1, -A_2, -A_3) \\ &= \min\big((RT_1 - A_1), (RT_1 - A_2), (RT_1 - A_3)\big) \end{aligned} \tag{6.6}$$

where we have used the fact that $\max(a, b) = -\min(-a, -b)$. A similar reasoning follows for the parameterized slack S_2 at the input of register R_2. As a result, the minimum slack S for this circuit, being the minimum of all parameterized slacks at registers inputs will also be represented as a piecewise planar surface defined by the min operation above.

While some PSTA techniques [6, 52] capture exactly the piecewise planar surfaces representing the nonlinear max and min operations, other techniques [5, 51] approximate and/or bound those operations using a single hyperplane. In general, both exact and approximate PSTA techniques result in timing quantities (arrival times/slacks)

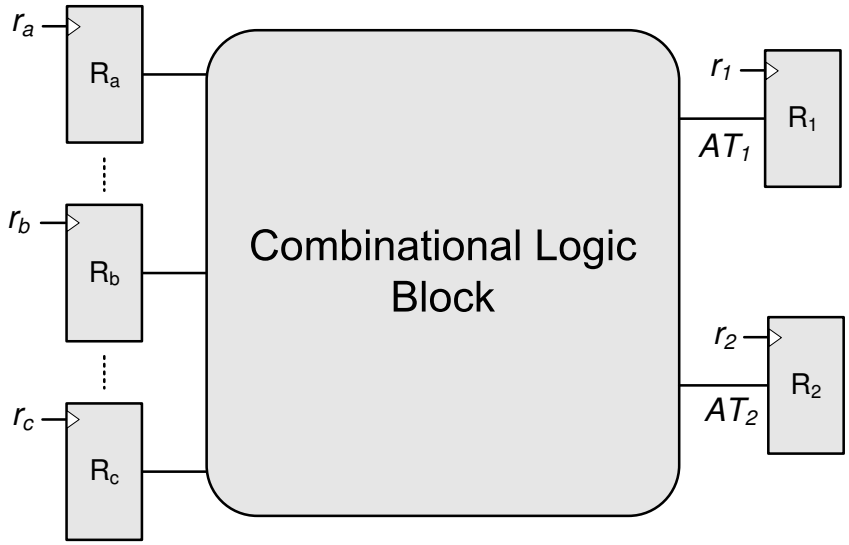


Figure 6.3: Slack computation

being parameterized as functions of the PVT parameters, as follows:

$$AT(X) = \begin{cases} a_o + \sum_{i=1}^{p} a_i X_i, & \text{approx. PSTA} \\ \max_{j=1}^{n}(a_{oj} + \sum_{i=1}^{p} a_{ij} X_i), & \text{exact PSTA} \end{cases} \tag{6.7}$$

$$S(X) = \begin{cases} s_o + \sum_{i=1}^{p} s_i X_i, & \text{approx. PSTA} \\ \min_{j=1}^{n}(s_{oj} + \sum_{i=1}^{p} s_{ij} X_i), & \text{exact PSTA} \end{cases} \tag{6.8}$$

where n is the number of hyperplanes that define the piecewise planar surfaces of the max and min operations, and $-1 \leq X_i \leq 1$ for all i.

6.5 Robustness Analysis

Although parameterized expressions of timing quantities resulting from PSTA are very useful, they do not *directly* provide a metric of robustness of these timing quantities to variations. Some further *processing* of these expressions is required, to extract this information. We are interested in transforming complex expressions of PVT parameters, such as (6.7) and (6.8), into a measurable or quantifiable *robustness metric*. In this section, we first define such a metric as a measure of how close a node is to violating its timing constraint. We also compare robustness and sensitivity and highlight the subtle difference between the two notions. Finally, we present our mathematical formulation for robustness analysis and describe our algorithm.

6.5.1 From Sensitivity to Robustness

Suppose that one is comparing two design realizations of the same circuit, for which PSTA has provided the two different parameterized slacks at some node, $S_1(X)$ and $S_2(X)$. Alternatively, suppose that $S_1(X)$ and $S_2(X)$ are the parameterized slacks at two nodes of the same design. Either way, we are interested in comparing the robustness of $S_1(X)$ and $S_2(X)$. If, at the nominal point $X = 0$, it turns out that $S_1(0) \geq S_2(0) \geq 0$, then one might be inclined to assume that S_1 is more robust than S_2 because it is larger, and thus variations would seem to affect it less adversely. However, this is not always true, as it may turn out that S_1 is more *sensitive* to variations than S_2, and consequently more prone to failure. Hence, sensitivity is an important measure that is closely tied to robustness.

As an example, Fig. 6.4 shows a comparison of two parameterized slacks, $S_1(X)$ and $S_2(X)$, where we have assumed a single parameter X, varying in $[0, X_{\max}]$. Fig. 6.4a shows a case where the slack with the larger nominal value turns out to be less robust. In fact, although $S_1(0) \geq S_2(0)$, $S_1(X)$ fails "before" $S_2(X)$, because the value of X for which $S_1(X)$ becomes zero, d_1, is smaller than d_2, the value of X for which $S_2(X)$ becomes zero. In this case, the more sensitive slack turns out to be the one that is less robust. On the other hand, Fig. 6.4b shows a case where the opposite happens: even though $S_1(X)$ is more sensitive than $S_2(X)$, it is actually more robust. This is

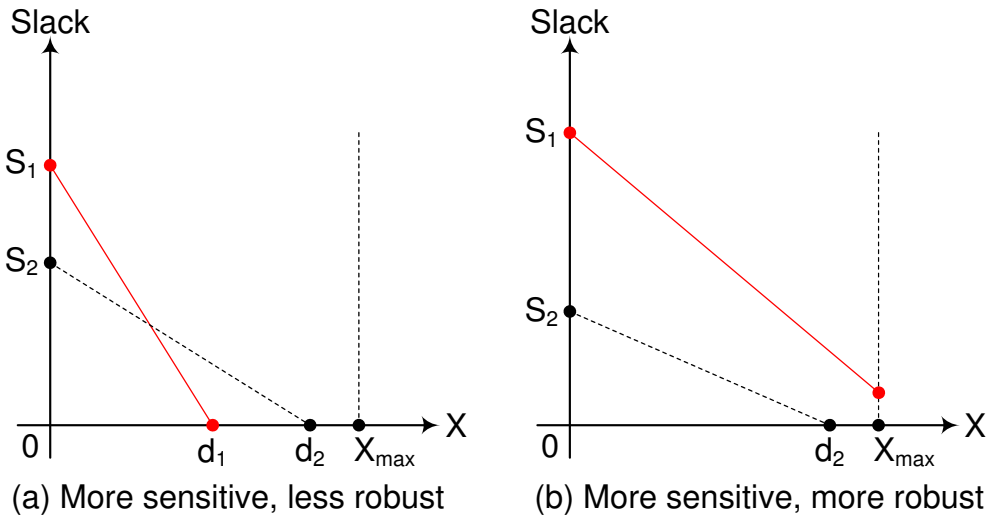


Figure 6.4: Sensitivity and robustness

because $S_1(X)$ does not fail in $[0, X_{\max}]$, while $S_2(X)$ fails for $X = d_2$. Therefore, while robustness is related to the *susceptibility of a node to violating timing*, sensitivity is related to the magnitude of timing deviation, per unit parameter variation, irrespective of whether or not timing is actually violated. Thus, a node having the larger sensitivity yet not failing anywhere in the parameter space is "more robust" than a node having smaller sensitivity, yet failing somewhere in the parameter space.

To fully capture the notion of design robustness, we need to somehow make use of both the nominal values *and* the sensitivities, in relation to the threshold where timing failure occurs.

6.5.2 Quantifying Robustness

For the simplified scenario shown in Fig 6.4, one can define robustness as simply the value of X for which timing is violated. This would be a good metric to use because it is quantitative; it would allow one to conclude, for example, that S_2 is more robust than S_1 whenever $d_2 > d_1$. However, in the general case where several parameters are

varying, and where parameterized timing quantities are piecewise planar surfaces, each with a different set of sensitivities, things can get more complicated. For one thing, finding a setting for the parameter vector X where timing is violated is not as simple as finding the x-intercept in Fig. 6.4, and requires a search in a higher-dimensional space. Furthermore, there could be be many such settings, making it hard to judge which one to use as a measure of robustness.

Distance-based Metric

We propose that a distance-based metric is a good choice for robustness analysis. Specifically, we define the robustness metric as the *minimum distance from the nominal point* in the PVT parameter space to any point where a timing violation occurs. Distance can easily abstract the large dimensionality of the problem by presenting a simple quantifiable measure that can be computed with little effort. By measuring distance from the nominal point, we are implicitly assuming that the nominal design is feasible, i.e., it meets the timing constraints. Thus, the minimum distance to any timing violation reflects the smallest (magnitude) deviation from the nominal point that would "break" the design.

In fact, such distance-based metrics have been used in the past as part of design centering for yield maximization [57–59]. The goal of design centering is to determine the optimal nominal settings of the *design parameters*, which are constrained to satisfy performance specifications in the presence of tolerances. These nominal values (which define the center point of the design) are optimized such that the *distance from the design center to the boundary* of the acceptability region is maximized, in the hope of maximizing yield. Distance is therefore used in design centering as a measure of safety, since the more "distant" the center is from the boundary, the higher is the expected yield. In our case, where we work with PVT parameters, rather than design parameters, we do not try to recenter our design so that distance is maximized. Instead, we only use distance as a measure of how robust a timing quantity (or a design) is in the face of PVT variations. In fact, in our problem, the nominal PVT point is fixed, as well as the ranges of variation, whereas in design centering, the space under study is that of the design parameters, whose nominal values, as well as tolerances (ranges) are to be

determined.

6.5.3 Mathematical Formulation

In this section, we present the mathematical formulation of our robustness analysis by means of *normed distances* to the boundary of the region where timing is met.

The Feasible Space

In general, given a parameterized timing quantity $T(X)$, we define its robustness, r, as the minimum distance (using some vector *norm*) from the nominal PVT point, $X = 0$, to any point in the PVT space where $T(X)$ violates timing. Recall that we have assumed in Section 6.4.2 that PSTA has already been used to analyze the design, and that parameterized slacks $S(X)$ and/or arrival times $AT(X)$ are available. Each timing quantity is either a hyperplane or a collection of n hyperplanes defining a piecewise planar surface, as shown in (6.7) and (6.8). In the analysis that follows, we assume that one is dealing with parameterized slacks $S(X)$, however, the same analysis can be easily applied to the case of parameterized arrival times. Recall that $S(X)$ is defined in (6.8) as the minimum of n hyperplanes:

$$S(X) = \min\left(S_1(X), \ldots, S_n(X)\right) \tag{6.9}$$

where $n \geq 1$ (for approximate PSTA, $n = 1$), and where:

$$S_j(X) = s_{oj} + \sum_{i=1}^{p} s_{ij} X_i, \quad j = 1, \ldots, n \tag{6.10}$$

We also assume that this set of n hyperplanes has already been reduced using pruning techniques, such as the ones proposed in [52] or in Chapter 5, so that every hyperplane $S_j(X)$ can become the minimum, *i.e.*, $S(X) = S_j(X)$, for some X.

With the above expressions for slack, the space where timing is satisfied, $\mathcal{C}$, is defined by $S(X) \geq 0$, which can be expressed as a convex polytope (intersection of linear constraints) by replacing $S(X)$ by its expression in (6.9), as the minimum of n

hyperplanes:

$$\mathcal{C} = \left\{ X \;\middle|\; S_j(X) = s_{oj} + \sum_{i=1}^{p} s_{ij} X_i \geq 0, \quad j = 1, \ldots, n \right\} \tag{6.11}$$

Also, we assume that the ranges of X_i's are normalized to $[-1, 1]$, so that the parameter space, $\mathcal{D}$, is defined as the following p-cube:

$$\mathcal{D} = \left\{ X \;\middle|\; -1 \leq X_i \leq 1, \quad i = 1, \ldots, p \right\} \tag{6.12}$$

Therefore, the intersection of $\mathcal{C}$ with $\mathcal{D}$ corresponds to all X in $\mathcal{D}$ for which timing is met, *i.e.,* $S(X) \geq 0$. We refer to this $\mathcal{C} \cap \mathcal{D}$ as the *feasible region.* As mentioned earlier, we assume that the nominal design is feasible (meets timing), so that the nominal point $X = 0$ is inside the feasible region. If one starts at the nominal point and moves outward, then two cases may arise. Either the boundary of $\mathcal{C}$ is encountered and crossed first at which point timing is violated (*i.e.,* $S_j(X) < 0$ for one or more j), or the boundary of $\mathcal{D}$ is encountered and crossed first, at which point the range is exceeded (*i.e.,* $|X_j| > 1$ for one or more j). For robustness analysis, we are interested in the minimum distance from the nominal point to any point within the range $\mathcal{D}$ at which timing is violated. Therefore, we are interested in the minimum distance to the boundary of $\mathcal{C}$, obviously provided that $S(X)$ fails somewhere inside $\mathcal{D}$.

Normed distance to a hyperplane

We want the minimum distance to the boundary of the convex polytope $\mathcal{C}$ defined by the set of n linear constraints in (6.11), $S_j(X) \geq 0$, $\forall j$. The boundary corresponds to n hyperplanes, h_j, $j = 1, \ldots, n$, where:

$$h_j \;:\; S_j(X) = 0 \tag{6.13}$$

$$:\; \sum_{i=1}^{p} s_{ij} X_i = -s_{oj} \tag{6.14}$$

$$:\; a_j^T X = b_j \tag{6.15}$$

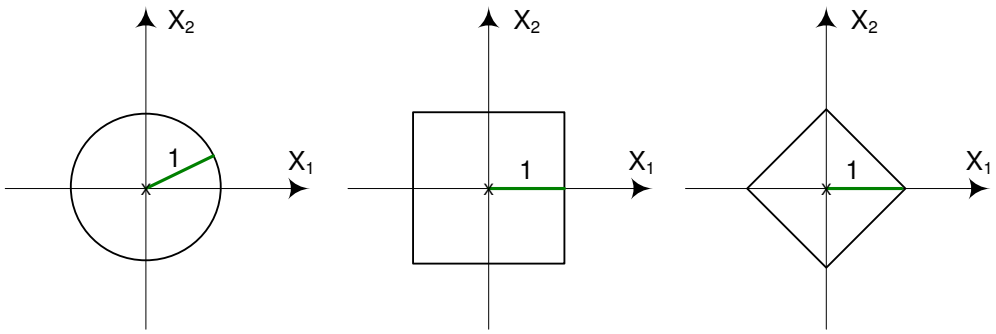


Figure 6.5: Unit ball in different norms

where a_j is the vector $a_j^T \triangleq [s_{1j}\ s_{2j}\ \cdots\ s_{pj}]$ and $b_j \triangleq -s_{oj}$.

Let $d_n(X_o, h_j)$ be the distance, in an arbitrary vector norm $\|X\|$, from a point X_o to the hyperplane h_j. This so-called *normed distance* can be expressed in terms of the norm's unit ball, $\mathcal{B}_n = \{X \mid \|X\| \leq 1\}$, as follows [60]:

$$d_n(X_o, h_j) = \min\{|\lambda| \;\big|\; (X_o + \lambda\mathcal{B}_n) \cap h_j \neq \emptyset\} \tag{6.16}$$

Therefore, to determine the normed distance from X_o to h_j, one needs to dilate the unit ball by λ around X_o until it *touches* the hyperplane. Fig. 6.5 shows different unit balls $\mathcal{B}_2$, $\mathcal{B}_\infty$, and $\mathcal{B}_1$ (in 2-D) for the L_2-, L_∞-, and L_1-norms, respectively, where:

$$L_2\text{-norm} \;=\; \|X\|_2 = \sqrt{\sum_{i=1}^{p} X_i^2} \tag{6.17}$$

$$L_\infty\text{-norm} \;=\; \|X\|_\infty = \max_{i=1}^{p} |X_i| \tag{6.18}$$

$$L_1\text{-norm} \;=\; \|X\|_1 = \sum_{i=1}^{p} |X_i| \tag{6.19}$$

The normed distance in (6.16) can also be expressed in terms of the dual norm

$\|X\|^\star$, as follows [59, 60]:

$$d_n(X_o, h_j) = \frac{|a_j^T X_o - b_j|}{\|a_j\|^\star} \tag{6.20}$$

where the dual norm is $\|u\|^\star = \sup\{u^T v \mid \|v\| \leq 1\}$. For the L_p-norm $\|X\|_p$ (including L_1, L_2, and L_∞ norms), defined by:

$$\|X\|_p = \left(\sum_i |X_i|^p \right)^{\frac{1}{p}}, \tag{6.21}$$

it is easy to determine its corresponding dual norm, defined as the L_q-norm $\|X\|_q$, where q and p are related in the following way:

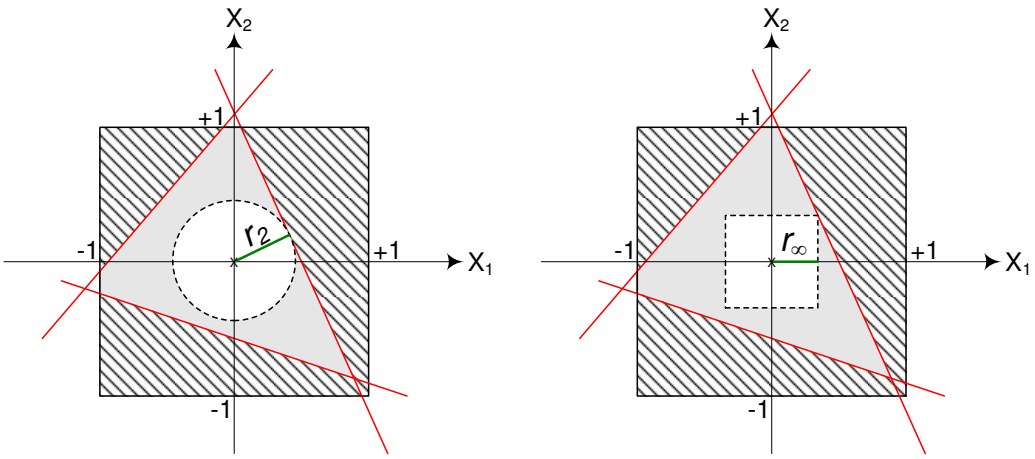
$$\frac{1}{p} + \frac{1}{q} = 1 \tag{6.22}$$

From the above equation, note that the dual norm of the L_2-norm is itself ($p = q = 2$), and that the L_1 and L_∞ norms are duals of one another ($p = 1$ and $q = \infty$).

Using the very simple expression for normed distance in (6.20), we can easily determine the robustness metric by finding the distance (in any L_p-norm) from the nominal PVT point $X = 0$ to every hyperplane h_j that defines the boundary of the region where timing is met, and record the smallest distance as the robustness, r, of the parameterized slack, $S(X)$. This will be shown next.

Algorithm and Illustration

A description of the algorithm, FindRobustness, is shown in Algorithm 7. It takes as input a parameterized slack, $S(X)$, and returns its robustness, r, as defined above. The algorithm starts by checking two corner cases. First, the nominal slack is checked, at the nominal point $X = 0$ (line 1). If $S(0) \leq 0$, then the nominal slack violates timing. In that case, $X = 0$ is not feasible, and we simply set $r = 0$. Nodes with $r = 0$ are the least robust because they violate timing even before considering parameter variations. The second corner case to check is whether the minimum value (over X) of $S(X)$ is positive (line 3). Since $S(X)$ is the minimum of $S_j(X)$'s and $-1 \leq X_i \leq 1$, this can be



(a) L_2-normed distance (b) L_∞ -normed distance

Figure 6.6: Robustness analysis

easily checked as follows:

$$\min_{X} \{S(X)\} = \min_{j=1}^{n} \left\{ s_{oj} - \sum_{i=1}^{p} |s_{ij}| \right\} \tag{6.23}$$

If the above expression is positive, it simply means that $S(X)$ does not fail anywhere in the parameter space and, in that case, we set $r = \infty$. Nodes with $r = \infty$ are the most robust since they are not prone to timing violations anywhere in $\mathcal{D}$.

If both these corner conditions are not met, then $S(X)$ will fail somewhere in the parameter space $\mathcal{D}$. In that case (line 5), we first set $r = \infty$ (or some upper bound value). Then, we compute the normed distance, r_j, from the nominal point $X = 0$ to the (boundary) hyperplane h_j defined by $S_j(X) = 0$. To do that, we use the formula in (6.20) for some choice of L_p-norm and its corresponding dual L_q-norm. Typically, L_2-normed distances are mostly prevalent in the literature on design centering, with some use of the L_∞-norm (and its corresponding L_1 dual norm). This is done for all boundary hyperplanes h_j's, and the smallest value of r_j is recorded as the robustness of $S(X)$ (lines 10-11).

Algorithm 7 FindRobustness

Input: $S(X) = \min(S_1(X), \ldots, S_n(X))$
where $S_j(X) = s_{oj} + \sum_{i=1}^{p} s_{ij} X_i$
Output: $r \in \mathbb{R}$

if $(S(0) \leq 0)$ **then**
  return $r = 0$
else if $(\min_X \{S(X)\} > 0)$ **then**
  return $r = \infty$
else
  $r = \infty$
  for $(j = 1, \ldots, n)$ **do**
    $S_j(X) = 0 \rightarrow h_j : s_j^T X = -s_{oj}$
    $r_j = d_n(0, h_j) = \frac{|s_{oj}|}{\|s_j\|^\star}$
    if $(r_j < r)$ **then**
      $r = r_j$
    end if
  end for
  return r
end if

Fig. 6.6 is a simple 2-D example depicting graphically how robustness analysis works, when both the L_2 and L_∞ norms are used. Note that the parameter space is defined by the square region where the two parameters are restricted to vary in $-1 \leq X_1, X_2 \leq 1$, and the feasible space where timing is met is defined by the grey triangle-like region that contains the nominal PVT point $X = 0$. The striped region outside the boundary of the feasible space is the region where timing is violated. Fig. 6.6a shows the robustness, r_2, computed in L_2-norm. This is equivalent to inflating an L_2-normed unit ball (disk) around $X = 0$ as shown, until it touches one of the hyperplanes at the boundary. Similarly, a L_∞-normed unit ball (square) is inflated around $X = 0$ in Fig. 6.6b to obtain the robustness, r_∞, in L_∞-norm.

6.5.4 Unbiased vs Biased Analysis

The robustness analysis presented so far has assumed that the PVT parameters are uncertain variables given in ranges and having equal spreads (assumed to be [-1,1]). As a result, the analysis implicitly gives equal weights to all possible directions in the space. This option would be the most appropriate if absolutely no additional information is known about parameter variations and their interactions. We will refer to this type of analysis as *unbiased analysis*. On the other hand, if additional or partial information is available, whether it is from historical data or from process experience, then one could make use of this information so as to *bias* the robustness metric computation. For example, if parameters have different spreads, then one can use a scaling (diagonal) matrix to scale the norm itself and compute the distance-based metric in the new scaled norm. Also, if it is observed that certain parameters exhibit some *covariance*, that is, if certain parameters are likely to vary in the same (or opposite) direction, then one can rotate the norm by a nondiagonal scaling matrix. It is shown in [59] that, given any (diagonal or nondiagonal) scaling matrix W, the scaled norm is given by $\|X\|_W = \|W^{-1}X\|$, and its scaled dual norm is given by $\|X\|_W^\star = \left\|W^T X\right\|$. Both unbiased and biased analyses are depicted in Fig. 6.7, where in (b) we have scaled the unbiased L_2-norm by reducing the range of X_2, and in (c) have rotated the norm by 45°, emphasizing the fact that X_1 and X_2 are likely to vary in the same direction.

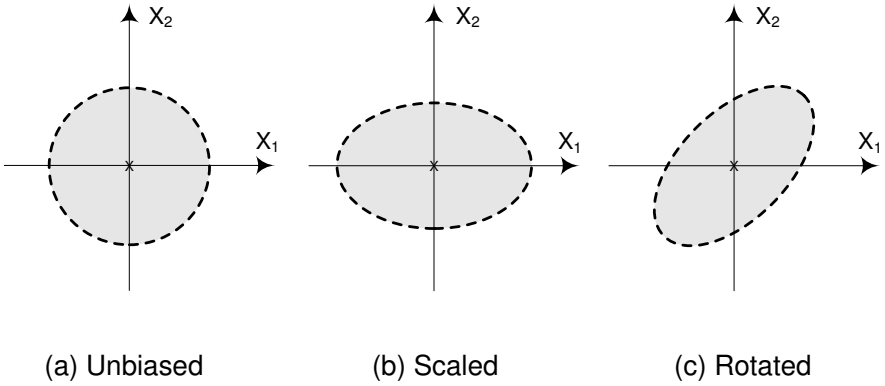


Figure 6.7: Biasing the norm

6.6 Results

In this section, we present the simulation results that were obtained on a 45nm commercial microprocessor design. Two parameterized static timing analysis flows were implemented in `C++` on top of an STA timing engine. The first is an exact PSTA flow that parameterizes timing quantities in the form of piecewise planar surfaces (collection of hyperplanes) defined by the max or min operations described in (6.7) and (6.8). The exact PSTA implementation is based on the pruning techniques of [6, 52]. The second flow is an implementation of the approximate PSTA technique of [5], which parameterizes every timing quantity as a single hyperplane. For both flows, we have considered global variations in four different parameter types, namely supply voltage (V_{dd}), Miller Coupling Factor (MCF), and channel length for both NMOS and PMOS devices (L_n and L_p). In addition, L_n and L_p are each divided into two types, based on whether the device is nominal or low power, and further into three types based on layout dependent information. Parameter variations are assumed to be independent, so a total of 14 different PVT parameters were considered in the analysis (12 for L, MCF, and V_{dd}). In our robustness analysis, the L_2-norm was used to compute all normed distances from the nominal point to the boundary hyperplanes, as is typical in design centering.

We ran both exact and approximate PSTA on different microprocessor blocks and

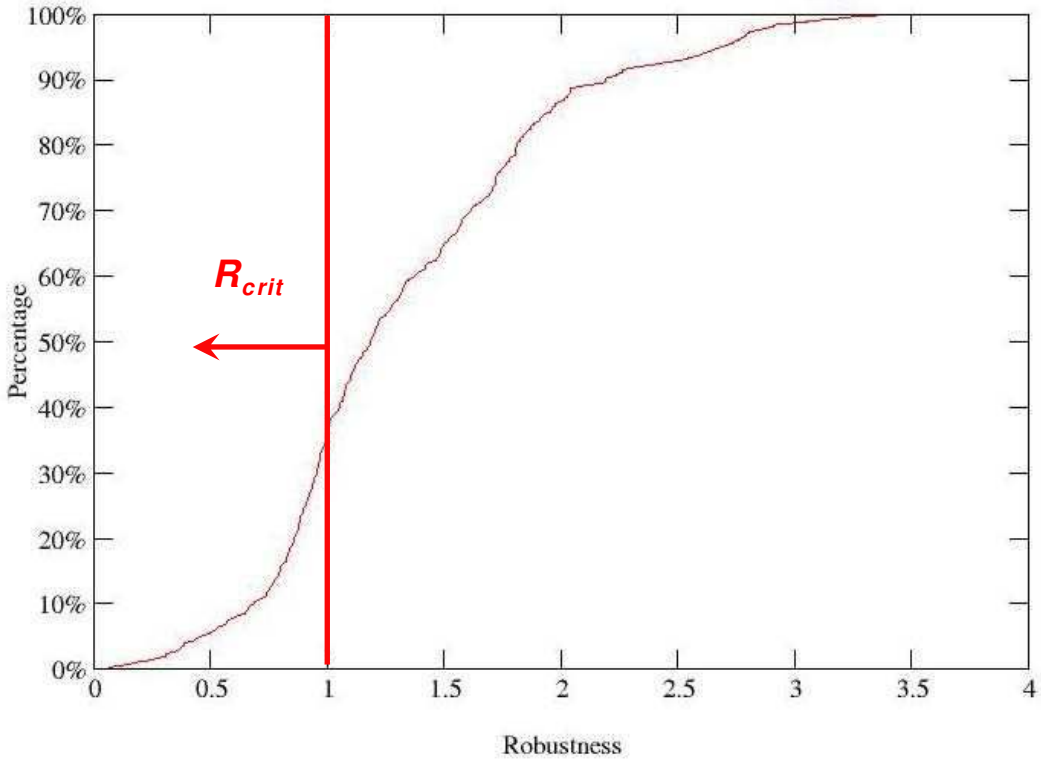


Figure 6.8: Cumulative robustness distribution of failed slacks

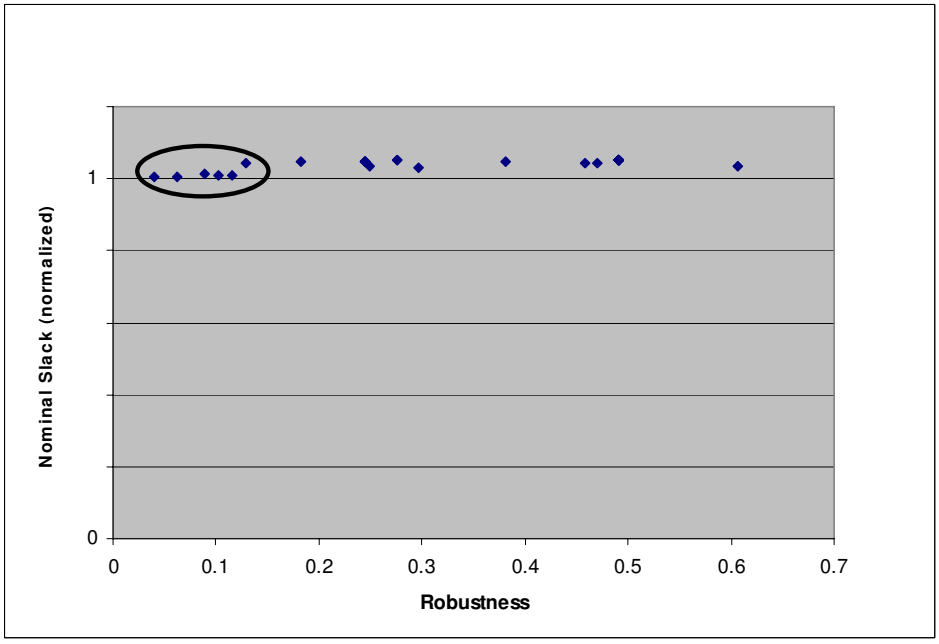


Figure 6.9: Nominal slack vs robustness

have determined parameterized arrival times at every node and parameterized slacks at the inputs of all registers in the blocks. Our robustness analysis was then applied on the parameterized slacks to quantify their robustness, as described in Section 6.5. Recall that we have normalized the variation range of every parameter to $[-1, 1]$, and have considered 14 parameters. Therefore, if a slack fails somewhere in the parameter space, then its robustness r must fall between $r_{\min} = 0$ and $r_{\max}$ (based on the L_2-norm), where:

$$r_{\max} = \sqrt{\sum_{i=1}^{14} (\pm 1)^2} \approx 3.74 \tag{6.24}$$

Fig. 6.8 shows a plot of the cumulative robustness distribution of all failed slacks for one microprocessor block, `ckt1`. It provides a ranking of all failed slacks according to where in the range [0, 3.74] their robustness falls. Looking at the plot, it makes sense to start by fixing the slacks that have the smallest values of robustness, since those are the ones that are most prone to failure. In general, it is useful to have a

robustness threshold, R_{crit}, such that all slacks with robustness less than R_{crit} would be considered critical and thus fixed. Note that R_{crit} does not have to be large since one is interested in detecting the slacks that are failing for a *small* deviation around the nominal point, at least for microprocessors.

Fig. 6.9 shows a plot of nominal slack vs robustness for different nodes. Slack values were normalized, and 16 nodes with very similar nominal slacks were picked (as shown). We have also assumed, for purpose of illustration, that if a slack goes below 90% of its nominal value (due to variations), then this would be considered a timing failure. In other words, we're simply using 90% of nominal slack instead of 0 as the threshold for failed slack. Based on this slack threshold, we have computed the robustness of the different slacks, and plotted nominal slack vs robustness. As shown on the plot, robustness values fall in [0.05, 0.6], which is a large spread given that the nominal slacks are almost the same. In fact, if one had no access to robustness information, all the 16 slacks would *seem* to be equally robust looking only at their nominal slack. However, after factoring in our robustness metric, one can easily determine which nodes are the most susceptible to variations. In a sense, the circled slacks are *closer to the edge of the cliff* than the ones with larger robustness.

We also checked if the results of robustness analysis are consistent when applied to *i)* exact PSTA vs *ii)* approximate PSTA. First, exact PSTA is invoked on another microprocessor block, `ckt2`, to obtain parameterized arrival times at every node and parameterized slacks at the inputs of all registers in the block. There were $\approx$ 1500 parameterized slacks, with n (number of hyperplanes defining every slack) ranging from 1 to 346 hyperplanes. Robustness analysis is then applied on the parameterized slacks with slack threshold set to 0, and (exact) robustness is recorded. Then, approximate PSTA is invoked, and parameterized slacks were obtained, each consisting of only a single hyperplane. Robustness analysis is then applied and (approximate) robustness is recorded. Out of the 1500 parameterized slacks, only 41 slacks failed somewhere in the parameter space, and thus have robustness values in [0, 3.74]. Fig. 6.10 shows a plot of approximate robustness (based on approximate PSTA) versus exact robustness (based on exact PSTA). In general, for the results of both robustness analyses to be consistent, the ranking of slacks in terms of their robustness should be preserved. In

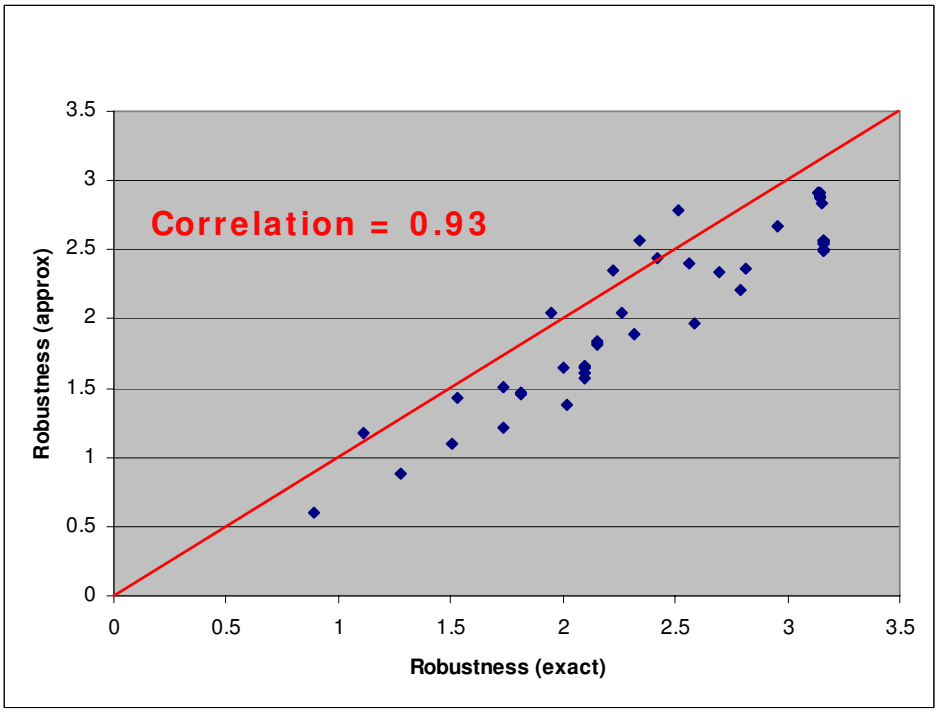


Figure 6.10: Ranking nodes according to their robustness: exact PSTA vs approximate PSTA

other words, the points should follow some straight line (not necessarily $y = x$, although $y = x$ would be an ideal case). This is what we see in the plot; the points are highly correlated (we have found the correlation to be $\rho = 0.93$) and they fall close to $y = x$.

6.7 Summary

In this chapter, we presented a new robustness metric that can be used to quantify the vulnerability of designs to parameter variations. Our robustness metric provides a novel way to easily interpret the results of parameterized static timing analysis by *i)* determining if and when nodes can fail, *ii)* ranking those nodes according to their robustness, and *iii)* fixing the ones that are least robust. Using distance as a metric for robustness, we find the smallest normed distance from the nominal point in the PVT space to any timing violation, which can be computed efficiently using closed

form expressions.

7 Statistical Leakage Estimation

7.1 Introduction

In this chapter, we present an efficient technique for finding the mean and variance of the full-chip leakage of a candidate design, while considering logic-structures and both die-to-die and within-die process variations, and taking into account the spatial correlation due to within-die variations. Our model uses a "random gate" concept to capture high-level characteristics of a candidate chip design, which are sufficient to determine its leakage. These high-level characteristics include information about the process, the standard cell library, and expected design characteristics. We show empirically that, for large gate count, the set of all chip designs that share the same high level characteristics have approximately the same leakage, with very small error. Therefore, our model can be used as either an *early* or a *late* estimator of leakage, with high accuracy. In its simplest form, we show that full-chip leakage estimation reduces to finding the area under a scaled version of the within-die channel length auto-correlation function, which can be done in constant time.

7.2 Background

As a result of technology scaling, leakage current is becoming a major design challenge, affecting both circuit performance and power. Leakage power is expected to continue to increase and due to limited power budgets, it will affect the feasibility of future microprocessor and ASIC designs [61]. Thus, estimating full-chip leakage becomes increasingly important. The leakage current of a circuit is not, however, simply the sum of the leakages of the devices in the circuit. Not only do logic-gate struc-

tures, such as stacking, affect the device leakage, but process variations make leakage estimation statistical in nature. Leakage current can be classified into two main components, namely sub-threshold leakage and gate tunneling leakage [62, 63]. Over the past few years, both types of leakage currents have been extensively studied, particularly sub-threshold leakage. Nevertheless, gate leakage can be important as noted in [64], although recent advances in the process, such as the introduction of new high-k materials and metal gates claim to have reduced gate leakage substantially.

Full-chip leakage estimation is useful at different points in the design flow. Towards the end of the design flow (late mode estimation), leakage estimation can be used as a final sign-off tool, and requires a complete netlist with possibly a circuit placement. On the other hand, early estimation of leakage (early mode estimation) provides the full-chip leakage given limited information about the design, which is very useful to allow for design planning.

Earlier work on leakage estimation [65–68] concentrated on early mode estimators. However, they either did not consider logic-gate structures and other transistor topologies, and/or did not consider the effect of correlation between the variations on the total leakage, which is important to model. Narendra et al. [65] estimate the mean of full-chip sub-threshold leakage; they consider within-die variations, but ignore within-die correlations and do not take into account the effect of gate topologies. Furthermore, they do not estimate the standard deviation of full-chip leakage. Rao et al. [66] estimate sub-threshold leakage by first finding fitting parameters for the leakage current for individual gates in the library, and use the parameters to map the leakage distribution of the gate, due to within-die variations, to a log-normal distribution. They compute the total leakage of a circuit block using an approximation for sums of independent lognormal distributions. The authors, however, ignore within-die spatial correlations. Rao et al. have also tackled the problem of estimating full-chip leakage in another way [67]. They model different types of leakage separately as a product of a nominal value and a multiplicative function that represents the deviation from the nominal value due to variations. While, the authors separate the variations into global and local variations, the local variations are considered to be independent, and thus the effect of correlation is not factored into the final result; also, they do not provide an estimate

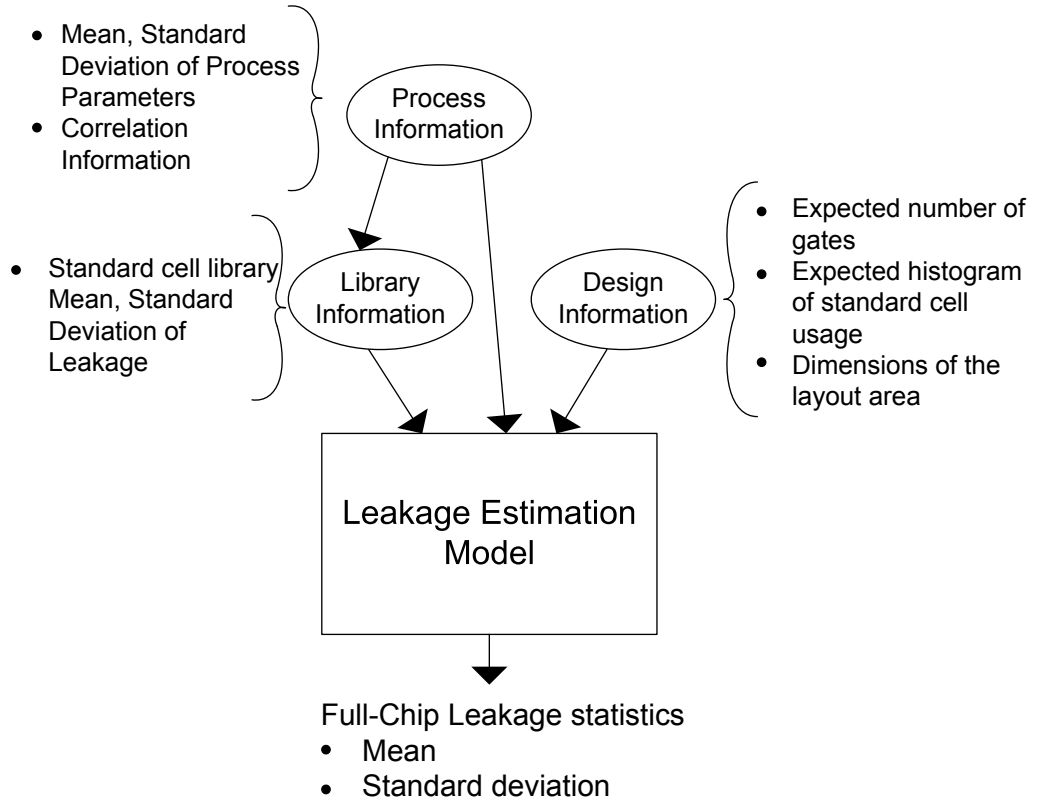


Figure 7.1: Leakage estimation model and the high-level characteristics required

of the standard deviation of full-chip leakage. Zhang et al. [68] in addition to considering process variations, also consider temperature and voltage variations. Instead of fitting the effect of process variations on leakage into an analytical equation, they use the BSIM model equations directly. However, just as the other early-estimators, they do not consider spatial correlations; they also do not consider gate topologies in their work.

More recent work [63, 69] has taken into consideration both the effects of gate topologies and within-die spatial correlation. Chang et al. [69] first precharacterize their library by fitting the different types of leakage currents to analytical forms. To model spatial correlation, they use the grid model [27] and determine the leakage in a grid by summing a set of correlated lognormal distributions, and then find the full-chip leakage distribution by summing the leakage distributions for each grid. Agarwal et al. [63] modeled spatial correlations differently using a quad-tree die partitioning method. To determine the final leakage distribution they sum the correlated lognormals using Wilkinson's method. Both these methods are late mode estimators of leakage, requiring minimally the circuit netlist and possibly a circuit placement to provide a leakage estimate. Also, since they operate at the level of the netlist, they can be expensive on large circuits, with a complexity of $\mathcal{O}(n^2)$ (some refinements are possible to reduce this cost, but with some loss of accuracy [69]).

7.3 Overview

Given the need to budget for power constraints, there is a need for accurate early mode estimators that take into consideration both correlation and gate topologies. As for late mode estimators, more efficient techniques are required. In this chapter, we present a new model and methodology for full-chip leakage estimation, in which certain high-level characteristics of a candidate chip design are used to determine its leakage statistics with high accuracy. For late mode estimation, these characteristics can be *extracted* from the netlist and/or placement. For early mode estimation, these characteristics can be simply specified as *expected* values based on previous design experience or on decisions made in the floorplanning stage. Our methodology uses a concept of a "ran-

dom gate" to capture these characteristics and considers both correlations and gate topologies. We show that these high-level characteristics are sufficient to determine the leakage statistics of a design. We restrict our analysis to sub-threshold leakage estimation, although our mathematical framework can be easily extended to handle gate leakage.

A flow diagram of the system is shown in Fig. 7.1. Given information about (1) the process, (2) the standard cell library, and (3) certain high-level design characteristics, we predict the mean and standard deviation of full-chip leakage. The process information includes the mean and standard deviation of the underlying process variations, such as the variations in transistor length or threshold voltage, and information regarding the within-die spatial correlation. The standard cell library information includes the leakage characteristics of the cell library under process variations; this information can be obtained by pre-characterizing the cells in the library. Finally, some information on the candidate design is needed, including the (extracted or expected) cell usage histogram (*i.e.*, frequency of use distribution) for cells in the library, the (extracted or expected) number of cells in the design, and the dimensions of the layout area. With this, we determine the full-chip leakage statistics (mean and variance) for the design.

To carry out the estimation, we propose a model that is *generic*, in the sense that it is a *template* for all designs that share the same values for these high-level characteristics. We use probability theory as the vehicle to implement this template, so that all designs that share the same values of these high-level characteristics will be *members* or *instances* of this probabilistic template model. We introduce the concept of the Random Gate (RG), which allows us to capture the characteristics of a candidate design. This allows the leakage statistics to be obtained in $\mathcal{O}(n)$ time, where n is the number of cells in the design, but we then also show that, for large gate counts, the statistics of the full-chip leakage can be written in integral form, allowing for the computational complexity of our estimator to become $\mathcal{O}(1)$ time[1]. The key point in this chapter is that *large* designs that share the same high-level characteristics will have approximately the same leakage statistics and, by leveraging this property, our

[1]When used as a late mode estimator, there will be some additional cost to extract the cell usage histogram from the netlist, but that also can be constant-time, or linear-time in the worst case.

estimation engine provides accurate and efficient estimation, either early or late in the design flow.

7.4 Modeling Process Variations

7.4.1 Parameter Model

Variations normally have two components: a Die-to-Die (D2D) component, and a Within-Die (WID) component. The D2D component is a variation between different instances of the die and is shared by all devices on the same die. The WID component of variation, however, causes different devices on the same die to have different process parameters; the WID variations have some correlation across the die. D2D and WID variations are considered to be (statistically) independent [41] and thus the total variance of a process parameter, such as the channel length (L), when both sources of variation are considered can be written as:

$$\sigma^2 = \sigma_{dd}^2 + \sigma_{wd}^2 \tag{7.1}$$

where σ_{dd}^2 is the variance of the D2D variation and σ_{wd}^2 is the variance of the WID variation. We will assume that all process parameters follow a Gaussian distribution, which is in line with the literature on leakage estimation. The resulting Random Variable (RV) for channel length (or any process parameter) can also be written with respect to their D2D and WID components as:

$$\mathbf{L}(i) \;=\; \mu + \sigma_{dd}\mathbf{Z}_0 + \sigma_{wd}\mathbf{Z}(i) \tag{7.2}$$

where i refers to an arbitrary device, μ is the mean of $\mathbf{L}$, $\mathbf{Z}_0$ is a zero mean standard normal RV with unit variance representing the D2D component which is shared by all devices on the die, and where the WID component is represented by a zero mean unit variance standard normal RV $\mathbf{Z}(i)$, a notation that emphasizes that it may be different for different devices on the die. For example, a second device j on the die will have a channel length with the following RV:

$$\mathbf{L}(j) = \mu + \sigma_{dd}\mathbf{Z}_0 + \sigma_{wd}\mathbf{Z}(j) \tag{7.3}$$

Given that $\mathbf{Z}(i)$ and $\mathbf{Z}(j)$ may be spatially correlated with a correlation of $\rho_{L_{wd}}(i,j)$, and that $\mathbf{Z}_0$ is shared by $\mathbf{L}(i)$ and $\mathbf{L}(j)$, then $\mathbf{L}(i)$ and $\mathbf{L}(j)$ will be correlated as well. We now show how we can express the total length correlation in terms of the D2D and WID breakdown, and the WID spatial correlation. Let us first write (7.2) and (7.3) as follows:

$$\mathbf{L}(i) - \mu = \sigma_{dd}\mathbf{Z}_0 + \sigma_{wd}\mathbf{Z}(i) \tag{7.4}$$

$$\mathbf{L}(j) - \mu = \sigma_{dd}\mathbf{Z}_0 + \sigma_{wd}\mathbf{Z}(j) \tag{7.5}$$

Assuming that the correlation between the WID variations $\rho_{L_{wd}}(i,j)$ is available as will be discussed in the next section, the covariance between the process parameters can then be written as:

$$E\left[(\mathbf{L}(i) - \mu)(\mathbf{L}(j) - \mu)\right] = \sigma_{dd}^2 + \sigma_{wd}^2 \rho_{L_{wd}}(i,j) \tag{7.6}$$

$$= \sigma^2 \rho_{L_{total}}(i,j) \tag{7.7}$$

where $\rho_{L_{total}}(i,j)$ is the total channel length correlation between $\mathbf{L}(i)$ and $\mathbf{L}(j)$ due to both D2D and WID components. We can now solve for the correlation between the total variation that composes both the D2D and WID variation to be:

$$\rho_{L_{total}}(i,j) = \frac{\sigma_{dd}^2 + \sigma_{wd}^2 \rho_{L_{wd}}(i,j)}{\sigma^2} \tag{7.8}$$

The above equation can be simplified if we define α to be a ratio of the D2D variance to the total variance as:

$$\alpha = \frac{\sigma_{dd}^2}{\sigma^2} \tag{7.9}$$

which allows the total correlation to be written as a function of the correlation of the

WID variation and the ratio of the D2D variance to the total variance:

$$\rho_{L_{total}}(i,j) = \alpha + (1-\alpha)\,\rho_{L_{wd}}(i,j) \tag{7.10}$$

The next section will describe the model that we used to capture the correlation in process parameters.

7.4.2 Correlation Model

Previous work on early-estimators of leakage did not take into account the spatial correlation that exists between the WID variations in the process parameters of different cells. However, to accurately estimate leakage, spatial correlation between variations must be taken into account [63, 69].

To model the WID spatial correlation between variations in transistor characteristics, we assume the existence of a spatial correlation function [70] that depends on the *distance* between the two transistors. Given the D2D and WID parameter variances, and the WID correlation, one can easily determine the total correlation between parameter variations (due to D2D and WID effects) by a simple normalization as was shown in (7.10). Not all functions, however, can be used as a spatial correlation function [70]; specifically spatial correlation functions are a family of monotonically decreasing non-negative functions [70]. One example of a spatial correlation function for WID variations is shown as the bold line in Fig. 7.2; the function has a correlation of one at a distance of zero since the devices are in fact the same device [70]; furthermore the sudden drop from one at distance zero is the cause of a uncorrelated random variation in the parameter, which exists even in devices that are very close together [70]. The above spatial correlation function considers only the correlation in WID variations and therefore it dies down to zero after a certain distance. If we consider the D2D variation using (7.10), the total correlation will decrease down to α as shown in Fig. 7.2, where α is set to 0.5 [70].

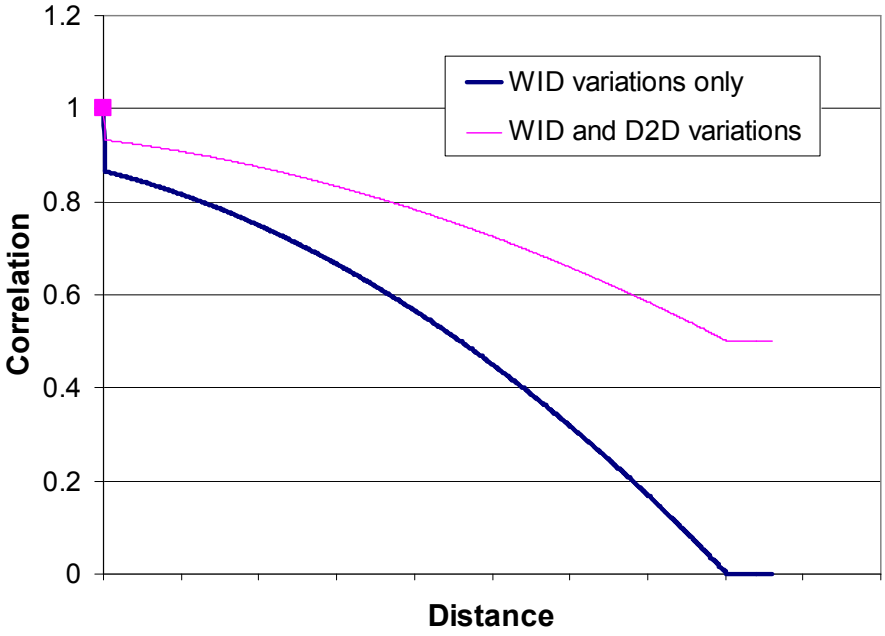


Figure 7.2: Possible correlation model considering both within-die and die-to-die variations

7.5 Modeling at the Cell Level

While the statistics of the underlying process parameters can be obtained from the foundry, the leakage statistics of each cell cannot be immediately obtained. Since each cell has a different topology, with different transistor stacks, the leakage in each cell is affected differently by the underlying variations in the transistor length and threshold voltage. Furthermore the cell's input vector states also affect the leakage distribution of each cell.

7.5.1 Cell Leakage

Leakage current is determined primarily by transistor, not interconnect, parameters. Of the many transistor parameters that affect sub-threshold leakage, the truly relevant ones are channel-length (L) and threshold voltage (V_t), as shown in [71], due to the exponential dependence of sub-threshold leakage current on these two parameters. Threshold voltage variations are mainly due to two effects: random dopant fluctuations in the channel and the V_t roll-off effect whereby V_t varies in response to variations in L. In this chapter, when we refer to V_t variations, we specifically refer to the effect of random dopant fluctuations. We lump the effect of V_t roll-off on leakage into the L variations, because the two are directly related. This allows us to make the simple statement that V_t variations are purely random (independent) across the die [72], while L variations are not [69] (they include some within-die correlation). This approach is in line with the modern treatment of leakage in published work [66].

Since V_t variations are independent, while L variations are not, it follows immediately that, for full-chip leakage estimation, while V_t variations may be relevant for finding the *mean* of the total leakage, they are definitely not relevant for finding the *variance* of the total leakage. The reason for this is simple: the variance of the sum of n independent random variables is $\sim n\sigma^2$, while the variance of the sum of n highly correlated random variables is $\sim n^2\sigma^2$. Thus, for large chips (large n), the variance of chip leakage due to V_t variations is negligible compared to that due to L variations. This too is in line with the modern published work on leakage [66]. Thus, for leakage *variance* estimation, we can focus on L alone. As for the effect of V_t variations on

the *mean* leakage, that can be easily determined through a multiplicative term that depends on the variance of V_t, which is derived from the mean of the log-normal distribution, similar to [73]. As this is standard textbook material, it will not be covered here.

To model the distribution of the leakage of each cell, we use two methods that have different levels of computational complexity and accuracy. The first method uses a Monte-Carlo (MC) analysis to obtain the leakage statistics of each cell. While this technique needs extensive simulations, it does give us some confidence in the resulting distributions. The second method, an analytical approach, uses a limited sampling of the leakage of the cell, and then fits the leakage of the cell into a functional form, from which we easily compute the mean and variance of the distribution. These two methods are discussed below, and we then discuss correlation and circuit state dependency.

Monte-Carlo Technique

We use a commercial 90nm CMOS technology, along with its associated standard cell library of which we use 62 cells that include the Static Random Access Memory (SRAM) cell, various flip flops and a range of different logic cells. For each cell and input combination, we perform a MC analysis to determine the mean and standard deviation of the cell's leakage. The MC analysis is done assuming all the variations in the transistor channel length within the cell are completely correlated, which is reasonable in practice given that the transistors in each cell are very close together. This is in line with previous work [69], where all cells within a grid are assumed to be completely correlated.

Analytical Technique

Rao et al. [66] introduced a mathematical model to express the leakage current, X, of a given cell as a function of channel length, L as:

$$X = ae^{bL+cL^2} \tag{7.11}$$

and showed that the analytical BSIM3 models vastly overestimated the leakage of devices that had gate lengths that deviated by more than 5% from their nominal, and that the fitted model above with the triplet (a, b, c) can accurately model the leakage of different topologies including individual transistors and transistor stacks [66].

In our work, we first fit each cell's leakage into (7.11), and then use the triplet (a, b, c) to determine analytically the mean and variance of the underlying leakage distribution. To determine the triplet for each cell, we first perform a series of seven SPICE simulations, where the length of the transistors in the cell are modified from -3σ to 3σ in intervals of σ (where σ is the standard deviation in the transistor length) and measure the leakage. We then perform the Levenberg-Marquardt [74] method to fit the data into the above functional form and obtain (a, b, c). The model in (7.11) can fit the leakage of most cells quite well as can be seen in Fig. 7.3 where the analytical model is compared to SPICE simulations of a four-input-AND-into-OR cell. For some cells, however, the analytical model does not fit quite as well, as can be seen in Fig. 7.4, for a double-two-input-AND-into-two-input-NOR cell.

Note that, unlike [66] where numerical integration is used to approximate the leakage mean and variance, we use the fitted model with the triplet (a, b, c) to determine analytically and exactly the mean and variance of the underlying leakage distribution. The complete derivation, which was moved to Appendix A, results in the following:

$$\mu_{\mathbf{X}} = M_{\mathbf{Y}}(1) \tag{7.12}$$

$$\sigma^2_{\mathbf{X}} = M_{\mathbf{Y}}(2) - \mu^2_{\mathbf{X}} \tag{7.13}$$

where $M_{\mathbf{Y}}(t)$ is the moment-generating function of $\mathbf{Y} = \ln \mathbf{X}$ which can be shown to be:

$$M_{\mathbf{Y}}(t) = (1 - 2K_1 t)^{-\frac{1}{2}} e^{\left[\frac{K_2^2 K_1 t}{1-2K_1 t} + K_3 t\right]} \tag{7.14}$$

by using the moment generating function of the "Non-Central Chi-square" distribution where K_1, K_2 and K_3 are simple functions of the regression parameters (a, b, c) and

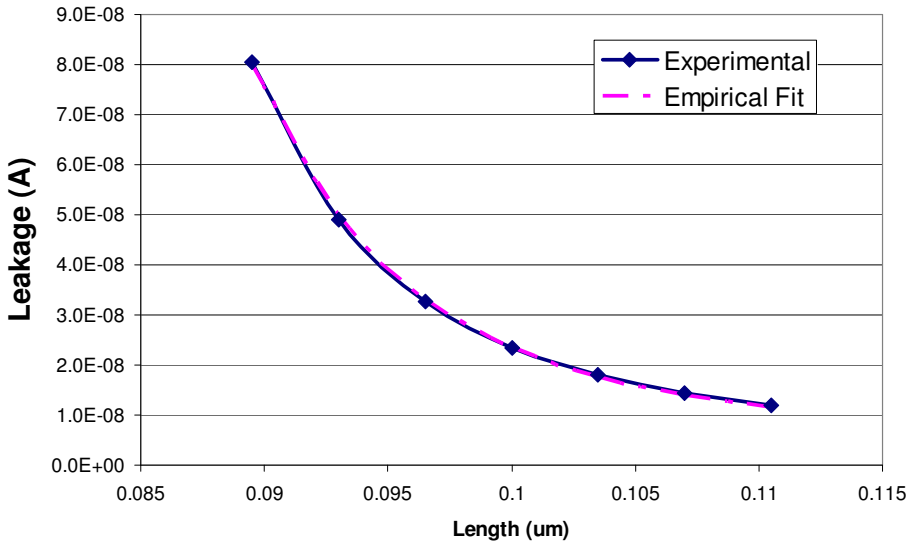


Figure 7.3: Comparison of analytical fit with results from SPICE of an AO cell

the mean μ and standard deviation σ of the channel length, as follows:

$$K_1 = c\,\sigma^2 \qquad K_2 = \frac{1}{\sigma}\left(\frac{b}{2c} + \mu\right) \tag{7.15}$$

$$K_3 = \ln a + b\mu + c\,\mu^2 - c\left(\frac{b}{2c} + \mu\right)^2 \tag{7.16}$$

To check the accuracy of the analytical model in determining the mean and standard deviation of cell's leakage, we compare the results obtained from the fitted model to the results obtained through MC analysis for all 62 cells with all input combinations. For the mean, the analytical method is quite close to the MC results; there is less than a 2% error for all gates, and the average absolute error is 0.44%. For the standard deviation, the average absolute error is 3.1%, and the maximum error is about 10%. The histogram of error in the mean and standard deviation for all cells and all input combinations is shown in Appendix A. Note that the error in the mean and standard deviation is not a result of the mathematical derivation, but due to the leakage curve

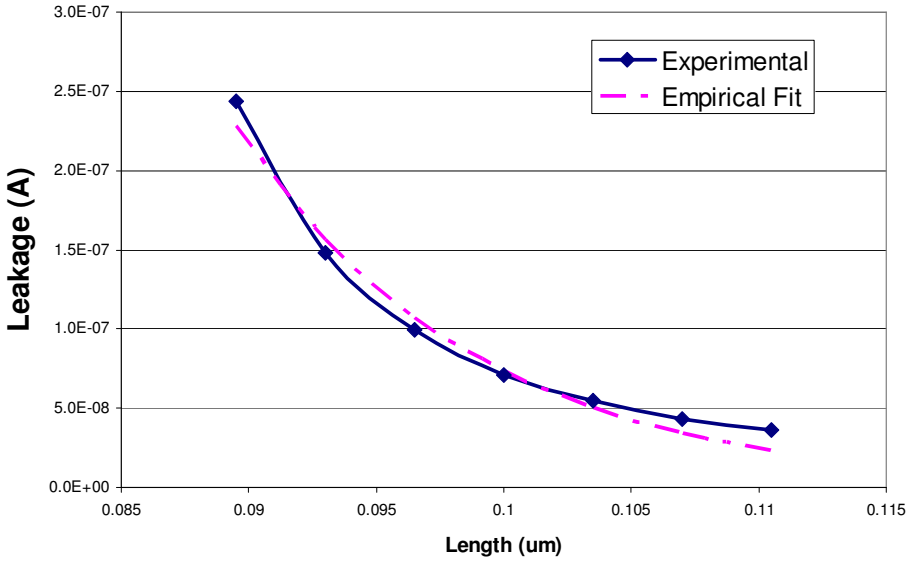


Figure 7.4: Comparison of analytical fit with results from SPICE of an double-two-input-AND-into-two-input-NOR

not being exactly mapped to the functional form ae^{bL+cL^2}. Thus, there is a trade-off between computational complexity and accuracy; if MC analysis is performed on all gates, then the distribution models for all gates will have high accuracy; on the other hand, using the functional form requires minimal simulation time.

7.5.2 Leakage Correlation

As mentioned earlier, we assume the existence of a spatial correlation function that gives the correlation between *process parameters* as a function of the distance separating two locations, but which does not provide the correlation between the *leakages* of two cells at these locations. Using the regressed triplets, (a, b, c), we have developed an analytical method that determines the leakage correlation between any pair of gates placed at two arbitrary locations on the die given the correlation in their channel lengths. In other words, we have determined a *mapping* $\rho_{m,n}(l_i, l_j) = f_{m,n}(\rho_L(l_i, l_j))$ where $\rho_L(l_i, l_j)$ is the channel length correlation between two locations l_i and l_j, $f_{m,n}(\cdot)$ is the derived

mapping for gates m and n and $\rho_{m,n}(l_i, l_j)$ is the leakage correlation for gates m and n placed at locations l_i and l_j respectively. Note that the mapping depends on the types of gates m and n since a triplet (a, b, c) is associated with every gate type.

The details of the derivation leading to this mapping were omitted from this section and moved to Appendix B. Fig. 7.5 shows the results of the leakage correlation for a pair of gates given channel length correlation, as determined by the analytical mapping $f_{m,n}(\cdot)$, compared with the leakage correlation from MC analysis; note that the analytical technique shows a good match to the MC results. Also the leakage correlation is near the $y = x$ line, at which leakage correlation equals channel length correlation. We have performed the analysis for all pairs of gates, and shown that the analytical mapping provides accurate results in all cases. The set of mappings $f_{m,n}(\cdot)$ for different pairs of gates are slightly different but they all closely follow the $y = x$ line (refer to Appendix B). We will use this observation that the leakage correlation is close to the length correlation in the case where MC analysis is used to obtain the cell leakage statistics since we do not have the (a, b, c) triplet to obtain the leakage correlation exactly. We will discuss this in more detail in Section 7.7.1.

7.5.3 Input Combinations

The signal probability (probability that a logic signal is 1) certainly has an effect on leakage. This effect is quite strong for single logic gates, causing a spread of 10× in some cases. However, for large circuits, the impact of signal probability is significantly diminished due to averaging of their effects (law of large numbers). To study this effect, we have swept the signal probabilities from 0 to 1 and have found, as shown in Fig. 7.6, that the effect on large circuit leakage is not pronounced and is also dependent on the frequency by which various cells are employed in the design. The figure shows the leakage mean, and similar behavior has been found for the leakage variance. For a practical solution approach, one has the option of simply setting the signal probabilities at some ball-park mid-level value, such as 0.5. A better approach, which we employ, is to first characterize every cell for all its input states; then, based on this pre-characterized data, and for the given frequency of use distribution for cells, find the signal probability setting which maximizes the mean leakage, effectively finding the maximum of a plot

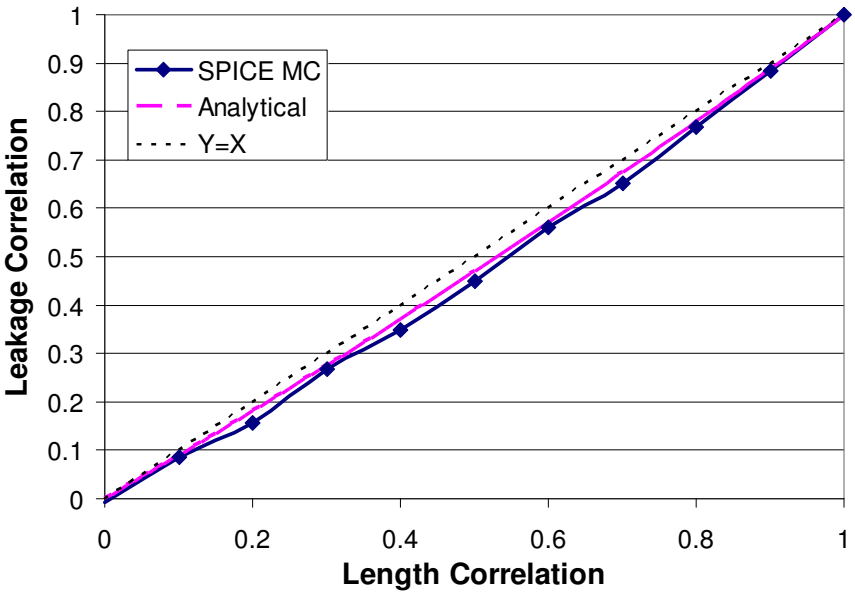


Figure 7.5: Correlation in leakage vs correlation in channel length for a pair of gates

such as Fig. 7.6. Empirically, we find that this setting turns out to be very good for finding the maximum leakage mean for the candidate design, as well as its maximum leakage variance. This approach gives a conservative estimate, in the face of uncertainty about eventual signal probabilities.

7.6 Full-Chip model

What determines the leakage of a large circuit? We will demonstrate empirically that certain high-level characteristics of a candidate design are sufficient to determine its leakage. In a library-based standard-cell design environment, these characteristics are:

1. The cell library (characterized for leakage)
2. The (actual or expected) frequency of usage for cells in the library
3. The (actual or expected) number of cells in the design

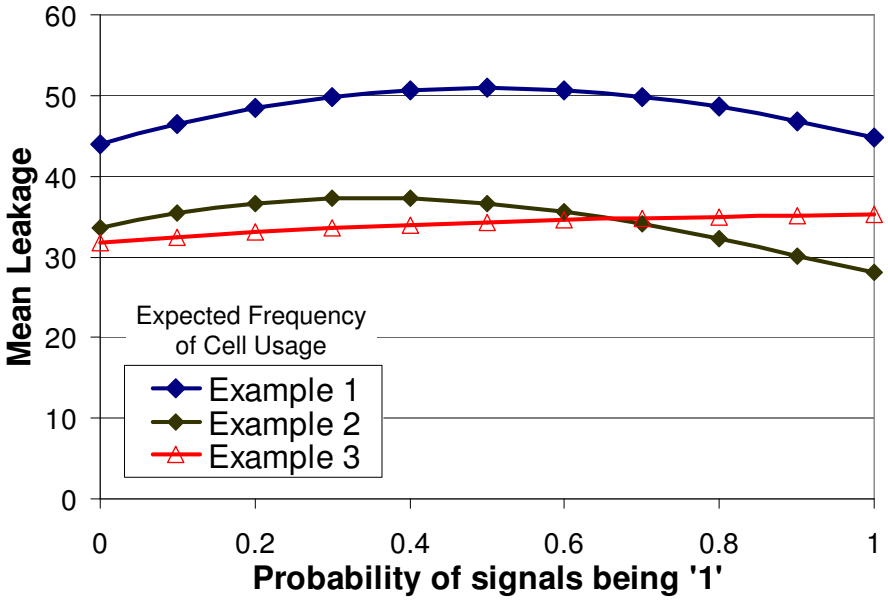


Figure 7.6: Effects of signal probability on chip leakage

4. The dimensions of the layout area

To carry out the leakage estimation, we propose a model for the candidate chip design that is *generic*, in the sense that it is a *template* for all designs that share the same values for these high-level characteristics. We use probability theory as the vehicle to construct this template, so that all designs that share the same values of these high-level characteristics will be *members* or *instances* of this probabilistic template model. After developing our leakage predictor based on this model, we will then show that the leakages of all instances of specific designs which are members of this model converge towards the predicted leakage value as the circuit size increases; Fig. 7.9 offers a "sneak preview" of this convergence.

7.6.1 Model Definition and Suitability

Formally, our full-chip model is a rectangular array of a number (n) of identical *sites*, as shown in Fig. 7.7, where every site is occupied by a probabilistic abstraction that we call a *random gate* (RG), and such that the dimensions of the array are equal to the dimensions of the layout area of the candidate design, and that the number of sites n is equal to the number of cells in the design. But what is a RG? Simply put, a RG is similar to a Random Variable (RV); however, unlike a RV that assumes real numbers as *outcomes* or *instances*, the instances of a RG are gates from the standard-cell library, with probabilities identical to those in the frequency of use distribution. In other words, the RG discrete probability distribution is identical to the frequency of cell usage of the design.

This full-chip array model is a suitable probabilistic representation of all designs having the high-level characteristics highlighted earlier. On one hand, its dimensions and gate count match the dimensions of the layout and the number of cells in the candidate design. On the other hand, the frequency of cell usage of the design is also matched by the way the RG discrete probability distribution is defined. Hence, if an *instance* of the full-chip model is defined to be n RG instances at every site in the array, then the frequency of cell usage for that full-chip model *instance* will be identical to the frequency of cell usage of the candidate design, for large n. Therefore, the full-chip model is a probabilistic representation of a set of designs with the same high-level characteristics, and those designs are in fact *instances* of our model. Using this fact, we will use the full-chip model to estimate the leakage of the candidate design.

One possible reaction to this proposal is that all sites in the full-chip model are of identical size while obviously cells in the library are of different sizes. Another comment is that the array seems to leave no room for interconnect routing. Both these issues do not present a problem. In fact, the size of a site is really the size of the layout area, divided by the number of cells, thus it is the average size of a cell and the interconnect that may be associated with it. Thus, all that is captured by the notion of a RG site is the idea that the leakage due to one cell would on average be spread out or "allocated" to the layout area of a single site.

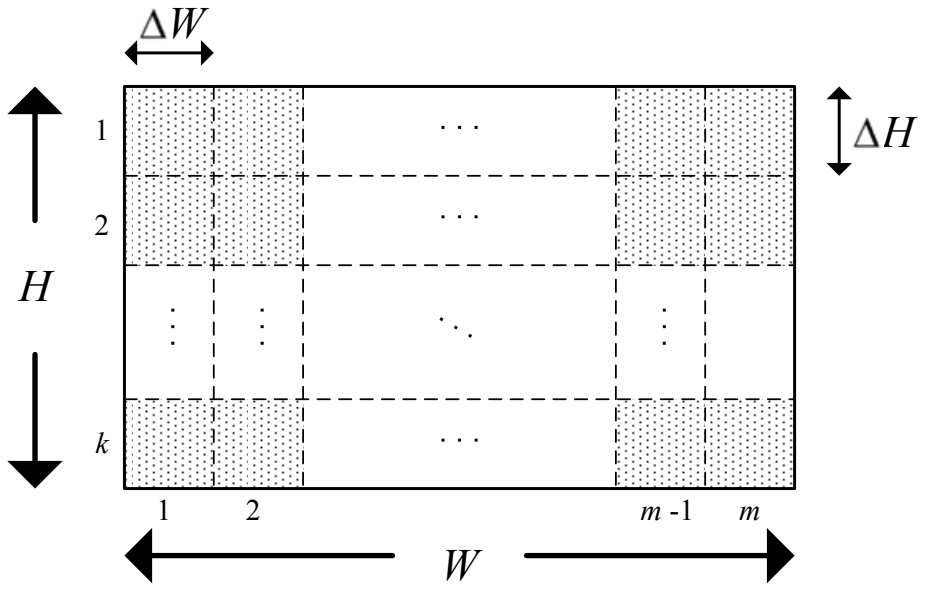


Figure 7.7: Abstract organization of die

7.6.2 Leakage Statistics of a Random Gate

As stated earlier, the RG is simply a gate picked at random from the library, according to a discrete probability distribution that is identical to the frequency of gate usage. To perform full-chip leakage estimation based on our model, we need to construct and mathematically define the leakage statistics of the RG.

Let $\mathbf{I}$ be an RV that takes as values the *type* of a gate picked from the library at random to be used in the design. This means that $\mathbf{I} \in \{1, 2, \ldots, p\}$, where p is the total number of gates in the library, and that the distribution of $\mathbf{I}$ is identical to the frequency of gate usage. Let α_i be the frequency of usage of gate i. Then:

$$\mathcal{P}\{\mathbf{I} = i\} = \alpha_i \quad \forall i = 1, 2, \ldots, p \quad \text{and} \quad \sum_{i=1}^{p} \alpha_i = 1 \tag{7.17}$$

Let $\mathbf{X_I}$ be an RV that represents the leakage of a gate picked according to the distribution of $\mathbf{I}$. Then by definition, $\mathbf{X_I}$ is the leakage of the RG. Consequently, $\mathbf{X_I}$ is defined on two probability spaces; the space of $\mathbf{X}$ due to channel length variations, and the

space of $\mathbf{I}$ due to the choice of gate type. Note that for an arbitrary realization of say $\mathbf{I} = i$, $\mathbf{X_I}$ will be equal to $\mathbf{X}_i$, that is the RV that represents the leakage of gate of type i. Recall that the statistics of $\mathbf{X}_i$, *i.e.*, its mean μ_i and standard deviation σ_i, have already been determined during pre-characterization for all gates i in the library, using either the MC or the analytical techniques. We can determine the mean leakage $\mu_{\mathbf{X_I}}$ of the RG as follows:

$$\mu_{\mathbf{X_I}} = E\left[\mathbf{X_I}\right] = E_I\left[E_X\left[\mathbf{X_I} \,|\, \mathbf{I} = i\right]\right] \tag{7.18}$$

$$= E_I\left[E_X\left[\mathbf{X}_i\right]\right] = \sum_{i=1}^{p} \alpha_i \, \mu_i \tag{7.19}$$

where $E_X[\cdot]$ and $E_I[\cdot]$ are the expected values over the spaces of $\mathbf{X}$ and $\mathbf{I}$, respectively. To determine the variance $\sigma^2_{\mathbf{X_I}}$ of $\mathbf{X_I}$, we start by determining its second moment $E\left[{\mathbf{X_I}}^2\right]$ as:

$$E\left[{\mathbf{X_I}}^2\right] = E_I\left[E_X\left[{\mathbf{X_I}}^2 \,|\, \mathbf{I} = i\right]\right] \tag{7.20}$$

$$= E_I\left[E_X\left[\mathbf{X}_i^2\right]\right] = \sum_{i=1}^{p} \alpha_i \left(\sigma_i^2 + \mu_i^2\right) \tag{7.21}$$

Given the second moment and the mean, the variance can be determined as follows:

$$\begin{aligned} \sigma^2_{\mathbf{X_I}} &= E\left[{\mathbf{X_I}}^2\right] - \mu^2_{\mathbf{X_I}} \\ &= \sum_{i=1}^{p} \alpha_i \left(\sigma_i^2 + \mu_i^2\right) - \left(\sum_{i=1}^{p} \alpha_i \, \mu_i\right)^2 \end{aligned} \tag{7.22}$$

To account for different input states, the summation in the above equations for the mean and variance are updated to account for the different weights of each input state.

7.6.3 Random Gate Leakage Correlation

In addition to the RG leakage statistics defined in the previous section, we need to construct and define the RG leakage correlation.

Recall that $\mathbf{X_I}$ is defined as the leakage of a random gate picked from the library

according to the distribution of $\mathbf{I}$, and placed at some location on the die. Let $\mathbf{X_I}(l_i)$ and $\mathbf{X_I}(l_j)$ be the leakages of the two RGs at two arbitrary locations l_i and l_j. It is important to understand that $\mathbf{X_I}(l_i)$ and $\mathbf{X_I}(l_j)$ are identically distributed, and any correlation among these RVs is only due to the correlation over the space of process variations and not over the space of gate selection.

Let $C_{\mathbf{X_I}}(l_i, l_j)$ be the covariance of $\mathbf{X_I}(l_i)$ and $\mathbf{X_I}(l_j)$, which is defined as $C_{\mathbf{X_I}}(l_i, l_j) = E\left[\mathbf{X_I}(l_i)\,\mathbf{X_I}(l_j)\right] - \mu^2_{\mathbf{X_I}}$. It can be shown, using conditional expectation, that this covariance is given by:

$$C_{\mathbf{X_I}}(l_i, l_j) = \sum_{m=1}^{p}\sum_{n=1}^{p} \alpha_m\,\alpha_n\,C_{m,n}(l_i, l_j) \tag{7.23}$$

where $C_{m,n}(l_i, l_j)$ is the covariance of the leakage of two gates of types m and n, when placed at locations l_i and l_j, respectively, *i.e.*, $\mathbf{X}_m(l_i)$ and $\mathbf{X}_n(l_j)$. Note that the covariance of the leakage of the random gate $\mathbf{X_I}$ is the expected value over $\mathbf{I}$ of the covariances of all pairs of gate types. This result is somewhat intuitive since the random gate is an abstraction that embodies all gates in the library. Starting from (7.23), we can normalize $C_{m,n}(l_i, l_j)$ by the standard deviations of gates m and n to get their leakage correlation $\rho_{m,n}$. Then, we use the analytical mapping $f_{m,n}(\cdot)$ from Section 7.5.2 to relate the leakage correlation $\rho_{m,n}$ to channel length correlation ρ_L, as follows:

$$\begin{aligned} C_{\mathbf{X_I}}(l_i, l_j) &= \sum_{m=1}^{p}\sum_{n=1}^{p} \alpha_m\,\alpha_n\,[\rho_{m,n}(l_i, l_j)\sigma_m\,\sigma_n] \\ &= \sum_{m=1}^{p}\sum_{n=1}^{p} \alpha_m\,\alpha_n\,\sigma_m\,\sigma_n\,f_{m,n}(\rho_L(l_i, l_j)) \end{aligned} \tag{7.24}$$

Let $F(\rho_L(l_i, l_j))$ be equal to the final expression in (7.24) above, and notice that this equation assumes that l_i and l_j are different. When they are the same, $C_{\mathbf{X_I}}(l_i, l_j)$ is just the variance $\sigma^2_{\mathbf{X_I}}$. Thus:

$$C_{\mathbf{X_I}}(l_i, l_j) = \begin{cases} F(\rho_L(l_i, l_j)) & \text{for} \quad l_i \neq l_j \\ \sigma^2_{\mathbf{X_I}} & \text{for} \quad l_i = l_j \end{cases} \tag{7.25}$$

By enforcing this correlation structure on our RG array, we ensure that instances of this array have the same correlation structure as the candidate design.

7.7 Full-Chip Leakage Estimation

For a specific placed design, based on a pre-characterized cell library, one can determine the full-chip leakage statistics using techniques from standard probability theory [24] for finding the sum of a number of correlated RVs (each RV corresponds to the leakage of one cell instance). This would be an $\mathcal{O}(n^2)$ approach, which can be expensive for large circuits (some refinements are possible to reduce this cost, but with some loss of accuracy [69]). Throughout this chapter, we will refer to the leakage obtained from such an $\mathcal{O}(n^2)$ approach as the *true leakage* of a given design.

Apart from the issue of computational cost, such an approach is available only later in the design flow once a netlist and placement are available; it is useful only as a final check, and not as a prelude to corrective action. In this section, we will first show how we can determine the full-chip leakage statistics in linear time, $\mathcal{O}(n)$, and then show how this can be improved to obtain the statistics in constant time, $\mathcal{O}(1)$. Importantly, we will also show that, for large gate counts, the statistics of any specific design that shares the same high-level characteristics under consideration converge to the values predicted by our model.

7.7.1 Linear-time method

Let $\mathbf{I}_T$ be an RV that represents the leakage of our full-chip model, *i.e.*, of the array of n RGs. This means that:

$$\mathbf{I}_T = \sum_{i=1}^{n} \mathbf{X}_\mathbf{I}(l_i) \tag{7.26}$$

where l_i is the location of the i^{th} random gate. We are interested in determining the statistics of $\mathbf{I}_T$, namely its mean $\mu_{\mathbf{I}_T}$ and variance $\sigma^2_{\mathbf{I}_T}$. The mean of $\mathbf{I}_T$ is equal to:

$$\mu_{\mathbf{I}_T} = E\left[\mathbf{I}_T\right] = \sum_{i=1}^{n} E\left[\mathbf{X}_\mathbf{I}(l_i)\right] = \sum_{i=1}^{n} E\left[\mathbf{X}_\mathbf{I}\right] = n\,\mu_{\mathbf{X}_\mathbf{I}} \tag{7.27}$$

The variance of $\mathbf{I}_T$ can be easily determined using a result from probability theory that the variance of a sum of correlated RVs is equal to the sum of pairwise covariances [24]. In other words:

$$\sigma^2_{\mathbf{I}_T} = \sum_{a=1}^{n}\sum_{b=1}^{n} C_{\mathbf{X_I}}(l_a, l_b) \tag{7.28}$$

Note that the above double summation accounts also for the cases where $l_a = l_b$, for which the covariance is essentially the variance. Using the fact that any covariance can be written in terms of the correlation, $C_{\mathbf{X_I}}(l_a, l_b) = \rho_{\mathbf{X_I}}(l_a, l_b)\sigma^2_{\mathbf{X_I}}$, we can write the total leakage variance in its final form:

$$\sigma^2_{\mathbf{I}_T} = \sigma^2_{\mathbf{X_I}} \sum_{a=1}^{n}\sum_{b=1}^{n} \rho_{\mathbf{X_I}}(l_a, l_b) \tag{7.29}$$

where the variance of the full-chip leakage is a function of the variance of the random gate and the extent of leakage correlation across the chip.

At this point, we have determined the mean of the total leakage (in constant time), and have shown that the computation of the variance of the total leakage requires a double summation over the number of gates on the chip. This $\mathcal{O}(n^2)$ complexity is not practically acceptable, especially knowing that n can be extremely large, on the order of millions. By taking into account the rectangular shape of the die and the sole dependence of the leakage correlation on the distance between different locations, we are able to cut down the complexity of computing the total leakage variance to $\mathcal{O}(n)$, as follows.

Let the RG array consist of k rows and m columns, where the total number of gates, n, is equal to the product $k \times m$, as shown in Fig. 7.7. Each location or "site" on the grid can be represented by a pair (r, s) where r is the horizontal index taking values $r = 1, \ldots, m$ and s is the vertical index taking values $s = 1, \ldots, k$. Also, assume that the height H and width W of the array are known. Let ΔH and ΔW be the height and width of the site where every gate will be placed.

Given the above parameters, the centre to centre distance d_{ij} between any two

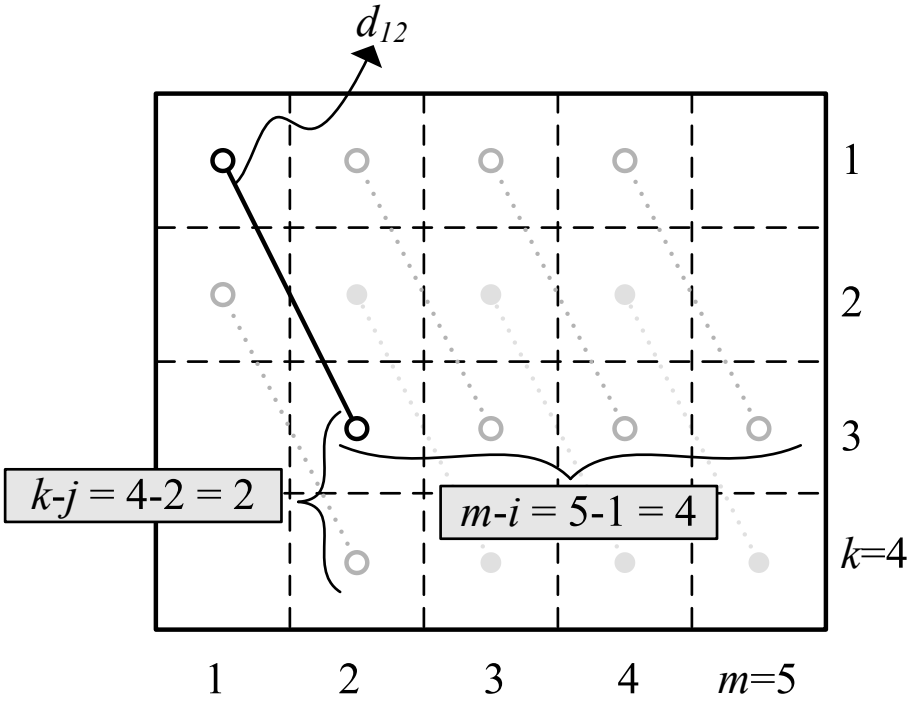


Figure 7.8: Number of occurrences of a certain distance vector

sites (r_1, s_1) and (r_2, s_2) can be easily determined to be $d_{ij}=\sqrt{(i\cdot\Delta W)^2+(j\cdot\Delta H)^2}$ where i is defined as the algebraic difference in horizontal indices, *i.e.*, $(r_2 - r_1)$, and j is defined as the algebraic difference in vertical indices, *i.e.*, $(s_2 - s_1)$. Note that $i = 0, \pm1, \ldots, \pm(m-1)$ and $j = 0, \pm1, \ldots, \pm(k-1)$.

Now recall the total leakage variance defined in (7.29) where the double summation covers all possible pairs of locations, and each location is a site on the grid defined by two indices. Since the correlation depends only on the distance d_{ij} between the pairs of locations, we can simplify the above expression greatly by performing the sum over the different distances rather than the pairs of locations. To do that, however, we need to determine the number of times each distance d_{ij} occurs. This is relatively easy for a rectangular $k \times m$ grid, as can be seen in Fig. 7.8, where the number of times a distance d_{ij} occurs along the width of the die is $m - i$ and along the height of the die is $k - j$. Using these two value, the number of occurrences n_{ij} of d_{ij} can be determined to be

the following:

$$n_{ij} = (m - |i|) \cdot (k - |j|) \tag{7.30}$$

Since the leakage correlation between any two given locations depends only on the distance between these locations, we will explicitly highlight this fact, $\rho_{\mathbf{X_I}}(l_a, l_b) = \rho_{\mathbf{X_I}}(d_{ij})$ where i and j in the above equation are the algebraic differences in the horizontal and vertical indices of l_a and l_b.

Starting from (7.29), we will transform the quadratic summation that runs over all pairs of locations, into a summation that runs over the set of possible distances induced by the rectangular shape of the grid. This set will be covered if all the algebraic differences i and j are covered. After accounting for the number of times each algebraic difference occurs, n_{ij}, we get the following expression for the total leakage variance:

$$\sigma^2_{\mathbf{I}_T} = \sigma^2_{\mathbf{X_I}} \sum_{i=-m}^{m} \sum_{j=-k}^{k} (m - |i|) \cdot (k - |j|)\ \rho_{\mathbf{X_I}}(d_{ij}) \tag{7.31}$$

where the double summation runs at most $\mathcal{O}(k \times m) = \mathcal{O}(n)$ times, which is linear in circuit size. Note that the expression in (7.31) is an exact transformation of (7.29) without any approximations, and was possible due to different factors:

1. The concept of random gate which allows us to express the total leakage as a sum of *identically* distributed RVs. This in turn made it possible to extract the variance of the random gate outside the double summation in (7.29).
2. The sole dependence of the leakage correlation on the *distance* between the pair of locations rather on the location itself.
3. The rectangular shape of the grid, which allows for closed form expression of the number of times each distance occurs.

Next, we validate our full-chip leakage model, both as an early and a late estimator of leakage.

Validation

Two types of validation tests were run, by first considering randomly generated circuits, as a way to make conclusions about the set of all circuits of a given size, and then by

Table 7.1: % Error in full-chip standard deviation for ISCAS85 circuits compared to the RG estimates

c499	c1355	c432	c1908	c880	c2670	c5315	c7552	c6288
1.04%	0.41%	1.14%	0.36%	0.74%	0.52%	0.23%	0.34%	1.38%

considering specific benchmark circuits.

In the first set of experiments, a large number of circuits were randomly generated so as to match a frequency of cell usage that was specified *a priori*. The circuits were then placed and routed using Cadence Encounter, and their true leakage statistics (mean and variance) were found. Fig. 7.9 shows the maximum positive and negative difference between the means and standard deviations of the leakages of these circuits compared to the estimates provided by our model. It can be seen that as the number of gates in the circuits increases, the difference approaches zero; at a circuit size of 11,236 gates, the maximum difference is 2.2%. This small amount of error indicates that the set of all chip designs that share the same high level characteristics have approximately the same full chip leakage statistics and thus these high-level characteristics are sufficient to determine chip leakage. This first set of experiments serves to justify the statement that this approach is useful as an *early estimator* of full-chip leakage.

In the second set of experiments, we show how the model can be used as a *late estimator* of leakage for real (placed and routed) circuits. In this test, we have *extracted* the relevant high-level characteristics from each ISCAS85 circuit, namely the number of gates used, the histogram of cells used, and the dimensions of the layout; then with these values, we have used our model to estimate the leakage statistics of every circuit. Table 7.1 lists the errors in the full-chip leakage standard deviation, for all ISCAS85 circuits, between our model and the true leakage of these circuits. The errors are very small (notice, however, that these do not include any cell leakage modeling errors, which were discussed earlier in Section 7.5). We do not show the errors in the mean leakage because they are truly negligible.

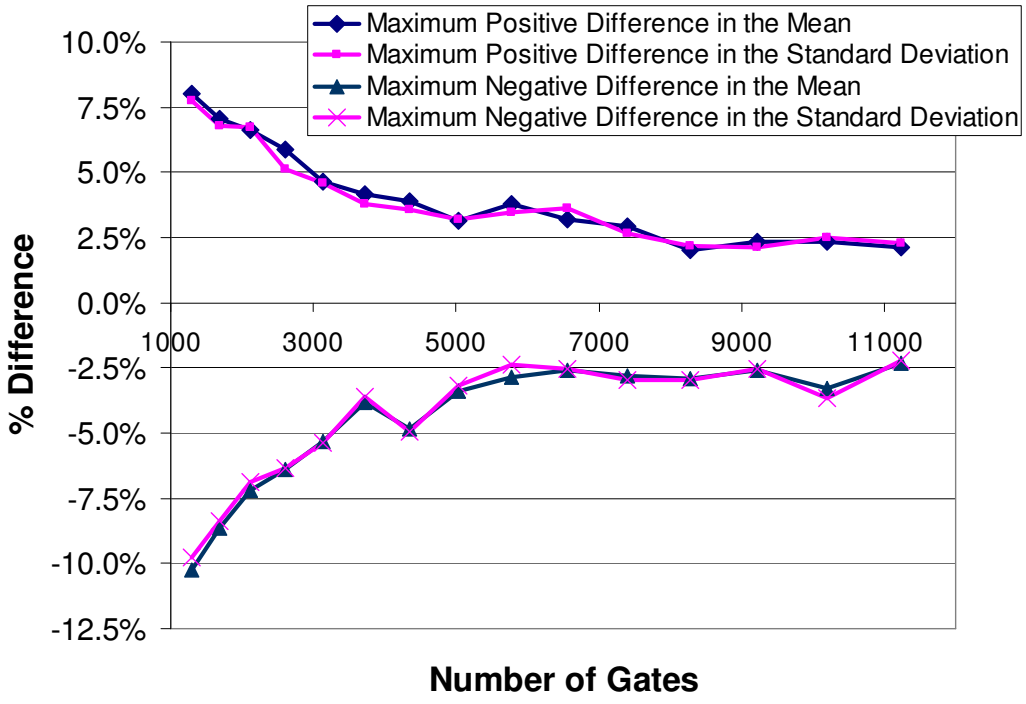


Figure 7.9: Errors in the estimation of mean and standard deviation of full-chip leakage

Simplified Correlation Assumption

In Section 7.5 we noted that the cell leakage statistics (*i.e.*, the mean and standard deviation of leakage) can be obtained in two ways; either (1) a MC analysis would be done or (2) the cell's leakage would be fitted into a functional form to get three fitting parameters (a, b, c). Using these parameters, the leakage mean and standard deviation were analytically obtained. The fitted parameters also allowed us to determine the leakage correlation between any pair of gates, $\rho_{m,n}$, given the channel length correlation ρ_L. Using the mapping, $f_{m,n}(\cdot)$, the RG leakage correlation was determined in (7.24).

If we, however, choose to obtain the leakage statistics of each cell through MC analysis, we would not be able to use $f_{m,n}(\cdot)$ to determine the leakage correlation between pairs of cells because the correlation mapping depends on the fitting parameters, which are not available in MC mode. Without this mapping, the RG leakage correlation cannot be determined. The solution to this problem lies in Fig. 7.5, where we have noted that the leakage correlation of any pair of cells is approximately equal to the correlation in the channel length of these cells. In other words, $\rho_{m,n} \approx \rho_L,\ \forall m, n$. With this simplified correlation assumption, (7.24) can be used to determine the RG leakage correlation.

To determine the amount of error introduced by this assumption, we have compared the difference between the standard deviation when assuming $\rho_{m,n} = \rho_L$ compared to the analytical approach, *i.e.*, when using the true $f_{m,n}(\cdot)$ mapping. Regardless of whether we assume solely WID variations or have both WID and D2D variations, the percentage error is below 2.8%, as shown in Fig. 7.10.

7.7.2 Constant-time method

In this section, we show how, for large values of n, we can approximate the linear summation in (7.31) by an integral to obtain the statistics of full-chip leakage in constant time. This transformation is possible because the correlation function that shows up under the integral (as shown next) is a well-behaved monotonically decreasing function.

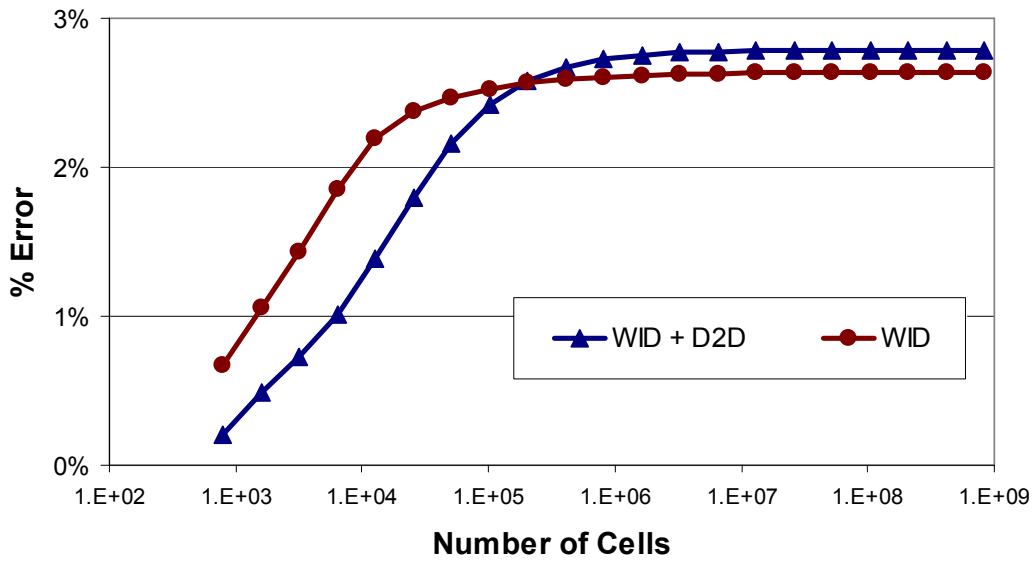


Figure 7.10: % Error in leakage standard deviation for $\rho_{m,n} = \rho_L$ compared to $\rho_{m,n} = f_{m,n}(\rho_L)$

2D Integration in Rectangular Coordinates

Starting from (7.31), let $x_i = i \cdot \Delta W$ and $y_j = j \cdot \Delta H$, and by multiplying out ΔW and ΔH we obtain:

$$\sigma^2_{\mathbf{I}_T} = \frac{\sigma^2_{\mathbf{X_I}}}{\Delta W \Delta H} \sum_{i=-m}^{m} \sum_{j=-k}^{k} (W - |x_i|) \cdot (H - |y_j|)\ \rho_{\mathbf{X_I}}(d_{ij}) \tag{7.32}$$

where $W = m \cdot \Delta W$, $H = k \cdot \Delta H$, and $d_{ij} = \sqrt{x_i^2 + y_j^2}$. By using a double integral to approximate the double summation over discrete values, we obtain:

$$\sigma^2_{\mathbf{I}_T} \approx \frac{\sigma^2_{\mathbf{X_I}}}{(\Delta W \Delta H)^2} \int_{-W}^{W} \int_{-H}^{H} (W-|x|)(H-|y|)\rho_{\mathbf{X_I}}\left(\sqrt{x^2+y^2}\right) \mathrm{d}y \mathrm{d}x \tag{7.33}$$

Let the area of a RG site be $\mathcal{A}_{\text{site}} = \Delta W \Delta H$ and the area of the die be $\mathcal{A} = n\mathcal{A}_{\text{site}}$.

Note that the function being integrated is even, so that we can write:

$$\sigma^2_{\mathbf{I}_T} \approx 4\sigma^2_{\mathbf{X_I}} \frac{n^2}{\mathcal{A}^2} \int_0^W \int_0^H (W-x)(H-y)\rho_{\mathbf{X_I}}\left(\sqrt{x^2+y^2}\right) \mathrm{d}y\mathrm{d}x \tag{7.34}$$

The expression in (7.34) approximates the full-chip leakage variance for large values of n. Since the number of gates on the chip is typically in the order of millions, the approximation is valid in most cases. What is interesting about this expression is that it only requires the computation of an integral, which can be performed in constant-time using a good numerical integration routine; the leakage variance computation does not depend on the number of gates n, it is $\mathcal{O}(1)$.

1D Integration in Polar Coordinates

To make our computation even more efficient, under certain conditions we can transform the double integral in (7.34) into a single integral in polar coordinates. First we write an exact mapping of (7.34) in double-integral form using polar coordinates:

$$\sigma^2_{\mathbf{I}_T} \approx \frac{4\sigma^2_{\mathbf{X_I}} n^2}{\mathcal{A}^2} \int_0^{\frac{\pi}{2}} \int_0^{D(\theta)} (W - r\cos\theta)(H - r\sin\theta)\rho_{\mathbf{X_I}}(r)\, r\mathrm{d}r\mathrm{d}\theta \tag{7.35}$$

where $D(\theta)$ is the distance from the origin to the boundary of the rectangular integration domain, which is less than the largest distance on the array. If the distance at which the WID correlation function reaches 0 is less than the minimum of the height or width of the array, then the double integral in (7.35) can be written as a single integral. To derive this single integral, let us for the moment assume that there are no D2D variations and that $\rho_{\mathbf{X_I}}$ becomes zero at a distance $D_{\max}$. If $D_{\max}$ is less than $\min(W, H)$ then (7.34) can be written as:

$$\sigma^2_{\mathbf{I}_T} \approx \frac{4\sigma^2_{\mathbf{X_I}} n^2}{\mathcal{A}^2} \int_0^{D_{\max}} \int_0^{\frac{\pi}{2}} (W - r\cos\theta)(H - r\sin\theta)\rho_{\mathbf{X_I}}(r)\, r\mathrm{d}\theta\mathrm{d}r \tag{7.36}$$

Since the correlation function does not depend on θ, we can further simplify the above expression by separating the integrals:

$$\sigma^2_{\mathbf{I}_T} \approx \frac{4\sigma^2_{\mathbf{X_I}} n^2}{\mathcal{A}^2} \int_0^{D_{\max}} \rho_{\mathbf{X_I}}(r) r \left[\int_0^{\frac{\pi}{2}} (W - r\cos\theta)(H - r\sin\theta) \mathrm{d}\theta \right] \mathrm{d}r \tag{7.37}$$

The expression in the brackets can be analytically integrated and results in the following expression:

$$g(r) \;=\; 0.5r^2 - (W+H)r + \frac{\pi}{2}WH \tag{7.38}$$

which leads to the final expression for full-chip leakage variance:

$$\sigma^2_{\mathbf{I}_T} \;\approx\; \frac{4\sigma^2_{\mathbf{X_I}} n^2}{\mathcal{A}^2} \int_0^{D_{\max}} r \cdot g(r) \cdot \rho_{\mathbf{X_I}}(r) \mathrm{d}r \tag{7.39}$$

When also considering D2D variations, recall from Section 7.4 that the correlation never reaches zero, and thus the single integral technique does not immediately apply. However, if we divide up the correlation function $\rho_{\mathbf{X_I}}(r)$ into a constant portion, ρ_c, and a portion that does go to 0 at $D_{\max}$, $\rho'_{\mathbf{X_I}}(r) = \rho_{\mathbf{X_I}}(r) - \rho_c$, then the single integral can be written as:

$$\sigma^2_{\mathbf{I}_T} \approx \left[\frac{4\sigma^2_{\mathbf{X_I}} n^2}{\mathcal{A}^2} \int_0^{D_{\max}} r \cdot g(r) \cdot \rho'_{\mathbf{X_I}}(r) \mathrm{d}r \right] + \rho_c \sigma^2_{\mathbf{X_I}} n^2 \tag{7.40}$$

Validation

The value of the standard deviation of the full-chip leakage obtained from the numerical integration (7.34) was compared to the value obtained from the $\mathcal{O}(n)$ approach presented in Section 7.7.1.

As can be seen in Fig. 7.11, for circuits that have more than ten thousand gates there is less than 0.01% error between the numerical integration and that of the linear-time algorithm. For circuits with a small number of gates (<100) the % error is more than 1%; this is due to the granularity of the gates being a significant proportion of the total area of the design causing the integral to be less accurate than the true sum. For

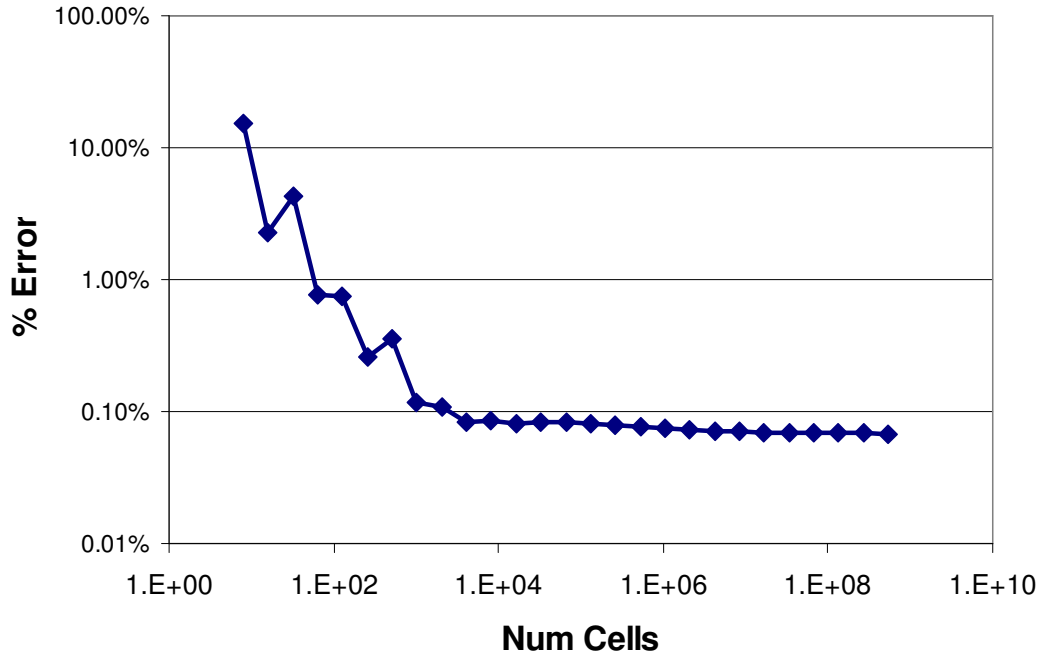


Figure 7.11: % Error between numerical integration and linear time algorithm

larger designs, the area of the logic gates compared to the area of the design approaches zero, allowing the numerical integration to provide good results, with less than 0.1% error.

Given that the $\mathcal{O}(n)$ time algorithm takes less than one second for circuits with less than 1000 gates, one can use the $\mathcal{O}(n)$ time algorithm in those cases, and use the numerical integration for circuits with a much larger number of gates.

7.8 Summary

In this chapter, we presented a probabilistic full-chip model that can be used to estimate, in constant-time, the leakage statistics of candidate designs either at an early or a late stage, while considering within-die correlations. We proposed and verified that certain high-level characteristics of a candidate chip design are sufficient to determine its leakage. These high-level characteristics, shown in Fig. 7.1, include information about the process, the standard-cell library, and the design in question. We showed that, for large gate count, the set of all chip designs that share the same high level characteristics have approximately the *same* full-chip leakage statistics, with very small error. We capture this set by a full-chip model based on Random Gates (RGs).

8 Conclusion

Continued technology scaling trends have led to many challenges that are facing digital integrated circuits in current process technologies. With the impact of increased process and environmental variability on circuit timing and leakage power reaching alarming levels, it is becoming increasingly difficult to design circuits that will achieve their target frequency and power budgets. With traditional corner-case analysis becoming too expensive, pessimistic, and unable to handle within-die variations, there is a clear need for developing new techniques that can analyze and predict circuit performance in the presence of variability. These techniques would be used not only to quantify the impact of variability on timing and leakage power, but also to drive circuit design and optimization in order to achieve robust designs with high yield.

In this thesis, we have presented different techniques for variation-aware timing and leakage analysis. On the timing front, we have considered different aspects of the problem, including handling within-die spatial correlations at an early stage of design, modeling random process parameters with arbitrary distributions as well as uncertain non-random environmental parameters, allowing for different types of variational delay models including linear and quadratic models, and capturing circuit timing under variability exactly at every point in the parameter space. For example, Chapter 3 presented a pre-placement statistical static timing analysis technique that can operate at an early stage of design, when within-die correlations are still unknown, and can produce a margin range that is valid under any arbitrary correlation. In Chapter 4, a general framework for parameterized static timing analysis was proposed, allowing for linear and quadratic delay models, and supporting both random and uncertain parameters in the same framework. In Chapter 5, another parameterized static timing analysis technique was presented, allowing to capture circuit delay exactly at any

point in the process and environmental parameter space by propagating forward all the potentially critical paths in the circuit. This was enabled using efficient pruning strategies, which can identify the critical paths and discard the non-critical ones. A novel distance-based metric for timing robustness in parameterized static timing analysis was proposed in Chapter 6. This metric can be used to quantify the susceptibility of parameterized timing quantities to failure, thus enabling designers to fix the nodes with smallest robustness values in order to improve the overall design robustness to variations.

Finally, on the leakage front, a probabilistic full-chip leakage estimation technique was presented which can be used as either an early or a late estimator of leakage statistics, with high accuracy. This was possible using the concept of *random gate* which captured certain high-level characteristics of a candidate design and allowed for an efficient full-chip leakage estimation. In its simplest form, the leakage estimation reduced to finding the area under a scaled version of the within-die channel length auto-correlation function, which can be done in constant time.

Collectively, the techniques that were presented in this thesis can fill real and diverse design needs. We believe that a natural place for these proposed algorithms would be in commercial or in-house CAD tools, such as static timing or power analysis and optimization engines. It is worth noting that the parameterized static timing and robustness analysis techniques that were proposed in Chapters 4, 5, and 6 were also implemented and successfully tested at Intel's Strategic CAD Labs as part of a static timing analyzer used for research purposes.

Although the techniques presented in this thesis constitute a step forward, we believe that the problem of variation-aware performance verification is far from solved, and further research will be needed as more challenges may arise in the future, especially as technology continues to scale further toward the physical process limits. In addition, more research is needed on variation-aware performance optimization to answer the question of *what to fix* to produce designs that would meet their target specification with high yield.

Appendices

A Leakage Statistics using Analytical Method

In this appendix, we present the mathematical framework which allows us to analytically determine the mean and standard deviation of cell leakage, given the fitted functional form with the triplet (a, b, c). This framework can be applied for any leakage model that is quadratic exponential (which includes both sub-threshold and gate leakage). Recall that the leakage of each cell in the library is modeled as:

$$\mathbf{X} = a\, e^{b\mathbf{L}+c\mathbf{L}^2} \tag{A.1}$$

We are interested in determining the mean $\mu_{\mathbf{X}}$ and variance $\sigma^2_{\mathbf{X}}$ of the cell leakage, $\mathbf{X}$, given the mean and variance of the channel length, $\mathbf{L}$, and the regression parameters (a, b, c). Assume that $\mathbf{L}$ is a normally distributed RV with mean μ and standard deviation σ. Let $\mathbf{Y} = \ln \mathbf{X}$; then $\mathbf{X} = e^{\mathbf{Y}}$. The mean and variance of $\mathbf{X}$ can be written as follows:

$$\mu_{\mathbf{X}} = E[\mathbf{X}] = E\left[e^{\mathbf{Y}}\right] \tag{A.2}$$

$$\sigma^2_{\mathbf{X}} = E\left[\mathbf{X}^2\right] - \mu^2_{\mathbf{X}} = E\left[e^{2\mathbf{Y}}\right] - \mu^2_{\mathbf{X}} \tag{A.3}$$

Let $M_{\mathbf{Y}}(t)$ be the moment-generating function of $\mathbf{Y}$. By definition, this function is equal to:

$$M_{\mathbf{Y}}(t) = E\left[e^{t\mathbf{Y}}\right] \tag{A.4}$$

This function has been studied in the literature, and has a closed form expression for most known distributions [24]. Note that (A.2) and (A.3) can be written in terms

of the moment-generating function of $\mathbf{Y}$:

$$\mu_{\mathbf{X}} = M_{\mathbf{Y}}(1) \tag{A.5}$$
$$\sigma^2_{\mathbf{X}} = M_{\mathbf{Y}}(2) - \mu^2_{\mathbf{X}} \tag{A.6}$$

The above result shows that the mean and variance of the cell leakage can be determined if $M_{\mathbf{Y}}(t)$ is known. To do that, we must determine the distribution of $\mathbf{Y}$. Since $\mathbf{Y} = \ln \mathbf{X}$, it follows from (A.1) that:

$$\mathbf{Y} = \ln a + b\mathbf{L} + c\mathbf{L}^2 \tag{A.7}$$

where $\mathbf{L}$ has a normal distribution. Let $\hat{\mathbf{L}}$ be a normalized version of $\mathbf{L}$:

$$\hat{\mathbf{L}} = \frac{\mathbf{L} - \mu}{\sigma} \tag{A.8}$$

This last equation shows that $\hat{\mathbf{L}}$ has a standard normal distribution with zero mean and unit variance. We can easily write (A.7) in the following form:

$$\mathbf{Y} = K_1 \left(\hat{\mathbf{L}} + K_2\right)^2 + K_3 \tag{A.9}$$

where:

$$K_1 = c\,\sigma^2 \tag{A.10}$$
$$K_2 = \frac{\left(\frac{b}{2c} + \mu\right)}{\sigma} \tag{A.11}$$
$$K_3 = \ln a + b\mu + c\,\mu^2 - c\left(\frac{b}{2c} + \mu\right)^2 \tag{A.12}$$

The motivation behind this transformation is to write $\mathbf{Y}$ in terms of an RV with a known distribution. Let $\mathbf{W} = \hat{\mathbf{L}} + K_2$; then $\mathbf{W}^2$ has a "Non-Central Chi-square" distribution with $\nu = 1$ degrees of freedom because $\mathbf{W}$ is normal with non-zero mean and unit variance. Therefore we can write (A.9) in terms of $\mathbf{W}^2$:

$$\mathbf{Y} = K_1\mathbf{W}^2 + K_3 \tag{A.13}$$

This allows to write the moment-generating function of $\mathbf{Y}$ in terms of the moment-generating function of $\mathbf{W}^2$ as follows:

$$\begin{aligned} M_{\mathbf{Y}}(t) &= E\left[e^{t\mathbf{Y}}\right] && \text{(A.14)}\\ &= E\left[e^{t\left(K_1\mathbf{W}^2+K_3\right)}\right] && \text{(A.15)}\\ &= e^{tK_3}E\left[e^{tK_1\mathbf{W}^2}\right] && \text{(A.16)}\\ &= e^{tK_3}M_{\mathbf{W}^2}(K_1t) && \text{(A.17)} \end{aligned}$$

Since $\mathbf{W}^2$ has a Non-Central Chi-square distribution with $\nu = 1$ degrees of freedom, then its moment-generating function is known [24]:

$$M_{\mathbf{W}^2}(t) = (1-2t)^{-\frac{1}{2}} \cdot e^{\frac{\lambda t}{1-2t}} \tag{A.18}$$

where $\lambda = (K_2)^2$ is the non-centrality parameter.

Now by using the above equation for the moment-generating function of $\mathbf{W}^2$, we can determine the moment-generating function of $\mathbf{Y}$ using (A.17) and get the final expression in (7.14), from which the mean and variance of the cell leakage can be determined as shown in (A.5) and (A.6).

To determine the accuracy of this analytical technique, we have compared its results to MC analysis. Histograms of the the percent error in the mean and standard deviation are shown in Fig. A.1 and A.2 respectively for all 62 cells with all input combinations. For the mean, the analytical method is quite close to the MC results with errors less than a 2% for all gates. For the standard deviation the error is larger, with an average absolute error of 3.1% and a maximum error of about 10%. As mentioned in Section 7.5.1, the error in the mean and standard deviation is not a result of the mathematical derivation, but due to the leakage curve not being exactly mapped to (A.1).

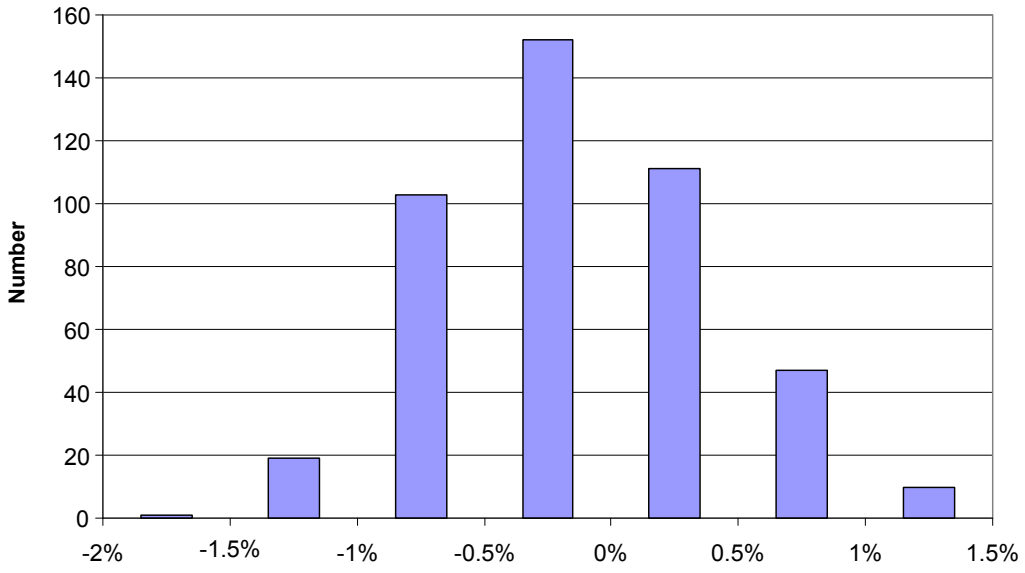


Figure A.1: Histogram of the % error in the mean of the analytical method compared to MC

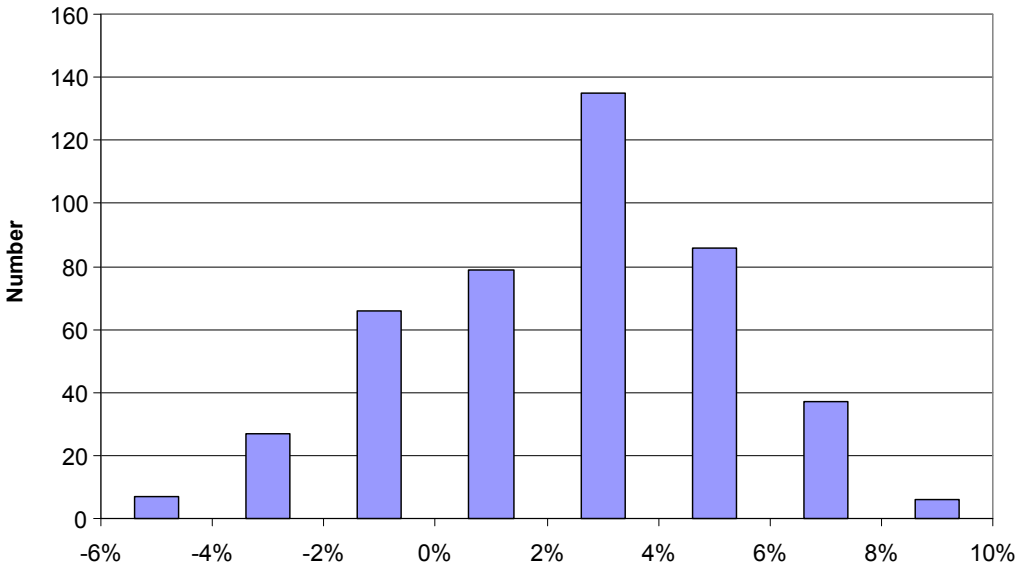


Figure A.2: Histogram % error in the standard deviation of the analytical method compared to MC

B Leakage Correlation using Analytical Mapping

In this appendix, we present the mathematical framework that allows us to analytically determine the correlation in the leakage currents of two cells given the correlation in their channel length. We also note that leakage correlation turns out to be very close to channel length correlation, in most cases.

Let $\mathbf{L}_1$ and $\mathbf{L}_2$ be two correlated RVs representing channel length at two arbitrary locations l_1 and l_2. We will assume that the correlation in channel length, $\rho_L(l_1, l_2)$, can be determined from the correlation model that we presented in Section 7.4.2. Recall that $\mathbf{L}_1$ and $\mathbf{L}_2$ are normally distributed with mean μ and standard deviation σ.

We are interested in determining $\rho_{m,n}(l_1, l_2)$ defined as the correlation in the leakage of two gates m and n given the channel length correlation $\rho_L(l_1, l_2)$. Particularly, we will use the analysis that follows to find a mapping $f_{m,n}(\cdot)$ such that:

$$\rho_{m,n}(l_1, l_2) = f_{m,n}\left(\rho_L(l_1, l_2)\right) \tag{B.1}$$

Let $\mathbf{X}_m$ and $\mathbf{X}_n$ be the leakage of two gates of type m and n from the library; these RVs depend respectively on $\mathbf{L}_1$ and $\mathbf{L}_2$ in the following way:

$$\mathbf{X}_m = a_1 e^{b_1 \mathbf{L}_1 + c_1 \mathbf{L}_1^2} \tag{B.2}$$

$$\mathbf{X}_n = a_2 e^{b_2 \mathbf{L}_2 + c_2 \mathbf{L}_2^2} \tag{B.3}$$

It is important to understand that the channel length and leakage correlations are due to the spatial correlation between the locations l_1 and l_2, and depend particularly on the distance between the two locations. Let $C_{m,n}(l_1, l_2)$ be the covariance of $\mathbf{X}_m$

and $\mathbf{X}_n$ defined as follows:

$$C_{m,n}(l_1, l_2) = E\left[\mathbf{X}_m \mathbf{X}_n\right] - \mu_{\mathbf{X}_m} \mu_{\mathbf{X}_n} \tag{B.4}$$

The leakage correlation $\rho_{m,n}(l_1, l_2)$ can be expressed as a function of the covariance:

$$\rho_{m,n}(l_1, l_2) = \frac{C_{m,n}(l_1, l_2)}{\sigma_{\mathbf{X}_m} \cdot \sigma_{\mathbf{X}_n}} \tag{B.5}$$

where $\mu_{\mathbf{X}_m}$, $\mu_{\mathbf{X}_n}$, $\sigma_{\mathbf{X}_m}$, $\sigma_{\mathbf{X}_n}$ denote the means and standard deviations of $\mathbf{X}_m$ and $\mathbf{X}_n$ respectively, as determined in Section 7.5.1.

Examining (B.5) and (B.4), it is easy to see that the problem of finding $\rho_{m,n}(l_1, l_2)$ can be solved if $E\left[\mathbf{X}_m \mathbf{X}_n\right]$ is determined. By letting $\mathbf{Y} = \ln\left(\mathbf{X}_m \mathbf{X}_n\right)$, we can write $E\left[\mathbf{X}_m \mathbf{X}_n\right]$ as a function of the moment-generating function of $\mathbf{Y}$:

$$\begin{aligned} E\left[\mathbf{X}_m \mathbf{X}_n\right] &= E\left[e^{\mathbf{Y}}\right] && \text{(B.6)} \\ &= M_{\mathbf{Y}}(1) && \text{(B.7)} \end{aligned}$$

since $M_{\mathbf{Y}}(t) = E\left[e^{t\mathbf{Y}}\right]$.

In this way, if we are able to determine the moment-generating function of $\mathbf{Y}$, we can evaluate it at 1 to determine $E\left[\mathbf{X}_m \mathbf{X}_n\right]$ using (B.7). Then, using (B.4) and (B.5), we can determine the leakage correlation $\rho_{m,n}(l_1, l_2)$. Using (B.2) and (B.3), we can write:

$$\begin{aligned} \mathbf{Y} &= \ln\left(\mathbf{X}_m \mathbf{X}_n\right) \\ &= \ln a_1 + \ln a_2 + b_1 \mathbf{L}_1 + b_2 \mathbf{L}_2 + c_1 \mathbf{L}_1^2 + c_2 \mathbf{L}_2^2 \end{aligned} \tag{B.8}$$

Assume that the correlation in the channel lengths $\mathbf{L}_1$ and $\mathbf{L}_2$ is $\rho_L(l_1, l_2) = \rho$. To model this correlation, we will use the following transformation; Let $\mathbf{Z}_1$ and $\mathbf{Z}_2$ be two

RVs defined as follows:

$$\mathbf{Z}_1 = \frac{1}{2\alpha}\left(\frac{\mathbf{L}_1-\mu}{\sigma}+\frac{\mathbf{L}_2-\mu}{\sigma}\right) \tag{B.9}$$

$$\mathbf{Z}_2 = \frac{1}{2\beta}\left(\frac{\mathbf{L}_1-\mu}{\sigma}-\frac{\mathbf{L}_2-\mu}{\sigma}\right) \tag{B.10}$$

where:

$$\alpha = \sqrt{\frac{1+\rho}{2}} \tag{B.11}$$

$$\beta = \sqrt{1-\alpha^2} = \sqrt{\frac{1-\rho}{2}} \tag{B.12}$$

The way they are defined above, $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are guaranteed to have certain properties. First, they are normally distributed since $\mathbf{L}_1$ and $\mathbf{L}_2$ are jointly normally distributed. In addition, they are guaranteed to have zero-mean, unit variance, and zero correlation (or covariance). This can be easily shown:

$$E\left[\mathbf{Z}_1\right] = \frac{1}{2\alpha}\left(E\left[\frac{\mathbf{L}_1-\mu}{\sigma}\right]+E\left[\frac{\mathbf{L}_2-\mu}{\sigma}\right]\right) = 0 \tag{B.13}$$

$$\begin{aligned}\mathrm{Var}(\mathbf{Z}_1) &= \frac{1}{4\alpha^2}\left(\frac{\mathrm{Var}\left(\mathbf{L}_1\right)}{\sigma^2}+\frac{\mathrm{Var}\left(\mathbf{L}_2\right)}{\sigma^2}+2\frac{\mathrm{Cov}\left(\mathbf{L}_1,\mathbf{L}_2\right)}{\sigma^2}\right)\\ &= \frac{2}{4(1+\rho)}\left(1+1+2\rho\right) = 1\end{aligned} \tag{B.14}$$

$$\begin{aligned}\mathrm{Cov}(\mathbf{z}_1,\mathbf{z}_2) &= \frac{1}{4\alpha\beta}E\left[\left(\frac{\mathbf{L}_1-\mu}{\sigma}+\frac{\mathbf{L}_2-\mu}{\sigma}\right)\left(\frac{\mathbf{L}_1-\mu}{\sigma}-\frac{\mathbf{L}_2-\mu}{\sigma}\right)\right]\\ &= \frac{1}{4\alpha\beta}\left(E\left[\left(\frac{\mathbf{L}_1-\mu}{\sigma}\right)^2\right]-E\left[\left(\frac{\mathbf{L}_2-\mu}{\sigma}\right)^2\right]\right) = 0\end{aligned} \tag{B.15}$$

Being normally distributed with zero-mean and unit variance, and having zero correlation, imply that $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are *independent standard normal* RVs. By reordering (B.9) and (B.10), we can express both $\mathbf{L}_1$ and $\mathbf{L}_2$ as a function of $\mathbf{Z}_1$ and $\mathbf{Z}_2$ as

follows:

$$\mathbf{L}_1 = \sigma(\alpha\mathbf{Z}_1 + \beta\mathbf{Z}_2) + \mu \tag{B.16}$$

$$\mathbf{L}_2 = \sigma(\alpha\mathbf{Z}_1 - \beta\mathbf{Z}_2) + \mu \tag{B.17}$$

By substituting (B.16) and (B.17) in (B.8), we can write $\mathbf{Y}$ in the following matrix form:

$$\mathbf{Y} = K_1 + K_2 + \begin{bmatrix} K_3 & K_4 \end{bmatrix} \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{Z}_1 & \mathbf{Z}_2 \end{bmatrix} \begin{bmatrix} K_5 & K_6 \\ K_7 & K_8 \end{bmatrix} \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} \tag{B.18}$$

where:

$$K_1 = \ln a_1 + b_1\mu + c_1\mu^2 \tag{B.19}$$

$$K_2 = \ln a_2 + b_2\mu + c_2\mu^2 \tag{B.20}$$

$$K_3 = \alpha\sigma\left[(b_1 + b_2) + 2\mu(c_1 + c_2)\right] \tag{B.21}$$

$$K_4 = \beta\sigma\left[(b_1 - b_2) + 2\mu(c_1 - c_2)\right] \tag{B.22}$$

$$K_5 = \alpha^2\sigma^2(c_1 + c_2) \tag{B.23}$$

$$K_6 = \alpha\beta\sigma^2(c_1 - c_2) \tag{B.24}$$

$$K_7 = K_6 \tag{B.25}$$

$$K_8 = \beta^2\sigma^2(c_1 + c_2) \tag{B.26}$$

Generally, K_6 and K_7 are non-zero, which will lead to cross terms when performing the matrix multiplication in (B.18) (*i.e.,* terms in $\mathbf{Z}_1\mathbf{Z}_2$). These terms will complicate the determining of $\mathbf{Y}$ and we would ideally like to remove them from the expression. This can be achieved through matrix diagonalization; let's denote the 2×2 matrix in (B.18) by A. Because A is symmetric, we can diagonalize A in the following way:

$$A = PDP^T \tag{B.27}$$

where D is a diagonal matrix having as entries the eigenvalues of A, *i.e.,* $\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$ and P is a matrix having as columns the eigenvectors of A. Since A is symmetric, these

eigenvectors are *orthogonal*. Moreover, we can choose P in such a way that its columns are also *orthonormal*. This decomposition is a standard practice and we use it here to transform $\mathbf{Z}_1$ and $\mathbf{Z}_2$ into a new set of RVs $\mathbf{V}_1$ and $\mathbf{V}_2$ that are also independent standard normals:

$$\begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \end{bmatrix} = P^T \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} \tag{B.28}$$

Using the above transformation, (B.18) is written as follows:

$$\mathbf{Y} = K_1 + K_2 + \begin{bmatrix} \hat{K}_3 & \hat{K}_4 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_2 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \end{bmatrix} \tag{B.29}$$

where:

$$\begin{bmatrix} \hat{K}_3 \\ \hat{K}_4 \end{bmatrix} = P^T \begin{bmatrix} K_3 \\ K_4 \end{bmatrix} \tag{B.30}$$

Note that both D and P can be easily determined for a 2×2 symmetric matrix as there is a closed form expression for the eigenvalues and eigenvectors of A. Note also that since the off-diagonal entries of D are zero, $\mathbf{Y}$ will have no cross terms.

Now that we removed the cross terms, we can write $\mathbf{Y}$ into the following quadratic form:

$$\begin{aligned} \mathbf{Y} &= K_1 + K_2 + \hat{K}_3\mathbf{V}_1 + \hat{K}_4\mathbf{V}_2 + \lambda_1\mathbf{V}_1^2 + \lambda_2\mathbf{V}_2^2 \\ &= (K_1 + \hat{K}_3\mathbf{V}_1 + \lambda_1\mathbf{V}_1^2) + (K_2 + \hat{K}_4\mathbf{V}_2 + \lambda_2\mathbf{V}_2^2) \end{aligned} \tag{B.31}$$

where $\mathbf{V}_1$ and $\mathbf{V}_2$ are independent standard normal RVs.

Note that the above equation is essentially two instances of (A.7). Using exactly the same analysis that follows (A.7), we can write $\mathbf{Y}$ in terms of two independent RVs that have a non-central chi-square distribution; this allows us to determine the moment generating function of $\mathbf{Y}$, $M_{\mathbf{Y}}(t)$. Once this is done, we use (B.7) to find $E[\mathbf{X}_m\mathbf{X}_n]$, and consequently determine $\rho_{m,n}(l_1, l_2)$ from (B.4) and (B.5).

The above analysis, whereby the leakage correlation between any pair of gates placed at two arbitrary locations can be determined given the correlation in the channel length

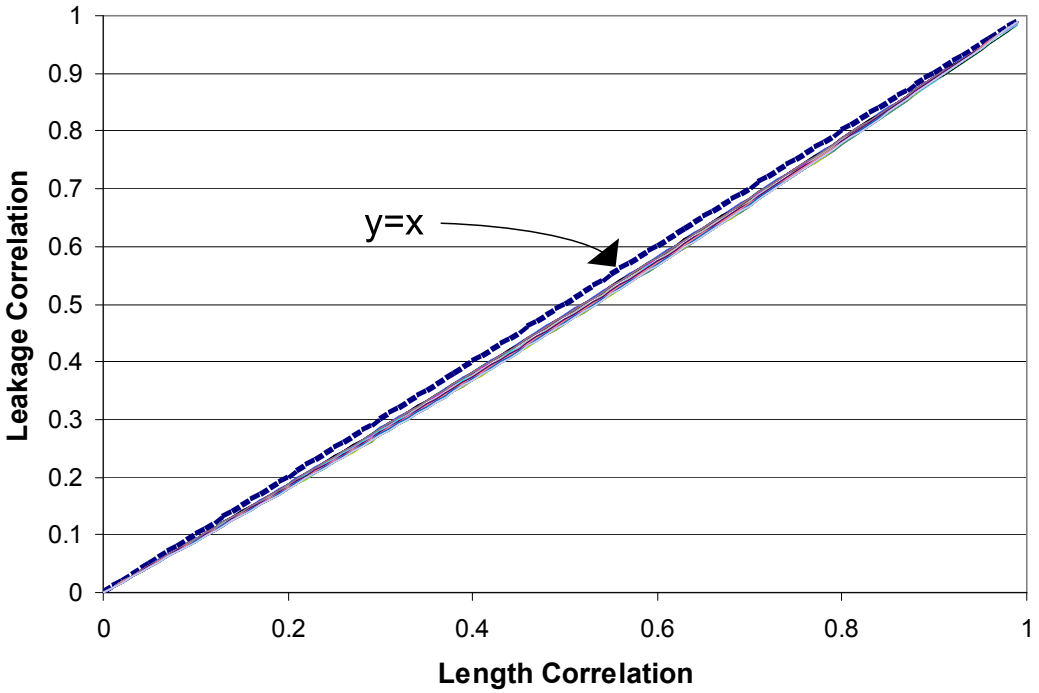


Figure B.1: Leakage correlation of pairs of different gates

at these two locations, is referred to as the mapping $f_{m,n}(\cdot)$:

$$\rho_{m,n}(l_1, l_2) = f_{m,n}\left(\rho_L(l_1, l_2)\right) \tag{B.32}$$

We have used this mapping to determine the leakage correlation between the pairs of cells in our library. The results obtained for all pairs of gates, while being close to each other, are not exactly the same as can be seen in Fig. B.1, where the correlation of 63 pairs of gates are plotted compared to the $y = x$ line. The difference can be better seen in a zoomed version of the plot in Fig. B.2. The resulting curves are convex functions that pass through $(0, 0)$ and $(1, 1)$; they closely follow the $y = x$, deviating slightly at $\rho_L(l_1, l_2) = 0.5$.

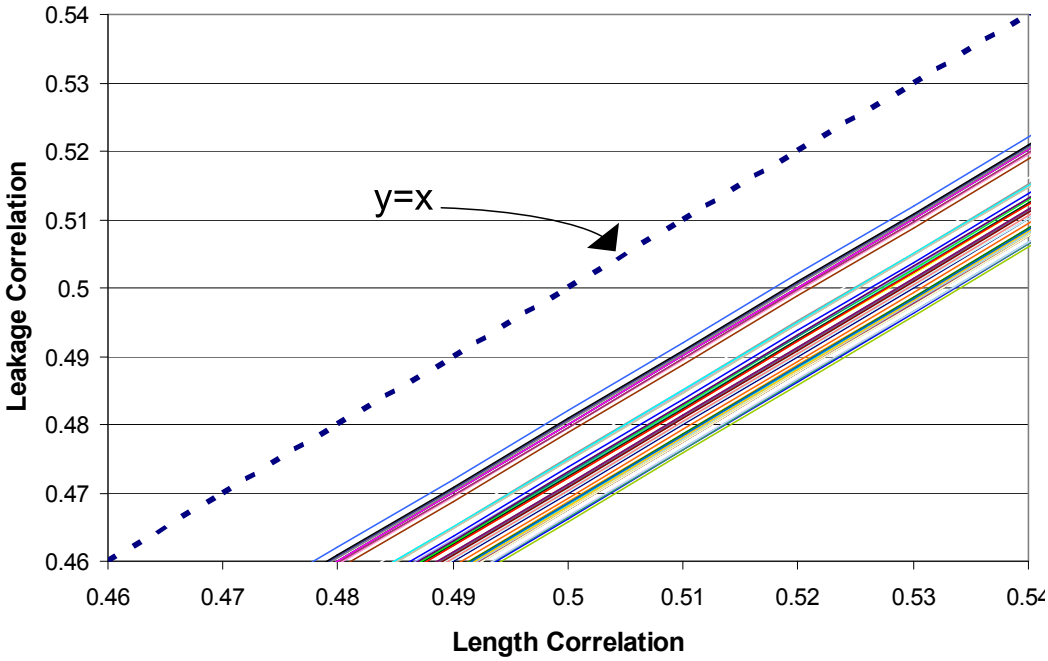


Figure B.2: Leakage correlation of pairs of different gates (zoom in)

References

[1] N. Azizi. *Challenges in Nanometre Digital Integrated Circuit Design*. PhD thesis, University of Toronto, 2007.

[2] D.J. Frank and H.S.P. Wong. Simulation of stochastic doping effects in Si MOSFETs. In *International Workshop on Computational Electronics*, pages 2–3, 2000.

[3] Semiconductor Industry Association. *International Techonology Roadmap for Semiconductors*. Available at: `http://public.itrs.net/`.

[4] K.R. Heloue and F.N. Najm. Early Statistical Timing Analysis with Unknown Within-Die Correlations. In *IEEE TAU Workshop, Austin*, 2007.

[5] K. R. Heloue and F. N. Najm. Parameterized timing analysis with general delay models and arbitrary variation sources. In *Design Automation Conference*, pages 403–408, Anaheim, CA, June 8–13 2008.

[6] K. R. Heloue, S. Onaissi, and F. N. Najm. Efficient block-based parameterized timing analysis covering all potentially critical paths. In *ICCAD*, pages 173–180, 2008.

[7] K.R. Heloue, C.V. Kashyap, and F.N. Najm. Quantifying robustness metrics in parameterized static timing analysis. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 209–216. ACM, 2009.

[8] K.R. Heloue, N. Azizi, and F.N. Najm. Modeling and estimation of full-chip leakage current considering within-die correlation. *Design Automation Conference*, pages 93–98, 2007.

[9] K.R. Heloue, N. Azizi, and F.N. Najm. Full-chip model for leakage-current estimation considering within-die correlation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(6):874–887, 2009.

[10] D.S. Boning and S. Nassif. Models of process variations in device and interconnect. *Design of High Performance Microprocessor Circuits, chapter 6*, 2000.

[11] P. Zuchowski, P. Habitz, J. Hayes, and J. Oppold. Process and environmental variation impacts on ASIC timing. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 336–342, San Jose, CA, November 7-11 2004.

[12] S.R. Nassif. Design for variability in DSM technologies. In *IEEE/ACM International Symposium on Quality of Electronic Design*, pages 451–454, San Jose, CA, March 2000.

[13] J.W. McPherson. Reliability challenges for 45nm and beyond. In *Design Automation Conference*, pages 176–181, 2006.

[14] J. Luo, S. Sinha, Q. Su, J. Kawa, and Chiang. An IC manufacturing yield model considering intra-die variations. In *Design Automation Conference*, pages 749–754, 2006.

[15] P. Gupta, A.B. Kahng, D. Sylvester, and J. Yangt. Performance-driven OPC for mask cost reduction. In *International Symposium on Quality of Electronic Design*, pages 270–275, 2005.

[16] N.B. Cobb, A. Zakhor, and E.A. Miloslavsky. Mathematical and CAD framework for proximity correction. In *Proceedings of SPIE on Optical Microlithography*, pages 208–222, 1996.

[17] T. I Kirkpatrick and N. R. Clark. PERT as an aid to logic design. *IBM journal of Research and Development*, 10(2):135–141, March 1966.

[18] Robert B. Hitchcock, Sr. Gordon L. Smith, and David D. Cheng. Timing analysis of computer hardware. *IBM journal of Research and Development*, 26(1):100–105, January 1982.

[19] David Blaauw, Vladimir Zolotov, and Savithri Sundareswaran. Slope propagation in static timing analysis. *IEEE Transactions on Computer-Aided Design*, 21(10):1180–1195, October 2002.

[20] K. Singhal and V. Visvanathan. Statistical device models for worst case files and electrical test data. *IEEE Transactions on Semiconductor Manufacturing*, 12(4):470–484, November 1999.

[21] M. Sengupta, S. Saxena, L. Daldoss, G. Kramer, S. Minehane, and J. Cheng. Application specific worst-case corners using response surfaces and statistical models. In *IEEE International Symposium on Quality Electronic Design (ISQED)*, pages 351–356, San Jose, CA, March 22-24 2004.

[22] A. Gattiker, S. Nassif, R. Dinakar, and C. Long. Timing yield estimation from static timing analysis. In *IEEE International Symposium on Quality Electronic Design (ISQED)*, pages 437–442, San Jose, CA, March 26-28 2001.

[23] M. Eisele, J. Berthold, D. Schmitt-Landsiedel, and R. Mahnkopf. The impact of intra-die device parameter variations on path delays and on the design for yield of low voltage digital circuits. *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, 5(4):360–368, December 1997.

[24] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York, NY, 2nd edition, 1984.

[25] G. Casella and C.P. Robert. Monte Carlo statistical methods. *New York, Spring*, 2005.

[26] A. Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 607–614, San Jose, CA, November 9-13 2003.

[27] H. Chang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single PERT-like traversal. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 621–625, San Jose, CA, November 9-13 2003.

[28] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *ACM/IEEE Design Automation Conference*, pages 331–336, San Diego, CA, June 7-11 2004.

[29] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 900–907, San Jose, CA, November 9-13 2003.

[30] A. Agarwal, V. Zolotov, and D. T. Blaauw. Statistical timing analysis using bounds and selective enumeration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1243–1260, September 2003.

[31] J. A. G. Jess, K. Kalafala, W. R. Naidu, R. H. J. M. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. In *ACM/IEEE 40th Design Automation Conference (DAC-03)*, pages 932–937, Anaheim, CA, June 2-6 2003.

[32] M. Orshansky and A. Bandyopadhyay. Fast statistical timing analysis handling arbitrary delay correlations. In *ACM/IEEE Design Automation Conference (DAC-04)*, pages 337–342, San Diego, CA, June 7-11 2004.

[33] K. A. Bowman and J. D. Meindl. Impact of within-die parameter fluctuations on future maximum clock frequency distributions. In *IEEE Custom Integrated Circuits Conference (CICC)*, pages 229–232, 2001.

[34] K. A. Bowman, S. G. Duvall, and J. D. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of Solid-State Circuits*, 37(2):183–190, February 2002.

[35] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proceedings of the IEEE*, 91(2):305–327, 2003.

[36] Z. Chen, M. Johnson, L. Wei, and K. Roy. Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks. In *International symposium on Low power electronics and design*, pages 239–244. ACM, 1998.

[37] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula. Computation and refinement of statistical bounds on circuit delay. In *ACM/IEEE 40th Design Automation Conference (DAC-03)*, pages 348–353, Anaheim, CA, June 2-6 2003.

[38] S. Bhardwaj, S. B.K. Vrudhula, and D. Blaauw. tau: Timing analysis under uncertainty. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 615–620, San Jose, CA, November 9-13 2003.

[39] F. N. Najm and N. Menezes. Statistical timing analysis based on a timing yield model. In *ACM/IEEE Design Automation Conference (DAC-04)*, pages 460–465, San Diego, CA, June 7-11 2004.

[40] K. R. Heloue and F. N. Najm. Statistical timing analysis with two-sided constraints. In *IEEE/ACM International Conference on Computer Aided Design*, pages 829–836, San Jose, CA, November 6-10 2005.

[41] S. G. Duvall. Statistical circuit modeling and optimization. In *International Workshop on Statistical Metrology*, pages 56–63, June 2000.

[42] Y. L. Tong. *The multivariate normal distribution*. Springer-Verlag, New York, NY, 1990.

[43] C. E. Clark. The greatest of a finite set of random variables. In *Operaions Research*, pages 145–162, March-April 1961.

[44] D. Bryan. The ISCAS'85 Benchmark Circuits and Netlist Format. *North Carolina State University*, 1985.

[45] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah. Parameterized block-based statistical timing analysis with non-gaussian parameters, nonlinear delay models. In *DAC*, pages 71–76, Anaheim, CA, June 13-17 2004.

[46] Y. Zhan, A. J. Strojwas, X. Li, and L. T. Pileggi. Correlation-aware statistical timing analysis with non-gaussian delay distributions. In *Design Automation Conference*, pages 77–82, Anaheim, CA, June 13-17 2005.

[47] L. Zhang, W. Chen, Y. Hu, J.A. Gubner, and C.C.P. Chen. Correlation-preserved non-gaussian statistical timing analysis with quadratic timing model. *DAC*, pages 83–88, 2005.

[48] V. Khandelwal and A. Srivastava. A general framework for accurate statistical timing analysis considering correlations. *DAC*, pages 89–94, 2005.

[49] J. Singh and S. Sapatnekar. Statistical timing analysis with correlated non-gaussian parameters using independent component analysis. *DAC*, pages 155–160, 2006.

[50] L. Cheng, J. Xiong, and L. He. Non-linear statistical static timing analysis for non-Gaussian variation sources. *Proc. DAC*, pages 250–255, 2007.

[51] S. Onaissi. and F. N. Najm. A linear-time approach for static timing analysis covering all process corners. *ICCAD*, pages 217–224, 2006.

[52] S. V. Kumar, C. V. Kashyap, and S. S. Sapatnekar. A framework for block-based timing sensitivity analysis. In *Design Automation Conference*, pages 688–693, Anaheim, CA, June 8–13 2008.

[53] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[54] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[55] T. Ottmann, S. Schuierer, and S. Soundaralakshmi. Enumerating Extreme Points in Higher Dimensions. *Nordic Journal of Computing*, 8(2):179–192, 2001.

[56] MOSEK ApS. Available at: `http://www.mosek.com/`.

[57] J. Bandler and H. Abdel-Malek. Optimal centering, tolerancing, and yield determination via updated approximations and cuts. *IEEE Transactions on Circuits and Systems*, 25(10):853–871, 1978.

[58] K. K. Low and S. W. Director. A new methodology for the design centering of IC fabrication processes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(7):895–903, July 1991.

[59] R. Brayton, S. Director, and G. Hachtel. Yield maximization and worst-case design with arbitrary statistical distributions. *IEEE Transactions on Circuits and Systems*, 27(9):756–764, 1980.

[60] A. Schöbel. *Locating Lines and Hyperplanes: Theory and Algorithms*. Kluwer Academic Pub, 1999.

[61] S. Borkar. Design Challenges of Technology Scaling. *IEEE MICRO*, pages 23–29, 1999.

[62] A. Agarwal, C.H. Kim, S. Mukhopadhyay, and K. Roy. Leakage in nano-scale technologies: mechanisms, impact and design considerations. In *Proceedings of the 41st annual conference on Design automation*, pages 6–11. ACM New York, NY, USA, 2004.

[63] Amit Agarwal, Kunhyuk Kang, and Kaushik Roy. Accurate estimation and modeling of total chip leakage considering inter- & intra-die process variations. *IEEE International Conference on Computer-aided Design*, 2005.

[64] D. Lee, W. Kwong, D. Blaauw, and D. Sylvester. Simultaneous subthreshold and gate-oxide tunneling leakage current analysis in nanometer CMOS design. *ISQED*, pages 287–292, 2003.

[65] Siva Narendra, Vivek De, Dimitri Antoniadis, and Anantha Chandrakasan. Full-chip sub-threshold leakage power prediction model of sub-0.18μm CMOS. *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2002.

[66] Rajeev Rao, Ashish Srivastava, David Blaauw, and Dennis Sylvester. Statistical analysis of subthreshold leakage current for VLSI circuits. *IEEE Transactions on VLSI Systems*, 12(2):131–139, February 2004.

[67] Rajeev R. Rao, David Blaauw, Dennis Sylvester, and Anirudh Devgan. Modeling and analysis of parametric yield under power and performance constraints. *IEEE Design & Test of Computers*, pages 376–385, July-August 2005.

[68] Songqing Zhang, Vineet Wason, and Kaustav Banerjee. A probablisitic framework to estimate full-chip subthreshold leakage power distribution considering within-die and die-to-die P-T-V variations. *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2004.

[69] Hongliang Chang and Sachin S. Sapatnekar. Full-chip analysis of leakage power under process variations, inlcuding spatial correlations. *IEEE Design Automation Conference*, 2005.

[70] Jinjun Xiong, Vladimir Zolotov, and Lei He. Robust extraction of spatial correlation. *International Symposium on Physical Design*, 2006.

[71] A. Srivastava, R. Bai, D. Blaauw, and D. Sylvester. Modeling and analysis of leakage power considering within-die process variations. In *Proceedings of the 2002 international symposium on Low power electronics and design*, pages 64–67. ACM New York, NY, USA, 2002.

[72] Ali Keshavarzi, Gerhard Schrom, Stephen Tang, Sean Ma, Keith Bowman, Sunit Tyagi, Kevin Zhang, Tom Linton, Nagib Hakim, Steven Duvall, John Brews, and Vivek De. Measurements and modeling of intrinsic fluctuations in mosfet threshold voltage. *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2005.

[73] D. Helms, G. Ehmen, and W. Nebel. Analysis and modeling of subthreshold leakage of RT-components under PTV and state variation. *Proceedings of the 2006 international symposium on Low power electronics and design*, pages 220–225, 2006.

[74] J.J. Mor. The Levenberg-Marquardt algorithm: implementation and theory.