ABSTRACT

Title of dissertation: VARIABILITY-AWARE VLSI DESIGN AUTOMATION FOR NANOSCALE TECHNOLOGIES

Vishal Khandelwal, Ph.D., 2007

Dissertation directed by: Professor Ankur Srivastava
Department of Electrical and
Computer Engineering

As technology scaling enters the nanometer regime, design of large scale ICs gets more challenging due to shrinking feature sizes and increasing design complexity. Aggressive scaling causes significant degradation in reliability, increased susceptibility to fabrication and environmental randomness and increased dynamic and leakage power dissipation. In this work, we investigate these scaling issues in large scale integrated systems.

This dissertation proposes to develop variability-aware design methodologies by proposing design analysis, design-time optimization, post-silicon tunability and runtime-adaptivity based optimization techniques for handling variability. We discuss our research in the area of variability-aware analysis, specifically focusing on the problem of statistical timing analysis. The first technique presents the concept of error budgeting that achieves significant runtime speedups during statistical timing analysis. The second work presents a general framework for non-linear non-Gaussian statistical timing analysis considering correlations.

Further, we present our work on design-time optimization schemes that are applicable during physical synthesis. Firstly, we present a buffer insertion technique that considers wire-length uncertainty and proposes algorithms to perform probabilistic buffer insertion. Secondly, we present a stochastic optimization framework based on Monte-Carlo technique considering fabrication variability. This optimization framework can be applied to problems that can be modeled as linear programs without without imposing any assumptions on the nature of the variability.

Subsequently, we present our work on post-silicon tunability based design optimization. This work presents a design management framework that can be used to balance the effort spent on pre-silicon (through gate sizing) and post-silicon optimization (through tunable clock-tree buffers) while maximizing the yield gains. Lastly, we present our work on variability-aware runtime optimization techniques. We look at the problem of runtime supply voltage scaling for dynamic power optimization, and propose a framework to consider the impact of variability on the reliability of such designs. We propose a probabilistic design synthesis technique where reliability of the design is a primary optimization metric.

VARIABILITY-AWARE VLSI DESIGN AUTOMATION FOR NANOSCALE TECHNOLOGIES

by

Vishal Khandelwal

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:
Professor Ankur Srivastava, Chair/Advisor
Professor Joseph JaJa
Professor Samir Khuller
Professor Shuvra S. Bhattacharyya
Professor Kazuo Nakajima
Professor Gang Qu

DEDICATION

This dissertation is dedicated to my parents for their love, support and encouragement.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Prof. Ankur Srivastava for his guidance and help through my PhD. His advice and support has been crucial in the completion of this dissertation and other research projects that I have successfully completed at University of Maryland.

I also want to thank Professor Joseph JaJa, Professor Samir Khuller, Professor Shuvra Bhattacharyya, Professor Kazuo Nakajima and Professor Gang Qu for serving on the dissertation committee. Their advice and support in completing this dissertation is greatly appreciated.

I would also like to thank my uncle and aunt Dr. Basant and Rita Khandelwal for their constant support and encouragement in the years that I have been at Maryland.

I owe my gratitude to a lot of colleagues and friends with whom I have had very fruitful discussions about my research. Specifically, I want to thank my colleague Azadeh Davoodi and Ashish Dobhal for their help in several research projects. Lastly, I want to thank all my friends, specifically Amit Agrawal, Manish Shukla, Rahul Ratan, Ravi Tandon, Anuj Rawat, Abhishek Kashyap, Amrit Bandyopadhyay for various discussions and their unwavering support through my PhD.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Nanoscale VLSI Design Automation

1.1 Introduction

The technological advances in the last decade have enabled the emergence of the deep sub-micron (sub 250nm) and nanometer (sub 90nm) eras in VLSI Design. High performance and low cost ICs are a direct result of this growth. In 1965, Gordan Moore predicted that the number of transistors per IC would double every two years. In the last three decades, we have more than kept up with the famous *Moore's Law*. Intel, one of the key players in the IC/Microprocessor design field has scaled the number of transistors on an IC with each generation of its product keeping in pace with Moore's prediction. The Intel-286 series developed in 1982 had about a 130,000 transistors, which scaled to about 7,000,000 transistors in 1997 for the Intel Pentium II series, then scaled to about 300,000,000 transistors in 2003 for the Intel Itanium 2 Processor and is projected to hit the 1 billion transistor mark for its ongoing 64 bit microprocessor designs in the coming years.

This brings to light the increasing complexity in designing such ICs, that are faster, smaller and more powerful that their previous generations. The existing CAD tools need to adapt themselves to these new rising challenges posed in design automation. Design of high performance digital ICs has become an extremely challenging task. Shrinking device dimensions, increasing manufacturing and envi-

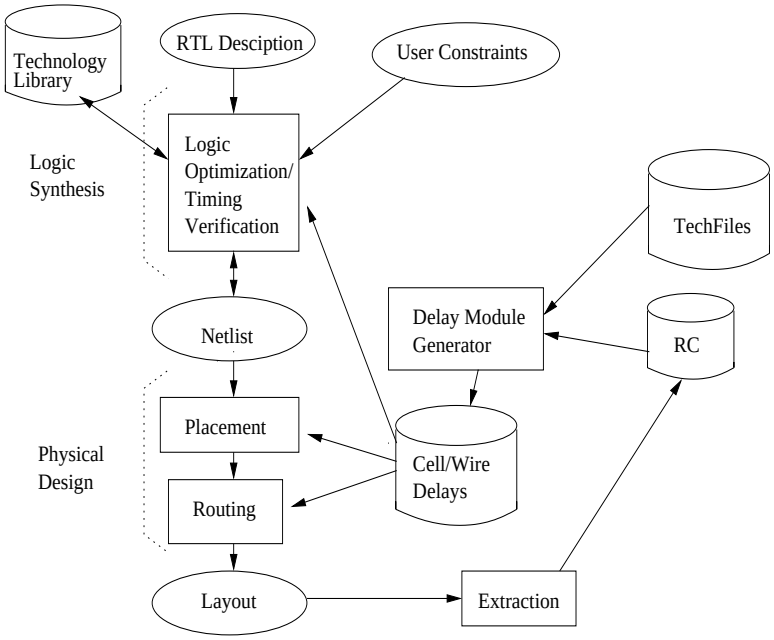


Figure 1.1: Typical VLSI Design Flow

ronmental variations has made fast design closure and high yield difficult.

The traditional design automation (DA) flow is rapidly having to adapt itself to these changes in Deep Sub-Micron (DSM) VLSI design. A typical VLSI design flow can be stated as in figure 1.1. As shown in the figure, the design specifications are defined using a hardware description language like VHDL or Verilog. This can be a behavioral level or a block level description of the design. Each design is associated with a technology library containing pre-characterized standard-cells that will be used for the design. The library contains standard information about cells like the timing, area and power specifications. Logic synthesis performs optimization on the high-level design specifications to generate a netlist for the design. Some of the optimization steps involved here are logic minimization, scheduling, binding, structuring, technology mapping, gate sizing, buffer insertion etc. [100]. We perform

timing verification at each step of optimization to ensure that the design meets the required timing constraints. The optimized netlist then undergoes physical synthesis that comprises of floor-planning, placement and routing (global and detailed) [81]. There are several detailed optimizations that are involved in each stage of the VLSI design flow. Essentially, physical design takes a netlist and generates a lower-level representation of it on silicon that is ready to be sent for fabrication. The technology information ensures that the physical design is compatible with the fabrication process. The location of the pins/pads, clock and power supply distribution etc. are characterized according to the technology specifications.

The new challenges in DSM technology has changed this rather *sequential* looking VLSI design flow into a more unified design flow. Logic and physical synthesis are no longer two sequential steps. Logic synthesis required estimation of design parameters that are available only after physical design. Hence, the design flow needs to have an interaction between logic synthesis and physical design to enable faster design closure. As seen in the figure, we can see that circuit parameters are extracted after physical synthesis and are then processed during the next iteration of logic synthesis. The current design methodologies are making this iterative flow more unified so as to achieve faster design closure. The quick turn-around time demanded by this market does not allow the designer to keep iterating on his design to meet specifications. In presence of fabrication and environmental variability, efficient design closure becomes an even bigger challenge. There is a need for more accurate models and estimation techniques to bridge the gap between logic and physical synthesis. An even bigger challenge is to consider the fabrication and

environmental variations into the VLSI design flow to enable quicker design closure and a higher fabrication yield.

The traditional optimization and analysis techniques often lead to sub-optimal or invalid (violates design constraints) solutions in the presence of fabrication variabilities. These variabilities make it extremely difficult to accurately estimate parameters even at lower levels of design flow. This fabrication and environmental variabilities cause the performance of the chip to deviate from the specifications leading to a dramatic reduction in the yield after fabrication. Also, it is not possible to consider these variations in the traditional design flow (which would perform several iterations of the design flow to generate a valid solution) due to design time constraints. The time to market window has shrunk significantly and these business aspects are also creeping into the way ICs need to be designed.

As a result of shrinking dimensions, secondary effects are becoming significant. For instance, until recently it was acceptable to perform timing analysis by considering only gate delays and ignoring wire delays. But in the deep sub-micron technology wire-delays are becoming more critical as compared to gate/device delays. It has been predicted [113] that for deep sub-micron technologies, almost 80% of the delay in critical paths will be the interconnect delays. The entire timing optimization/analysis paradigm has shifted to consider interconnect delays in the current technology.

In this work, we have tried to address the variability problem in DSM/Nanoscale VLSI design automation. We will introduce the problem of variability and discuss its sources. We will also discuss how design automation is adapting to these po-

tential problems by adopting a variability-aware design analysis and optimization methodology. We will look at some techniques for design analysis and optimization that explicitly consider the impact of variability on design performance.

We can categorize the sources of variability in DSM VLSI design [111] into three broad categories:

1. Fabrication/Manufacturing Variability
2. Environmental Variability
3. Estimation/Modeling Variability

1.1.1 Fabrication Variability: Sources and Issues

The current DSM technologies have extremely high costs and therefore require a rapid turn-around time to generate revenue to meet the financial constraints. There is tremendous pressure on the designers to create more complex and more powerful designs under small time-to-market windows. *Performance* and *yield* are both limited by the existence of fabrication variability effects in DSM designs. Fabrication process induced parameter variations cause performance fluctuations and have become important considerations in DSM ICs. Until now it was sufficient to consider die-die variations which were handled typically using a worst-case design methodology. In DSM there are significant within-die variations in terms of device and interconnect parameters [111]. Furthermore, these within-die variations are dependent not only on the fabrication process but also on the implementation

(physical design) of the IC. Hence, these sources of variability are caused by both the design-flow as well as the fabrication process.

The fabrication variability can be divided into two categories:

1. Die-Die variations: These are variations which are imposed on the design by the fabrication process. Within the same die, there is no variability in parameters. Such cases can be analyzed using the classic Monte-Carlo or worst-case techniques. The fabrication process can characterize the variability as a distribution for such an analysis. Though the increasing number of sources contributing to die-die variations is making it increasingly hard to accurately counter the impact of these variations on design analysis and performance.

2. Within-Die variations: These variations are both fabrication as well as design dependent. Firstly, there can be a large number of parameters which are varying making it harder to perform any form of analysis. The variations can be both spatially correlated or independent (random variations). We do not have accurate modeling as well as analysis techniques to handle these within-die variations. There variations can be in device parameters (like V_{th}, W, T_{ox} etc.) or in interconnect parameters (like sheet resistance R_s, L_{max} etc.)

Some trends in device and interconnect parameter trends have been given in the SIA technology roadmap [113]. Typical parameter values for each generation of technology node (between 250nm to 70nm) are given in table 3.5. In [111], the corresponding delay variability imposed by these technology parameter (within-die) variations are given in table 3.6. It is interesting to note that both device and

Parameter	1997	1999	2002	2005	2006
L_{eff} (nm)	250	180	130	100	70
T_{ox} (nm)	5	4.5	4	3.5	3
V_{dd} (V)	2.5	1.8	1.5	1.2	0.9
V_T (V)	0.5	0.45	0.4	0.35	0.3
W (μ)	0.8	0.65	0.5	0.4	0.3
H (μ)	1.2	1.0	0.9	0.8	0.7
ρ (m$\Omega/\square$)	45	50	55	60	75

Table 1.1: Technology Parameters

Parameter	1997	1999	2002	2005	2006
V_{dd}	9.5	10.8	10.0	1.2	0.9
	Device				
L_{eff}	32.4	28.3	25.5	24.6	23.8
T_{ox}	1.3	2.5	3.2	3.9	4.9
V_T	3.8	5.3	5.5	6.5	7.2
	Interconnect				
W	13.3	12.0	11.7	11.4	10.5
H	7.8	8.0	8.1	8.3	7.1
ρ	16.0	16.6	17.9	18.4	20.1

Table 1.2: Percentage Delay Variability Imposed by Within-Die Variations

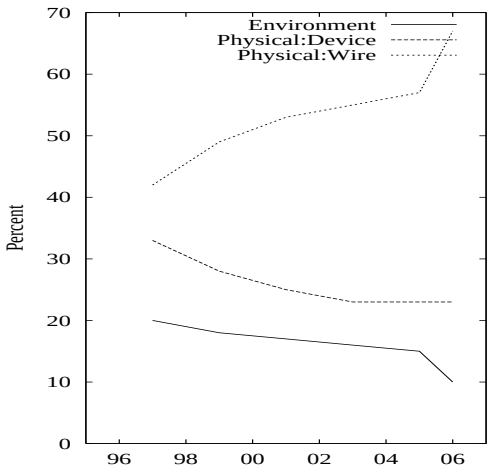


Figure 1.2: Technology Parameter Variations

interconnect variability are causing significant variations in delay.

The authors in [112] performed experiments on the above device parameters to study the variations induced by fabrication variability on the design. In figure 1.2, they present the technology parameter variation trends across five generations of technology node. We note that though the device variations (T_{ox}, V_T) somewhat stabilize with newer technology nodes, the interconnect variations keep increasing. This clearly highlights the importance of considering interconnect variations as well as device variations in any form of VLSI design optimization. The current deterministic models for estimating variability are not able to predict these variabilities with reasonable accuracy. In [25], the authors point out that:

- Technology scaling is continuously reducing physical dimensions and the effect of variabilities is such geometries is making the current estimating models very inaccurate.

- It was assumed earlier that the variations in devices was strongly correlated

thereby reducing the number of sources of variability. However, DSM technologies show large number of sources of variabilities which are correlated as well as independent. This increasing number of independent sources of variability is making analysis very difficult.

1.1.2 Environmental Variability: Sources and Issues

There is variability in design due to various environmental uncertainties. These include variations in power supply voltage V_{dd}, coupling noise, temperature variations, soft errors, electro-migration issues etc. These variations have been characterized as probability distributions and have been analyzed at the intra-die level using worst-case or Monte-Carlo techniques. But as we enter the nanoscale regime, these variations are starting to pose a bigger challenge and cannot be handled through the traditional techniques. As the number of source of these variations increase, using deterministic techniques to estimate such variations is becoming increasingly pessimistic leading to large overheads in performance while trying to counter of these variations. There is a growing need to consider environmental variations probabilistically/statistically during the design flow similar to fabrication variability.

1.1.3 Estimation/Modeling Variability: Sources and Issues

The estimation variability comes into design flow primarily due to design abstraction and the lack of accurate estimation techniques. Early in the design flow, the specifications are only at a very high level of abstraction and therefore interac-

tion with lower-level (physical design) can only be estimated [112]. It is very hard to make accurate predictions at such a high-level without knowing any information for the physical design since there is a certain amount of design uncertainty that is not concretely specified.

Additionally, we need to have accurate models that can be used to predict design parameters with reasonable accuracy. Often such accurate models either do not exist or are computationally expensive to be used in the design flow. Both logic synthesis and physical design suffer from insufficient parameter estimations. This has led to an increasing requirement for logic synthesis and physical design to be more unified so that information can be efficiently exchanged between the two steps. As shown in figure 1.1, the design parameters (like capacitance, parasitics etc.) is extracted after physical design (layout) and fed back to the logic synthesis stage to provide a more accurate estimation of wire delays and capacitances in the next iterations of design cycle. But this can still lead to potentially many iterations of the design cycle which is not acceptable due to tight design-time windows. In [35] the authors have shown that in DSM micron designs, the current interconnect delay models are not accurate when interconnect delay forms a large part of the critical path delay (which is true for DSM designs). Furthermore, they show that the iterative feedback in design flow to better estimate design parameters may not sufficient to allow for incremental optimization capabilities. This form of estimation variability can be countered by using better modeling techniques. There is a need to shift from the traditional deterministic models and adopt a probabilistic modeling framework for better estimation/prediction of device parameters.

These sources of variations have a big impact on design performance. Critical issues like timing and power inaccurately estimated leading to very poor design solutions.

1.2 New Design Methodology Paradigm

In the presence of variability, we need to redefine our design objectives. Correspondingly, we need to focus on different design methodology paradigms. We want to create designs that are robust and predictable. Robustness adds an inherent immunity towards the existing variabilities in the systems as is a desirable characteristic in any design. Additionally, we could also focus towards getting a higher parametric yield from our design methodology. Let us now understand how these two different paradigms come into the picture in DSM designs in presence of variability.

1.2.1 Predictable/Robust Designs

Robustness is an extremely desirable property from any design because it inherently implies stability towards variations. Predictability in design automation has been defined as a quantified form of accuracy/certainty [14]. Each step of VLSI design flow requires estimates to drive the optimization. If these estimates are more accurate, we end up with a more predictable design after optimization. The idea is to get an accurate estimate of the cost function being optimized, so that the final design is more robust. The goal of a predictability-driven objective function is to

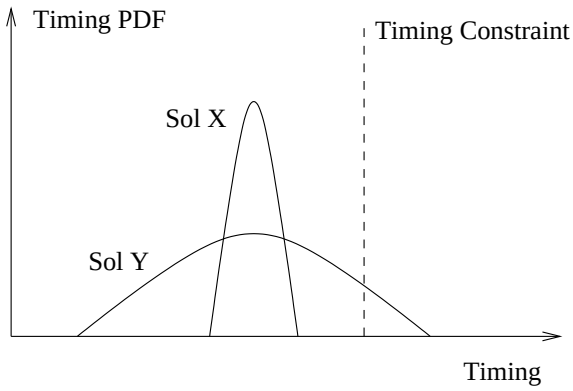


Figure 1.3: Predictable/Robust Solution

choose the most accurate solution.

A predictable estimate of an objective function does not vary much in presence of uncertainty and provides robustness in design. [18] defines a design to be robust if its performance is not influenced by factors like coupling noise, temperature or other variations. This essentially means that the design is more tolerant towards perturbations while still performing within acceptable limits. The authors point out that in order to achieve such a design methodology, the effects and interactions of these factors with design performance need to be investigated. The modeling of the optimization problem should identify and include these variations within the design framework. However, a robust/predictable solution hence obtained may not be the optimal solution and is also not guaranteed to meet the design constraints. As shown in figure 1.3, let us suppose that we generate two solutions X and Y. We can impose the variability on both these solutions and generate their timing PDFs. As shown in the figure, we can see that solution X is more predictable/robust as compared to solution Y because it has lesser uncertainty in value.

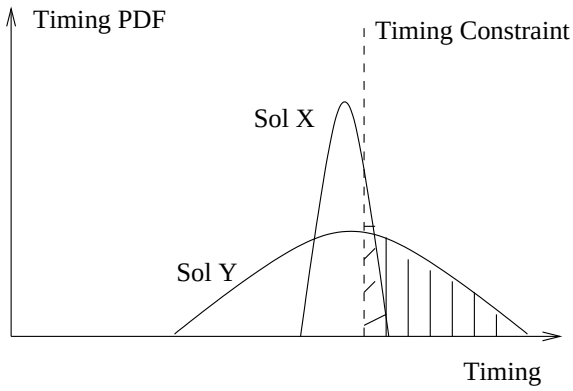


Figure 1.4: High Parametric Yield Solution

1.2.2 Parametric Yield Optimization

Yield from fabrication of ICs is a very important aspect of large scale production. This directly determines the cost of production and hence, we need to ensure that a high yield is achieved. Typically, the yield is calculated by binning the ICs according to their design performance (operating frequency, total power consumption etc.). ICs that do not meet the requirements are rejected. There can be catastrophic defects that lead to ICs that do not work at all. Defects caused due to dirt particles or photo defects are categorized under catastrophic (or non-recoverable) defects. Parametric yield is basically caused by fabrication variations and other disturbances in the environment. Essentially this results in sub-optimal performance of the ICs in terms of timing and can also be referred to as timing yield.

Improvement of timing yield in presence of process variations has caught a lot of attention lately [106, 89, 51, 11]. The objective in parametric yield maximization is to obtain a solution that is likely to meet the constraints with the highest prob-

ability after fabrication. As shown in figure 1.4, let us suppose that we generate two solutions X and Y. We can impose the variability on both these solutions and generate their timing PDFs. As shown in the figure, we can see that solution X is a better solution from a parametric yield perspective as compared to solution Y because it has a higher probability of meeting the timing constraints.

1.3 Current Approaches to Variability Driven Design

There are a few existing design paradigms that try to capture variability during design automation. The goal is to obtain fast design closure satisfying all constraints while ensuring that the IC meets the performance requirements after fabrication. There are both deterministic and probabilistic approaches in this context.

1.3.1 Deterministic Techniques

These techniques perform optimization based on a fixed (deterministic) estimate of the parameters. As soon as variability is introduced into the framework, it is hard for such technique to accurately capture the nature of the variability. There are worst-case approaches, sensitivity-based approaches and slack management based deterministic approaches that try to capture variability in a deterministic perspective.

Worst-case Analysis/Monte-Carlo

This is the traditional form of analysis. However, in presence of fabrication variations this approach tends to be very pessimistic. This is a corner-based approach that tries to identify the design corners in the solution space to ensure that the performance of the design is acceptable within the extreme boundaries. In a variability perspective, firstly there are a lot more design corners making this approach very inefficient. Additionally, there are design corners that exist with a virtually zero probability for all practical purposed and this approach tries to optimize the design around such points as well. As a result, the final design is very sub-optimal. Since all variations are not perfectly correlated, the worst-case scenario does not imply a worst-case occurrence of all parameters [78]. There are local independent random variations that also need to be considered. It is very pessimistic to assume that the worst-case occurs when absolutely every parameter is at its worst-case value.

Another possible approach is to generate upper and lower bounds on the solution. It is generally a hard task to obtain tight bounds on the potential solution. In [2], the authors proposed statistical bounds in a timing analysis framework. Intra-die variations have been analyzed deterministically using Monte-Carlo based simulations with reasonable efficiency. Certain within-die interconnect variations have also been modeled using worst-case deterministic approaches [120].

Sensitivity-Based Analysis

Sensitivity based optimization/analysis is generally performed to get a more robust design. The idea is to try to get a design that is less sensitive to variations in parameters. It is in general hard to identify the relevant sources of variability at each step in the optimization, so as to optimize the objective function to be less sensitive to these parameters. We note that the goal here is not to obtain an optimal performance from the design, but to obtain a robust performance from the design. In [110], the authors have shown that a stable/robust solution is not necessarily close to being an optimal one. Although this technique is very well suited towards a predictable/robust design methodology paradigm.

Constraint Relaxation Techniques / Slack Management

These techniques rely on inducing flexibility in the design constraints by relaxing some of them during the design flow. At higher-levels of design flow, this flexibility allows the later optimization steps to handle variability and other issues better. In [16], the authors introduce the concept of delay relaxation parameter as a property to reach design closure. The paper talks about scheduling in high-level synthesis. The timing constraints of functional resources are relaxed without violating the data-flow constraints. Future optimizations in logic and physical synthesis tend to benefit from this extra flexibility and increased the chances of design closure in presence of variability.

Slack in different paths of a circuit can be efficiently distributed to gain maxi-

mum benefits from optimization. In [24], the authors perform gate sizing to consider delay uncertainty. In [124], the authors have tried to implement the slack management paradigm in the probabilistic framework.

1.3.2 Probabilistic Analysis/Optimization

Probabilistic techniques have gained a lot of attention recently. Variability in DSM technologies have let to the failure of existing deterministic optimization paradigms that are not able to effectively capture the variations. A lot of work has been done in the area of timing analysis under variability from a probabilistic perspective [42, 105, 79, 2, 39, 10, 118, 27, 119, 1, 96, 43].

Probabilistic technique represent each parameter variation as a distribution and tries to maximize the distribution of the objective function during optimization. Accurate models representing parameter dependencies are required for such an approach. In [9] and [68], the authors try to model wire-length variability as a distribution in the post-placement pre-routing stage of VLSI design.

From a robustness perspective we try to find a solution that has minimum variance in its distribution in presence of variability. From a parametric yield perspective, we try to find a solution that has the least chances of violating the design constraints. Essentially, the designer is taking a probabilistic risk of not satisfying the constraints (in presence of variability). Voltage scheduling through such a risk-management paradigm for higher parametric yields has been proposed in [6]. In [115], the authors present a technique to perform buffer insertion to maximize the

parametric yield under wire-length variability.

1.4 Variability-Aware Design Methodology

1.4.1 The Basic Idea

As mentioned in the previous sections, variability-aware approaches can be both deterministic as well as probabilistic. There is an inherent limitation in the deterministic approach to capture the variations effectively. Recently, there has been a shift in focus towards probabilistic design methodologies that are better able to capture fabrication variabilities. It is very hard to model the design parameters as fixed quantities. The variability in their values makes it easier for us to capture their nature by modeling them as random variables. This allows us to generate a distribution (PDF/CDF) for each parameter (modeled as a random variable). The objective function can also be represented as a distribution. From a robustness/predictability perspective, we are looking to minimize the variance in the final solution but from a timing yield perspective, we are trying to maximize the chances of meeting the design constraints.

A probabilistic optimization framework could be represented as shown in figure 1.5. The central block of the framework is a probabilistic optimizer that takes in the design constraints (user constraints), the probabilistic optimization objective and the probabilistic risk that the designer is willing to take of violating the design constraints. Additionally, we require accurate models to capture the variability to represent it as distributions and accurate ways to capture the correlation informa-

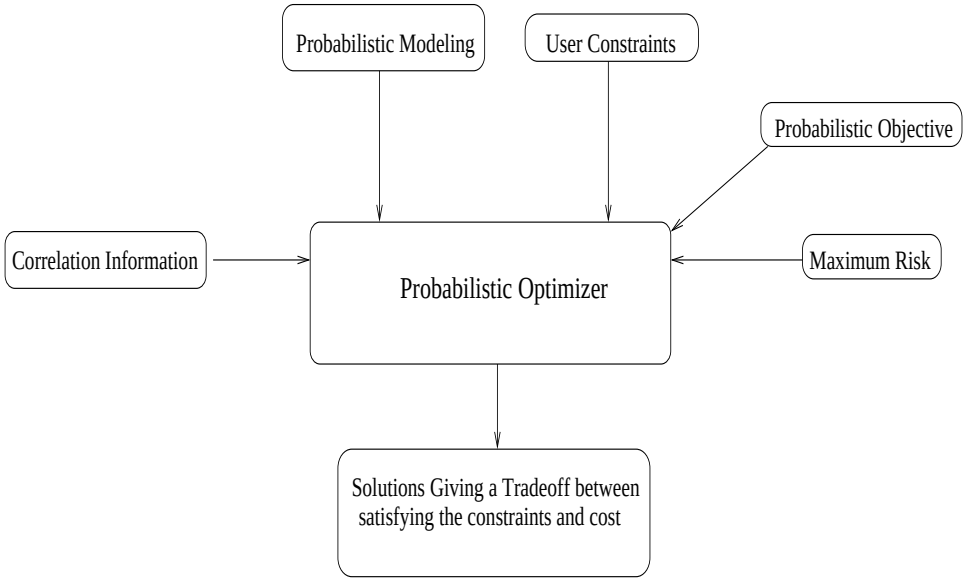


Figure 1.5: Probabilistic Optimization Framework

tion between these distributions. The final solutions from the optimizer are those that satisfy the constraints within the user defined risk limits. The one which has minimum cost (from a robustness or parametric yield perspective) is chosen as the final solution.

1.4.2 Key Advantages

There are several advantages of using such a probabilistic optimization framework in presence of variability.

- Handling fabrication variability in design flow: Since we can probabilistically model all design parameters, we can capture the variability in the design effectively. Each parameter variation can be estimated as a distribution and given to the probabilistic optimizer. This statistical information is assumed to be given to the optimizer.

- Faster design closure: Since we can consider variability issues concurrently while performing design optimization, we are able to reach a design satisfying all constraint faster. We do not need to iteratively try to refine the design to cover all process-corners in the solutions space.

- Risk management: The probabilistic framework allows the designer to decide the amount of risk he is willing to take in his design. A higher acceptable risk typically results in solutions with lower costs as a trade-off. The quality of solution versus the cost of the solution is presented an an interesting trade-off to the designer.

1.4.3 Key Challenges

Though there are several advantages in using a probabilistic optimization framework, there are several key challenges to developing such a methodology:

- Probabilistic modeling: It is important to be able to generate accurate probabilistic models of all design parameters. This requires the processing of statistical data on the parameter combined with its deterministic information. In general this poses a tough challenge to the designers. Also, the cost of evaluating/using such a model should not be very high (computational complexity) in order for it to be a practical solution.

- Correlation Modeling: It has been shown that correlations are very significant in parameter variations. From a timing perspective, most global variations induced due to fabrication variability are spatially correlated. We need to

be able to accurately capture these correlations during optimization. Additionally, not all source of variations are correlated, so we must also have the flexibility to capture independent correlations. Since all gates on a chip are manufactured through the same process, there are global correlations between them. Additionally, there can be variations in characteristic based on their spatial locations, i.e. gates that are placed in close physical proximity are more likely to see similar variations. It is hard to model these correlations such that they can be computed accurately and efficiently.

- Run-time complexity: Probabilistic optimization technique is general require a lot more computation than their deterministic counterparts. The reason for this is obvious since they try to process more information in each step (everything is a distribution as compared to a fixed value in deterministic framework). As compared to deterministic technique, probabilistic technique have significantly higher run-time complexity. We need fast technique to compute probabilistic data without sacrificing the accuracy of capturing correlations. On the flip-side, it is perhaps better to run one iteration of a slow probabilistic framework to get a good solution as compared to many runs of the deterministic algorithm (which occurs when variabilities get higher).

1.5 Techniques for Handling Randomness due to Variability

Variability due to fabrication and environmental randomness poses severe performance, yield and reliability issues in nanoscale designs. There are several philoso-

phies that can be applied to counter the impact of these randomness.

1.5.1 Reduce the Sources of Variations

One of the most promising ways to counter fabrication variability are to develop newer fabrication techniques where we have more control over the actually fabrication process. Such advances will allow us to reduce the manifestation of variations thereby avoiding the problem of randomness due to variability. A key requirement of this is to develop fabrication techniques that are firstly compatible with the existing fabrication flow as well as have a low cost overhead. Without these two qualities, it is extremely difficult for mainstream ASIC designs to adopt these newer fabrication techniques that can reduce the sources of variations.

1.5.2 Design-Time Optimization

In this approach, one can try to explicitly model and consider the impact of variability during the design flow. We appropriately modify our design analysis and optimization techniques to ensure that we can use both deterministic and probabilistic algorithms to consider the performance spread that occurs due to variations. Such an approach allows us to gauge and limit the performance band of the design to ensure that we get the desired yield and reliability without paying any extra overhead in design cost and design time. Key issues for such an approach are:

1. Accurate and compact modeling of the variability data

2. Analysis techniques that can use the modeling information to predict the im-

pact of variability on design performance through variability-aware analysis. In recent years, statistical timing analysis [42, 105, 79, 2, 39, 10, 118, 27, 119, 1, 96, 43] has emerged as one such effective analysis scheme that uses variability-aware modeling for timing analysis considering the impact of fabrication variability.

3. Design optimization techniques that are driven by the variability-aware analysis to probabilistically/statistically optimize the design performance considering variability. A lot of work has been done considering state of the art design optimization techniques like buffer insertion [115, 7, 69], gate sizing [117, 8, 73, 67, 80, 32] and leakage optimization [97, 89, 77, 73, 83].

1.5.3 Post-Silicon Design Tunability

This approach presents a powerful solution to the fabrication and environmental variability problem. Using this design philosophy, we can build in tuning knobs into the design which can be used to selectively alter design parameters once the chip has been manufactured. After fabrication, through external or on-chip testing, we can gauge the real manifestation of randomness due to fabrication variability and then appropriately tune the chip to counter the impact of variability and improve design yield and performance significantly. Tunability in designs can be provided through adaptive body-biasing, supply voltage scaling and through post-silicon tunable clock-tree buffers [117, 72, 114, 82, 36].

1.5.4 Runtime Adaptivity

Runtime adaptivity based techniques are effective in being able to counter the impact of runtime environmental variations. The design is such that it has the capability of sense the occurance of variations and adapts the design performance to counter the variation while still trying to meet the overall performance constraints on the design. Self-correcting designs and architectural modifications can be made by using techniques like reconfigurable logic and redundant path based designs to counter the impact of variability during runtime. Furthermore, another interesting possibility is to use larger blocks of asynchronous logic which is inherently immune to variations.

1.6 Organization

This dissertation is organized as follows:

Chapter 2 discusses our research contributions to the area of variability-aware analysis, specifically looking at the problem of statistical timing analysis. We present the basic background behind statistical timing analysis, discuss the current literature and present our research contributions in the area. We specifically present two works, one of which presents a framework for non-linear, non-Gaussian statistical timing analysis considering correlations [119]. The second work presents an interesting technique to control the trade-off between runtime and error induced during statistical timing analysis. This technique allows us to get significant run-

time speedups compared to traditional statistical timing analysis using the concept of *error budgeting*.

Chapter 3 discusses our research contributions to the area of variability-aware design optimization. We first present the existing literature in this area and then discuss two specific works in this area. Firstly, we talk about a probabilistic buffer insertion technique [115]. This work was one of the first to consider buffer insertion in presence of wirelength uncertainty and proposed technique to perform probabilistic design optimization using buffer insertion. Secondly, we present our work on a general optimization framework based on Monte-Carlo technique considering fabrication variability. In this work, we look at stochastic programming based technique in a linear programming framework. Such techniques can be applied to several VLSI-CAD problems without making any assumptions of the nature of the distributions of the variability data or the correlations between them.

Chapter 4 presents our work on design optimization through post-silicon tunability. The work titled variability-driven simultaneous gate sizing and post-silicon tunability allocation [117] presents a design management framework that can be used to balance the effort spent on pre-silicon (through gate sizing) and post-silicon optimization (through tunable clock-tree buffers) while maximing the yield gains.

Chapter 5 presents our work on runtime dynamic power optimization technique considering variability. We talk about probabilistic design synthesis and runtime optimization as a means to get reliable and robust designs. This work titled Simultaneous Resource Binding and Dual-Vdd Allocation for Power Optimization with Probabilistic Reliability Guarantee, introduces the concept of probabilistic re-

liability guarantee as a metric of optimization considering the impact of fabrication and environmental uncertainty at system-level design stage. We propose a framework that presents a design technique for runtime optimization of dynamic power through supply voltage scaling.

Chapter 6 concludes this dissertation and presents some interesting directions for future work in variability-aware design methodology for nanoscale technologies. Further, we present some directions to extend these ideas to distributed integrated and embedded systems as well focusing on the key challenges of performance, reliability and yield.

Chapter 2

Variability-Aware Timing Analysis

Static Timing Analysis is one of the most critical steps in VLSI design. It is used both during optimization as well as verification. In high performance design, it is absolutely imperative for the design to meet the required timing constraints. This brings out the need for fast, accurate and incremental timing analysis methodologies.

In the past static timing techniques [84, 88] have provided a reliable and efficient method for timing analysis during design sign-off or verification. These techniques were enhanced to accommodate for the DSM effects like coupling noise, RC and RLC interconnect modeling, simultaneous switching and other variations. The die-die and within-die variations were also handled typically by performing case analysis. As pointed out in [25], this paradigm is breaking down due to the increasing DSM effects. It is very hard for conventional static timing analysis to account for the variability accurately. These schemes are deterministic in nature and it is very hard to capture the nature of the distributions of the variability using such techniques. Performing a case-based or corner-based analysis required a large number of static timing runs as the number of independent sources of variability are increasing. Furthermore, at the design corner a worst-cast assumption is made which is pessimistic, while it is very hard to analyze all possible design corners. Missing one such critical design corner could lead to failure which are detected after the IC is

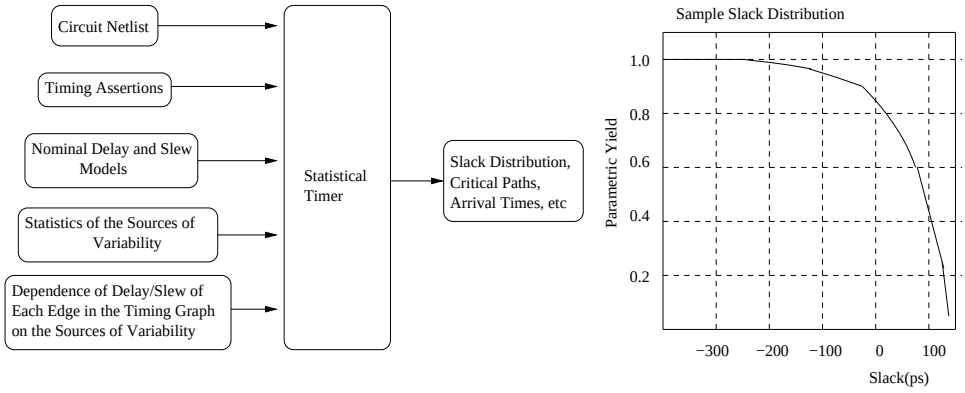


Figure 2.1: Statistical timer: block diagram & sample slack distribution

manufactured.

This brings the need to develop *Statistical Timing Analysis* (STA) techniques, that will allow the designer to aim for high-performance while giving a quantitative risk-management against the effects of fabrication variability. Let us now understand the basics about a typical statistical timing framework.

2.1 Statistical Timing Analysis

A conventional static timer takes the circuit as an input and builds a timing graph from it. Delay models are used to provide the timing information (delay, slew etc.) about each gate. The analysis computes the timing slack in the circuit from which the highest frequency of operation can be determined. Additionally, it can produce a list of failed timing tests, arrival times at gates, slack at gates, critical paths or any other timing information that may be useful.

As shown in figure 2.1, we can see that a statistical timer takes in additional information about the nature of the sources of variations. We can generated accurate

modeling of the fabrication variability to calculate the distribution of these variations and the correlation information. The statistical timer has the capability to link these sources of variability to their effect on timing values. The main output from the timer is a probability distribution of the slack. As shown in figure 2.1, we can see that the parametric yield of the circuit is almost 100 % at a slack of -300ps while it sharply drops down as the slack increases. Additional information like arrival times at each gate, slack at each gate, slew, critical paths etc. are also reported by the timer which can be used for later optimization.

A statistical timer needs to be able to model the correlations that exist between different parameter variabilities. In general most within-die variations are correlated although there is some independent randomness that exists in the die. Correlations can be path-based (reconvergent fanout) which essentially occur because two paths can share a sub-path between them. Correlation also exist between gates that share the same voltage islands (V_{dd} fluctuations). There is global correlation due to die-die variations (fabrication variability) as well as spatial correlations for within-die variations. Additional effects that thermal heating etc. also induce correlations into the gate behavior. And lastly, there is also some uncorrelated independent variations (doping concentration, T_{ox} etc.) that exist on the chip as well. A good statistical timer needs to be able to model these variations along with the correlation information.

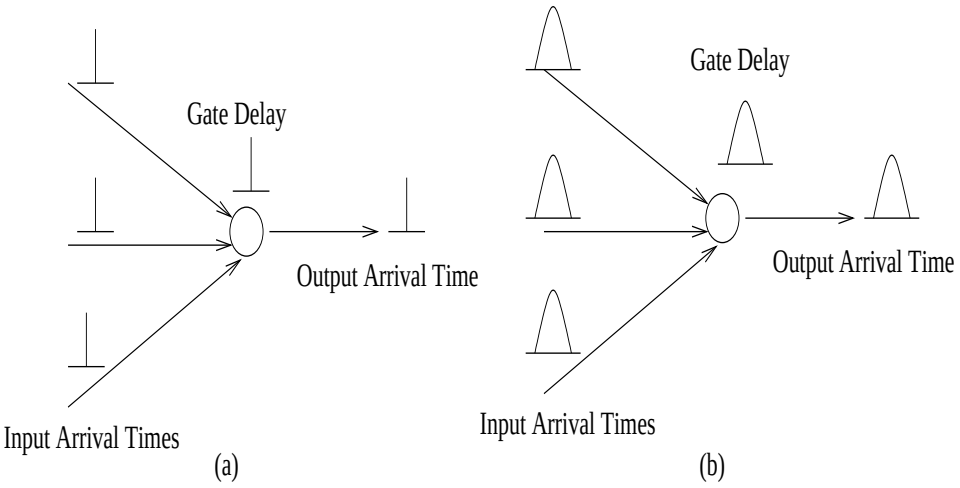


Figure 2.2: Static versus Statistical Timing Analysis

2.2 Current Approaches in STA

Let us now try to understand the working of a statistical timing analysis technique in more detail. Essentially, the idea is to try to capture the variability in design parameters by modeling them as distributions (PDFs/CDFs) or even as random variables (with a corresponding distribution). This essentially implies that in the STA modeling framework, each arrival time as well as gate delay becomes a distribution (represented as a PDF/CDF or a function of random variables). As shown in figure 2.2(a), a typical static timing framework represents each arrival time as well as gate delay by a deterministic value which can be easily propagated through the circuit generating the required timing information. In STA, as shown in figure 2.2(b), each arrival time/gate delay is modeled as a distribution and we now need to propagate the distribution through the circuit to generate the required timing information. Additionally, as mentioned earlier, the variations in the parameters are correlated globally as well as spatially. There can also be independent random

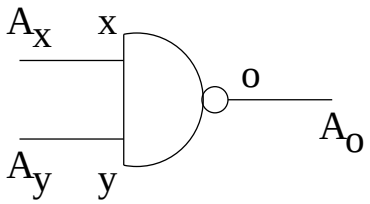


Figure 2.3: Gate with x and y input pins and output o

variations due to fabrication variability as well. The STA framework needs to be able to model as well as propagate these correlations accurately.

Timing Analysis involves the computation of two main operations on the timing variables. As shown in figure 2.3, let us assume that we are given a gate g with two input pins x and y and an output pin o. The arrival time A_o at the output pin o can be calculated given the arrival times at the input pins (A_x, A_y) and the gate delay (G_D). Mathematically, this operation can be represented as given by equation 2.1:

$$A_o = MAX(SUM(A_x + G_D), SUM(A_y + G_D)) \tag{2.1}$$

Hence, we first need to perform a SUM operation and then perform a MAX operation on the timing variables to calculate the distribution of the arrival time variables. While the SUM operation is computationally tractable, calculating the exact arrival time after the MAX operation is very difficult. The main problem here is that it is very hard to accurately compute the MAX of two timing PDFs/CDFs or functions of random variables efficiently. This has been a primary bottleneck in STA techniques and has prompted most researchers to model the distributions as Gaussian and enforce a Gaussian assumption on the timing distribution after the

MAX computation in order to exploit the various mathematical results that exist on Gaussian Random Variables. We will discuss this issue in greater detail later in this chapter.

2.2.1 Modeling Arrival-Time and Gate Delays

There are two basic ways in which the timing variables can be modeled in STA:

- In [10, 118], the authors have modeled the arrival times and gate delay distributions as PDFs and CDFs. The key problem with both these approaches is the assumption of independence in the timing variables (no correlation information being captured), which is the central assumption in the computational efficiency proposed by these schemes. In [2], the authors present a way to compute bounds on the exact PDF of the timing values in the circuit considering within-die parameter variations.

- Most of the recent work on STA [42, 105, 79, 27, 119, 1, 96, 43, 60] models arrival time and gate delays as a function of random variables. Each parameter that has variability is modeled as an independent random variable. The parameters that are globally or spatially correlated are shared between all gates in the circuit while the independent randomness at each gate is modeled as a separate independent random variable. Initially, this modeling was done such that the gate delay/arrival times were modeled as a linear function of random variables, but lately due to the increasing non-linear effects of variability on

timing, the recent works tries to extend this to a non-linear gate delay model framework [119, 43]. The main reason for modeling timing variables as random variables is to provide an efficient way of capturing correlations between different gates (by using the same random variables for each gate to represent global parameters). Also, there is a vast amount of mathematical framework that exists in random variable theory. In particular Gaussian Random variables are well-studied and have results that are extremely useful in computing the MAX operations. This is the primary reason why most of the existing work in STA modeled timing information as Gaussian Random Variables.

2.2.2 Block-Based STA Versus Path-Based STA

STA can be performed in two primary ways on any circuit:

- Path-Based: The basic idea here is to generate a list of critical paths and perform timing analysis only on these paths. The problem is this approach is that there could be exponentially many paths that need to be analyzed. It is very hard to decide how many critical paths should be timed. However, since a path-based computation involves only the SUM operation on each path, it is accurate in computing the distributions of the arrival times on each path. At the primary outputs, we just need to perform one MAX operation on all the critical path timing information to generate arrival time distribution at the primary output node. So if there is a path p with gates $a, b, ...k$ with gate delays $D_a, D_b, ...D_k$, the arrival time at the end of the path (which would be

a primary output gate) would be $A_p = D_a + D_b + ... + D_k$. The final arrival time at the primary output would be the MAX of all such paths (say x in number) ending at that gate as be given as:

$$Arrival - Time = MAX(A_1, A_2, ..., A_x) \tag{2.2}$$

There is a lot of research that has been proposed on this framework [42, 105, 79, 2, 39]. This approach is able to capture the global and spatial correlations efficiently and is also able to account for reconvergent fanout based correlations. However, a major drawback is that this approach is not able to provide the accurate timing distribution information at each gate (which can be used for further analysis and optimization) and is good only for final timing sign-off for the ICs.

- Block-Based: This technique does not consider path-based delay computation as described above. Here, we explicitly compute the arrival time distribution (as a PDF/CDF or a function of random variables) at each gate. The circuit is traversed topologically from the primary inputs to the primary outputs to generate the timing information at each gate. This provides an approach for incremental timing analysis and generated detailed timing information at each gate for further analysis/optimization. At each gate we perform SUM and MAX operations as given by equation 2.1. In [10, 118], the authors use PDFs/CDFs to propagate the timing information. Other work in block-based STAs have used parametric models to represent each arrival time and gate

delay as a function of random variables [27, 119, 1, 96, 43, 60]. Though the block-based schemes are computationally efficient, they are prone to approximations while computing the MAX operations on arrival time variables at each gate. Furthermore, using a parametric representation of timing value makes it easier to capture global/spatial correlation accurately but capturing the independent randomness at each gate (and propagating it through the circuit) is still a problem. The block-based approaches provide incremental timing capabilities which are useful during optimization.

.

2.3 Key Challenges in STA and Our Research Contributions

There are several challenges that exist in developing an accurate and efficient STA framework. There are two inherent problems in accurate STA, namely correlation modeling and accurate computation of the MAX operation. While the parametric modeling has enabled global/spatial correlation to be captured accurately, it is still hard to accurately propagate the independent randomness component of variability. It is trivial to model this independent randomness into the gate delay models by using a random variable at each gate to represent its random variation. But propagating this information accurately during STA is hard because for a design with 100,000 gate, we will have 100,000 independent random variables. Current STAs schemes [27] tries to make approximation on this by just maintaining one independent random variation term in each arrival time model, but this results in

loss of variation information. Hence, we need to develop a better way to model the independent randomness such that it is computationally efficient as well as accurate.

The MAX operation still poses the biggest challenge in STA. It is very hard to generate an accurate representation of the result of MAX operation on the arrival time distributions (expressed as a function of random variables). Since the random variables in general can have any distribution, the nature of the result of the MAX operation can be any distribution. However, in order to make this MAX operation feasible, researchers have chosen to model the arrival times as Gaussian random variables. There exists a lot of mathematical framework to facilitate this computation. In [42, 27, 60, 96], the authors have resorted to modeling each random variable as a Gaussian variation (which is not true in general) and have also considered a linear gate delay model that represents each gate delay as a linear sum of these Gaussian variations. This implies that the gate delays (and arrival times) too are Gaussian. Using analytical results proposed by [22], they have proposed a scheme that is able to approximate the result of the MAX operation back into the linear form. This approximation of linearity as well as Gaussian nature adds errors into the timing estimates generated from STA.

Firstly making an assumption that all parameter variations are Gaussian in inaccurate (for example the variations in the Via resistance is know to be non-Gaussian). Secondly, the gate delay and arrival time models can no longer be considered linear with increasing fabrication variations in DSM technologies (transistor channel length L_{eff} variations are known to be non-linear in their delay dependence). More recently, there have been attempts to generalize the STA framework

to consider non-Gaussian non-linear parametric variations [43, 119], but there are still lots of questions that need to be answered in the field of Statistical Timing Analysis.

In the rest of the chapter, we will present our contributions to the area of statistical timing analysis. We have tried to address some of the issues and challenges posed by variability in timing analysis. The first work presents a novel error budgeting formulations that tries to reduce the runtime complexity of statistical timing analysis. The second work presents an approach towards a general statistical timing analysis framework that does not make assumptions about the nature of the variabilities and the dependence of gate delays on these parameters.

In section 2.4, we propose a novel technique for optimizing the runtime in statistical timing analysis. Given a global acceptable error budget at the primary output which signifies the difference in the area of the accurate and approximate timing CDFs, we propose a novel formulation of budgeting this global error across all nodes in the circuit. This node error budget is used to simplify the computation of arrival time CDFs at each node using approximations. This simplification reduces the runtime of statistical timing analysis. We investigated two ways of exploiting this node error budget, firstly through piecewise linear approximation ([10]) and secondly though hierarchical quadratic approximation. Experimental results on ISCAS/MCNC benchmarks show that our approach is at most 3 times faster than accurate statistical timing analysis and had a very small error. We also found quadratic piecewise approximation to be more accurate than linear approximation but at lesser gains in runtime.

In section 2.5, we present a general Statistical Timing Analysis (STA) framework that captures spatial correlations between gate delays. Our technique does not make any assumption about the distribution of the parameter variations, gate delays and arrival times. We propose a Taylor-series expansion based quadratic representation of gate delays and arrival times which are able to effectively capture the non-linear dependencies that arise due to increasing parameter variations. In order to reduce the computational complexity introduced due to quadratic modeling during STA, we also propose an efficient linear-modeling driven quadratic STA scheme. We ran two sets of experiments assuming the global parameters to have uniform and Gaussian distributions respectively. On an average, the quadratic STA scheme had 20.5x speedup in runtime as compared to Monte-Carlo simulations with an rms error of 0.00135 units between the two timing CDFs. The linear-modeling driven quadratic STA scheme had 51.5x speedup in runtime as compared to Monte-Carlo simulations with an rms error of 0.0015 units between the two CDFs. Our proposed technique is generic and can be applied to arbitrary variations in the underlying parameters under any spatial correlation model.

2.4 Efficient Statistical Timing Analysis Through Error Budgeting

Growing importance of fabrication variability and estimation uncertainty has lead to increased significance of statistical timing analysis. Several researchers have investigated this issue in detail [3, 2, 1, 42, 59, 83, 96, 27, 10]. Statistical timing analysis problem essentially takes a DAG G = (V,E) as input with each node delay and arrival time represented as a distribution. It calculates the distribution of the arrival time at the primary outputs (POs) of the DAG. One of the most important issue in statistical timing analysis is the runtime. The latest work by Devgan et. al [10] proposes an approach for fast statistical timing analysis in which after the node arrival time CDF is evaluated, the CDF is approximated by a piecewise linear approach. This simplification results in massive gains in runtime.

Our work builds upon this approach for statistical timing analysis. The key problem in the approach presented in [10] is that whenever a signal is approximated by piecewise linearization, this linearization is performed using an arbitrary and predecided number of lines. Having too few lines could result in large amount of error and too many lines could result in large execution runtime. Hence an adaptive way of determining the degree of approximation for each signal is needed which can effectively perform a tradeoff between gain/loss in runtime with increase/decrease in error. In order to achieve this tradeoff we investigate the way error gets propagated in statistical timing analysis. We propose a closed form expression for this error propagation. Using this expression, we propose the philosophy of error budgeting. The error budgets at each node are used to approximate the node delay PDFs

and arrival time CDFs. We investigate two kinds of approximation strategies: linear (traditional) and hierarchical quadratic. This entire statistical timing analysis framework is put together in the SIS framework. Experimental results show that our budgeting approach comes very close to accurate statistical timing estimation (without any approximation) but can be at most 3 times faster. Comparatively, the traditional approach [10] had a large error in the output arrival time CDF. We also found the quadratic approximation to be much more accurate than linear approximation but with lesser gains in runtime.

2.4.1 Motivation and STA Framework

In this work, we propose a novel approach for speeding up statistical timing analysis by effectively controlling the amount of error injected for gains in runtime. Given the distribution of the arrival time at the primary inputs and the distribution of the gate delays, the problem is to evaluate the distribution of arrival time at the intermediate nodes as well as the output nodes in the circuit. Similar to static timing analysis, statistical timing analysis traverses the circuit topologically from the primary inputs to the primary outputs generating the arrival time distribution at the out of each intermediate node.

The SUM and the MAX operation in the statistical timing framework need to be computed on the distributions of arrival times and gate delays. In [10], the authors propose to model the arrival times as cumulative density functions (CDFs) and the gate delays as probability density functions (PDFs) as shown in figure 2.4(a).

t_1 and t_2 denote the range of the distributions in both the cases as shown in the figure.

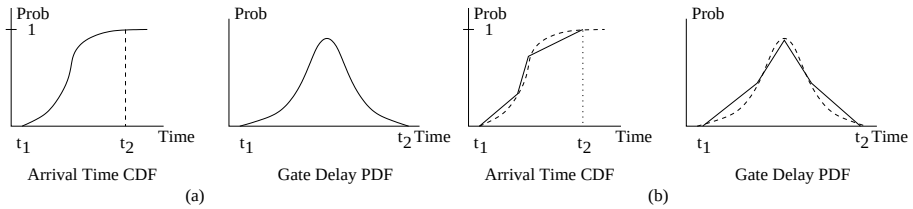


Figure 2.4: Distributions and their Linear Approximations

For computational efficiency of the SUM and MAX operations of statistical timing, these CDFs and PDFs are approximated using techniques of piecewise linear and quadratic approximations. The details of these modelings are given later in section 2.4.3. In figure 2.4(b), the CDF and PDF are shown under the piecewise linear approximation scheme. We will now discuss the SUM and MAX operation under these CDFs and PDFs. We assume that the arrival times and gate delays are independent of each other. The issue of statistical dependence due to re-convergent fanouts needs to be resolved [10], [1], [3]. In [10], the authors propose an efficient heuristic technique based on common mode removal approach which we have implemented in this work.

In [10], the authors show that the CDF of the arrival time $C_o^x(t)$ at the output due to input pin x is given by the convolution of the input arrival time CDF $C_x(t)$ with the PDF of the pin-to-pin gate delay $P_o^x(t)$ as given by equation 2.3. This follows from the fact that the probability distribution of the sum of two independent random variables is the convolution of their probability distributions.

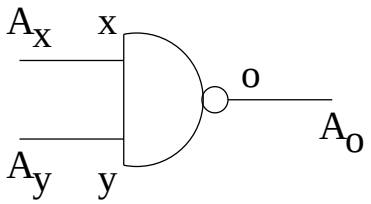


Figure 2.5: Gate with x and y input pins and output o

$$C_o^x(t) = \int_0^t (C_x(t-\tau) * P_o^x(\tau) d\tau) \tag{2.3}$$

Similarly, the CDF $C_o(t)$ after the MAX operation on the arrival time CDFs $C_o^x(t)$ and $C_o^y(t)$ at the output pin o (refer to figure 2.18) can be computed from equation 2.5. The CDF of the maximum of two independent random variables is the product of their CDFs.

$$A_o(t) = MAX(A_o^x(t), A_o^y(t)) \tag{2.4}$$

$$C_o(t) = C_o^x(t) * C_o^y(t) \tag{2.5}$$

Hence statistical timing operations SUM and MAX are now performed by doing a convolutions followed by a multiplication. Hence the arrival time distribution at the output of the gate, given the input arrival time distributions and the gate delay distribution can be given by equation 2.6.

$$C_o(t) = (C_x(t) \otimes P_o^x(t)) * (C_y(t) \otimes P_o^y(t)) \tag{2.6}$$

Now that we have the formulations for the MAX and SUM operation for statistical timing using CDFs and PDFs, we can run statistical timing analysis

similar to conventional static timing. Equation 2.6 can used to evaluate the output CDFs at each gate in the circuit. In order to speed up statistical timing evaluation, the approach of [10] linearizes the arrival time CDF into a prespecified number of lines. It also approximates the arbitrary node delay PDF into stepwise function. The authors then formulated a closed form expression for evaluating equations 2.3 and 2.5 when the arrival time CDFs were represented using a piecewise linear approximation and the node delay PDF was represented using a stepwise approximation. This results in huge speed ups in runtime when compared with a traditional point-wise convolution based approach. The overall runtime of timing analysis depends upon the total number of lines used to represent the arrival time CDF and the total number of steps used to represent the node delay PDF.

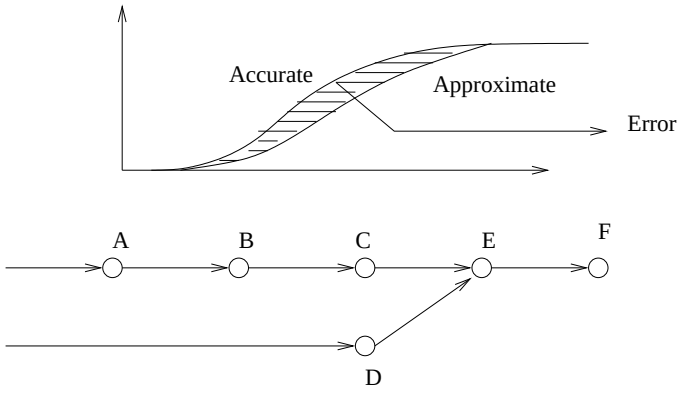


Figure 2.6: Error Budgeting

In this work we propose novel ways of controlling this tradeoff between the overall error and runtime. Specifically, we have investigated two issues in this direction.

1. Given an error budget that the user specifies, identify the degree of approx-

imation needed for each individual node arrival time CDFs and node delay PDFs

2. Investigating better approximation strategies like quadratic (instead of linear) for improving error and same runtime

Figure 2.6 illustrates the basic philosophy behind our approach. Given, node delay distributions in a DAG, the approach in [10] topologically computes the arrival time CDFs at each node. Whenever a new CDF is computed it is simplified by representing it as a piecewise linear approximation. This simplification adds an error into the statistical timing estimation which is controllable by the number of lines used to approximate the CDFs. Finally, the CDF at the output has some error when compared with the accurate arrival time CDF. In this work, we define this error as follows

$$ERROR = \int_{t_{min}}^{t_{max}} |C_{accurate} - C_{estimate}| dt \tag{2.7}$$

Essentially, this is the total area in the entire range of interest where the actual signal is different from the approximate signal. Let us suppose that we are provided a total error budget E that the user is willing to tolerate at the primary output. Given this error budget, we would like to assign it to all nodes in such a way that maximum gains in runtime occur. Traditionally, this global error budget is essentially spread uniformly. This is not a very effective strategy of distributing the global error since the DAG may have unbalanced paths. Consider the example DAG shown in figure 2.6. Approximating all node CDFs with the same number of

points would not be the best idea since node D is not critical. Hence the global arrival time CDF at node F has low sensitivity to the amount of error in the arrival time CDF at node D. Hence runtime speed-ups could be achieved by adding more error at D by approximating it in lesser number of lines. We call this concept *Error Budgeting*, since through this approach we strive to control the amount of error in the final output CDF for gains in runtime. We also investigate better approximation techniques like quadratic approximation for lesser error. The budgeting and approximation schemes are integrated into one statistical timing system.

2.4.2 Error Budgeting

In this section we will delve into the details of our budgeting formulation that distributes the global error budget at the PO to each node which can then be utilized for speeding up statistical timing analysis. The error budget at the primary outputs is defined in equation 2.7. In order to distribute this global error budget we need to investigate the way error in arrival time CDFs and node delay PDFs interact when subjected to SUM and MAX operations.

Error in SUM Operation

Figure 2.7 illustrates a situation in which the SUM operation is performed on two signals, one of which is represented as a CDF and other as a PDF (just like equation 2.3). The figure illustrates two representations for the input CDF and PDFs, one of which is accurate and one of which is an approximation. In this section

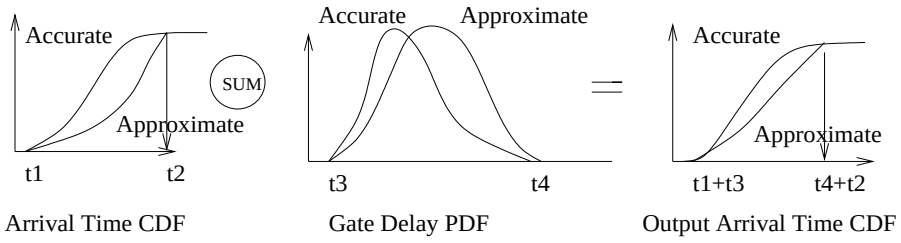


Figure 2.7: Error in SUM

we will discuss the error in the output CDF after the SUM operation as a function of the errors in the input CDF and PDF. The accurate output CDF is given by

$$C_{out}^{accurate}(t) = \int_0^t C_{in}^{accurate}(t-\tau) P_{node}^{accurate}(\tau) d\tau \tag{2.8}$$

The approximate output CDF is given by

$$C_{out}^{aprrox}(t) = \int_0^t C_{in}^{approx}(t-\tau) P_{node}^{approx}(\tau) d\tau \tag{2.9}$$

Since the SUM operation is essentially a convolution operation, the range of the output CDF is defined as follows. If the input CDF starts at t1 and ends at t2 (after t2 the CDF=1) and the input PDF starts at t3 and ends at t4, then the output CDF starts at t1+t3 and ends at t2+t4. Note that here the assumption is that both accurate and approximate curves start and end at the same delay value. As it would be clear in the next section, the way piecewise linear approximation or quadratic approximation is performed, this range does not change. Hence t1,t2,t3,t4 are the same for original and approximate curve. The error in the output CDFs is given by

$$Err_{out} = \int_{t_{min}}^{t_{max}} |C_{out}^{accurate} - C_{out}^{approx}| dt \tag{2.10}$$

The error term can be re-written as follows

$$Err_{out} = \int_{t_{min}}^{t_{max}} | \int_0^t C_{in}^{approx}(t-\tau) P_{node}^{approx}(\tau) d\tau - \int_0^t C_{in}^{accurate}(t-\tau) P_{node}^{accurate}(\tau) d\tau | dt \tag{2.11}$$

The range of the integral t_{min}, t_{max} is simply t1+t3 and t2+t4 respectively.

$$Err_{out} = \int_{t_{min}}^{t_{max}} | \int_0^t (C_{in}^{approx}(t-\tau) P_{node}^{approx}(\tau) - C_{in}^{accurate}(t-\tau) P_{node}^{accurate}(\tau)) d\tau | dt \tag{2.12}$$

Let us suppose that $C^{approx} = C^{accurate} + \delta C$ and $P^{approx} = P^{accurate} + \delta P$. Plugging this relation in equation 2.12 gives us the following result.

$$\int_{t_{min}}^{t_{max}} | \int_0^t (C_{in}^{accurate}(t-\tau)\delta P(\tau) + P^{accurate}(\tau)\delta C(t-\tau) + \delta P(\tau)\delta C(t-\tau)) d\tau | dt \tag{2.13}$$

Ignoring the second order term $\delta P(\tau)\delta C(t-\tau)$ and using the relation $|a+b| \leq |a| + |b|$, the above equation could be rewritten as

$$\int_{t_{min}}^{t_{max}} (\int_0^t |C_{in}^{accurate}(t - \tau)\delta P(\tau)| d\tau + \int_0^t |P^{accurate}(\tau)\delta C(t - \tau)| d\tau) dt \tag{2.14}$$

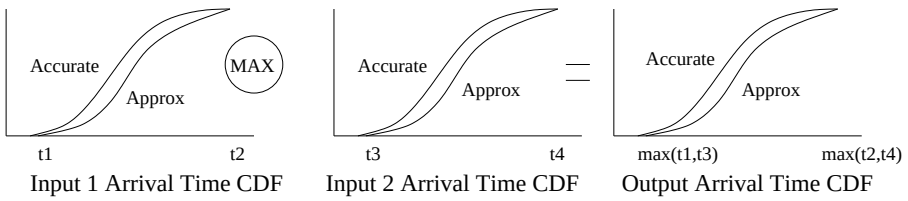


Figure 2.8: Error in MAX

$$\int_{t_{min}}^{t_{max}} (\int_0^t |C_{in}^{accurate}(t\ -\ \tau)\delta P(\tau)|d\tau\ +\ \int_0^t |P^{accurate}(\tau)||\delta C(t\ -\ \tau)|d\tau)dt \quad (2.15)$$

Let the error in the input CDF be $E1 = \int_{t1}^{t2} |(C_{node}^{accurate} - C_{node}^{approx})|dt$ and error in input PDF = $E2 = \int_{t3}^{t4} |(P^{accurate} - P^{approx})|dt$. Since $0 \leq C_{in}^{accurate}(t) \leq 1$ and $E1 \geq |\delta C(t-\tau)|$ it can clearly be seen that equation 2.15 is always $\leq$ the following

$$Err_{out} \quad \leq \quad \int_{t_{min}}^{t_{max}} (\int_0^t |\delta P(\tau)|d\tau \quad + \quad E1 \int_0^t |P^{accurate}(\tau)|d\tau)dt \quad (2.16)$$

$$Err_{out} \leq \int_{t_{min}}^{t_{max}} (E1 + E2)dt = (E1 + E2)(t_{max} - t_{min}) \quad (2.17)$$

Equation 2.17 gives an upper bound on the output CDF error based in the input errors. The range t_{max}, t_{min} is simply the range on which the output arrival time signal is defined. Therefore the output error is a linear combination of input errors.

Error in MAX Operation

Figure 2.8 illustrates a similar situation for the MAX operation. The input CDFs have the range (t1,t2) and (t3,t4) both for the accurate and approximate

cases. The output CDF which is a multiplication of the input CDFs has the range (t_{min}, t_{max}) = (max(t1,t3), max(t2,t4)). Once again the error in the output CDF is given as follows

$$Err_{out} = \int_{t_{min}}^{t_{max}} |(C_{out}^{approx}(t) - C_{out}^{accurate}(t)|dt \tag{2.18}$$

Let $C_{in1}^{approx}, C_{in2}^{approx}$ and $C_{in1}^{accurate}, C_{in2}^{accurate}$ denote the accurate and approximate CDFs for the input signals. Let $C_{in1}^{approx} = C_{in1}^{accurate} + \delta C_{in1}$ and $C_{in2}^{approx} = C_{in2}^{accurate} + \delta C_{in2}$. Using these relations and simplifying, we can write equation 2.18 as follows

$$Err_{out} = \int_{t_{min}}^{t_{max}} |C_{in1}^{accurate}\delta C_{in2} + C_{in2}^{accurate}\delta C_{in1}|dt \tag{2.19}$$

Let E1 = $\int_{t1}^{t2} |C_{in1}^{approx} - C_{in1}^{accurate}|dt$ and E2 defined similarly for the second input. Using $|a+b| \leq |a| + |b|$, it can be shown that following must hold.

$$Err_{out} \leq \int_{t_{min}}^{t_{max}} |C_{in1}^{accurate}\delta C_{in2}|dt + \int_{t_{min}}^{t_{max}} |C_{in2}^{accurate}\delta C_{in1}|dt \tag{2.20}$$

$$Err_{out} \leq E1 + E2 \tag{2.21}$$

Although equation 2.21 is an upper bound on the error, this bound is not good enough since it does not capture the criticality of the inputs. As discussed in the previous section, the error in a non-critical fanin would not affect the output error too much. Unfortunately, equation 2.21 does not capture this philosophy. Hence we refine this bound by making some approximations on the input CDFs.

Figure 2.9 illustrates three possible overlaps between the two input CDFs. In

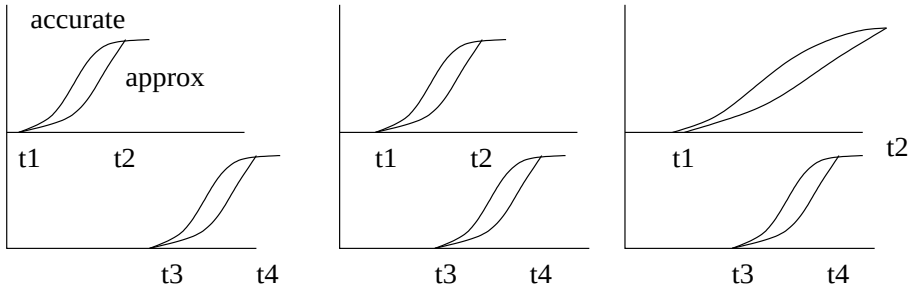


Figure 2.9: Error Bound in MAX

the first case, the CDFs have no overlap whatsoever. Here $t1 \leq t2 \leq t3 \leq t4$. In such a situation,$C_{out}$ will be zero until t_{min} = max(t1,t3) = t3 and will become 1 at t_{max} = max(t2,t4). Essentially the second signal is always more critical than the first one. Also, since, we have assumed the range of approximate and accurate curves to be the same, the error is zero outside it. If we focus on equation 3.63, the second term must be zero since over the range t_{max}, t_{min} = (t4,t3), the error in the first signal is zero. Hence the entire output error is contributed by E2. Analytically, this means that since signal-2 is critical, the error contributed by signal-1 does not affect the output signal.

In the second case in figure 2.9, the two input signals overlap such that $t1 \leq t3 \leq t2 \leq t4$. Here t_{min}, t_{max} = (max(t1,t3)=t3, max(t2,t4)=t4). In such a case, we assume that the two CDFs are lines with the following slopes

$$S1 = 1/(t2 - t1) \tag{2.22}$$

$$S2 = 1/(t4 - t3) \tag{2.23}$$

This approximation is needed in order to evaluate a closed form expression for

the output error in terms of the input errors. Hence the input CDF $C_{in1}(t)$ = S1(t - t1) $\forall$ t $t1 \leq t \leq t2$ and input CDF $C_{in2}(t)$ = S2(t - t3) $\forall$ t $t3 \leq t \leq t4$. Let us also approximate the error between the accurate and approximate CDFs to be uniformly distributed. This is illustrated in the following equations

$$\begin{aligned} \delta C_{in1}(t) &= E1/(t2-t1) \quad t1 \leq t \leq t2 && (2.24) \\ &= 0 \qquad\qquad otherwise && (2.25) \end{aligned}$$

$$\begin{aligned} \delta C_{in2}(t) &= E2/(t4-t3) \quad t3 \leq t \leq t4 && (2.26) \\ &= 0 \qquad\qquad otherwise && (2.27) \end{aligned}$$

Once again we would like to re-iterate the assumption that the range of accurate and approximate CDFs are the same. This issue will be further explained later. Equation 3.63 has two terms, each corresponding to the error contributed by the respective inputs. For each term, the entire range of integration is split into two parts: from $t_{min} = t3$ to $t2$ and from $t2$ to t_{max}. The first term in equation 3.63, $\int_{t_{min}}^{t_{max}} |C_{in1}^{accurate} \delta C_{in2}| dt$ therefore gets split into two integrals. Using the simplifying assumptions given by equations 2.22 and 2.26, this term can be written as follows

$$\int_{t_{min}}^{t_{max}} |C_{in1}^{accurate} \delta C_{in2}| dt = K2E2 \tag{2.28}$$

Here $K2 = (S1/t4 - t3)((t2^2 - t3^2)/2 - t1(t2 - t3)) + (t4 - t2)/(t4 - t3)$. Similarly the second term in equation 3.63 can be simplified as follows

$$\int_{t_{min}}^{t_{max}} |C_{in2}^{accurate} \delta C_{in1}| dt = K1E1 \tag{2.29}$$

Here $K1 = (S2/(t2-t1))((t2^2 - t3^2)/2 - t3(t2-t3))$. Therefore the total error is given by $K1E1 + K2E2$.

Now let us consider the final case in figure 2.9. In this case one input signal completely engulfs the other. In this case t_{min}, t_{max} is given by (max(t1,t3)= t3, max(t2,t4) = t2). Under similar simplifying assumptions form equations 2.23 and 2.24, we can re-express equation 3.63 as $K1E1 + K2E2$ with

$K1 = (S2/(t2-t1))((t4^2 - t3^2)/2 - t3(t4-t3)) + (t2-t4)/(t2-t1)$

$K2 = (S1/t4-t3)((t4^2 - t3^2)/2 - t1(t4-t3))$

It can be seen that in all cases the error is bounded by $K1E1+K2E2$ where K1 and K2 can be calculated using the proposed expressions. This gives us a compact and effective way of estimating the output error given the input errors and the ranges in which the input CDFs exist. It should be noted that the upper bound property may not hold anymore.

The assumptions made on the nature of the CDFs considering them to be linear ramps as given by equations 2.22 and 2.23 can be relaxed for better accuracy. We could also consider them to be Gaussian (or any other distribution) and evaluate closed form expressions for $K1$ and $K2$ (under the assumption that the error is uniformly distributed).

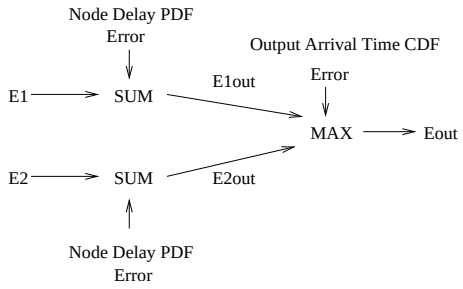


Figure 2.10: Error Injection in a Gate

Having delved into the details of how the error propagates in the SUM and MAX function, now we will describe the way error budgeting is performed for each node.

Error Budgeting for Runtime Optimization

Given a user defined error budget at the primary outputs of a DAG, we would like to assign error budgets to individual nodes in the DAG such that overall error budget constraint is satisfied and maximum gains in runtime could be achieved. Given the input DAG, let us add a sink node and add directed edges from all POs to this sink node. We also assume this sink node has zero delay.

Figure 2.10 illustrates the way error is injected into a gate. There are two inputs with errors E1 and E2. First these input signals are SUMmed with the corresponding input pin to output delay. At this point there is an error injected that corresponds to the error corresponding to linear approximation or quadratic approximation of the node delay PDFs as shown in figure 2.10. These CDFs are then MAXed together to get the node output arrival time CDF. Another error is added here which corresponds to the linear approximation or quadratic approximation

of the output CDF as shown in figure 2.10. Hence there are two kinds of errors associated with a gate: first is the one that gets injected due to simplification of the node delay PDFs and other due to simplification of the node output CDF. Hence in the entire DAG, each node has two variables corresponding to node delay PDF simplification (assuming all gates are 2 inputs) and one variable for node output CDF simplification. Therefore there are 3n error variables, where n is the number of nodes. Errors need to be assigned to these variables such that the overall sum of the errors for all variables is maximized and the error budget at the sink node is satisfied. Formally this can be written as follows.

$$Maximize \sum_{\forall nodes:i} (e^{pdf}_{input-j:i} + e^{pdf}_{input-k:i} + e^{cdf}_{i:out}) \tag{2.30}$$

$$e_{sink:out} \leq ERR - BUDGET \tag{2.31}$$

$$e^{dummy}_{input-j:i} = K1^{sum}_{ij} e_{j:out} + K2^{sum}_{ij} e^{pdf}_{input-j:i} \ \forall inputs - j : i \ \forall i \tag{2.32}$$

$$e^{dummy}_{out:i} = K1^{max}_{i} e^{dummy}_{input-j:i} + K2^{max}_{i} e^{dummy}_{input-k:i} \ \forall i \tag{2.33}$$

$$e_{out:i} = e^{dummy}_{out:i} + e^{cdf}_{out:i} \ \forall i \tag{2.34}$$

Equation 2.32 illustrates that when the input CDF is SUMmed with the node PDF, then the output error is a linear combination of the input error and the error injected by approximating the node delay PDF. The values of the linear constants

could be calculated as described in the previous subsections. Equation 2.33 illustrates that the output CDF error given the error injected by the MAX operation on two input signals. This error is a linear combination of these two input errors. The output CDF at node i is also approximated thereby introducing another error into the formulation as shown in equation 2.34. There is a global error budget at the sink node. The objective is to maximize the total error budget since this would be directly proportional to the overall runtime improvements. There is still the issue of assigning the constants in the above equations (essentially K1, K2 etc.). The last subsection derived analytical formulae for these constants that were dependent on the range of existence of each of the signals. These ranges can be easily derived for all signals in the DAG as follows. First replace all node delays by their minimum possible values and perform as static timing analysis (this gives the lower limit on arrival time). Then replace all node delays by their maximum possible values and perform static timing analysis (this gives the upper limit on arrival time). The range of arrival times for all signals essentially gives the range which is needed by the analytical formulae to compute the K1 and K2 terms. This completes the description of the budgeting formulation.

2.4.3 Linear and Quadratic Approximation Schemes

Our error budgeting scheme discussed in section 2.4.2 allocates an error to each approximation step. If we can ensure that the approximation error introduced at each step is within the error budget, we can control the total error in the CDF

of the output arrival time.

Piecewise Linear Approximation

We can approximate the PDF of gate delay and the arrival time CDF into piecewise linear PDF and CDF respectively [10]. We are given an error budget for each approximation step from the error budgeting technique explained in section 2.4.2. The piecewise linearization could be iteratively refined until the overall error is less than the budget. The piecewise linear CDF and PDF can then be decomposed into a sum of ramps as shown in figure 2.11(a) and (b) respectively. Hence, if an approximation step has a large error budget, we can approximate it with very few lines are get considerable runtime savings.

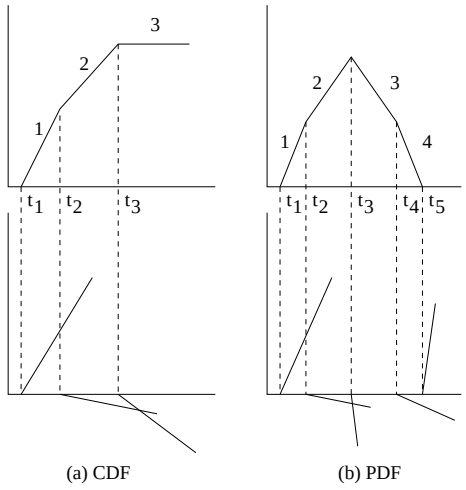


Figure 2.11: Decomposing CDF and PDF into sum of ramps

The SUM operation as defined before would now be applied to the piece-wise linear CDF (with n ramps) and piece-wise linear PDF (with m ramps) and result in mn convolutions. We can then add up these convolution results to get the

intermediate CDF after the SUM operation. However, we will retain them in this decomposed form for the MAX operation. The convolution between two ramps with slopes s_1 and s_2, starting at t_1 and t_2 can be calculated as shown in figure 2.12(a). The convolution result C_o has a closed form expression given by

$$C_o = s_1 s_2(1/6t^3 - (t_1/2 + t_2/2)t^2 + (1/2{t_1}^2 + 1/2{t_2}^2 + t_1 t_2)t - (1/6{t_1}^3 + 1/6{t_2}^3 + 1/2t_1{t_2}^2 + 1/2t_2{t_1}^2)) \quad (2.35)$$

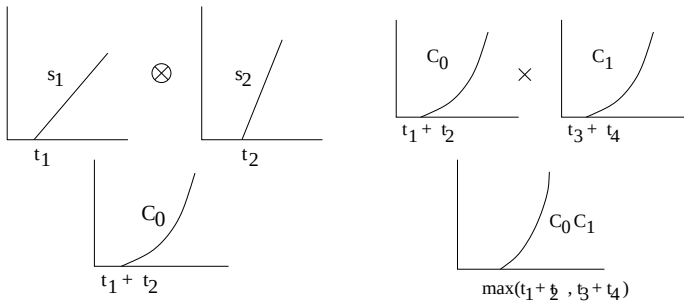


Figure 2.12: SUM and MAX

Hence, after the SUM operation we have each intermediate CDF represented as sum of mn cubic polynomials. The CDF of the arrival time at the output of the gate is given by the MAX operation on the CDFs obtained after the SUM operation on different input pins of the gate. The closed form expression for the resulting CDF is just the product of the CDFs from the SUM operation. Unlike the approach presented in [10], we do not linearize the CDFs after the sum operation because this step would inject unnecessary error into the CDFs. We can compute the MAX operation on the two CDFs C_0 (say with $m_0 n_0$ cubic polynomials after

SUM) and C_1 (say with $m_1 n_1$ cubic polynomials after SUM) as shown in figure 2.12(b) by taking the product of every pair of cubic polynomials that were obtained for both the CDFs after the SUM operation as given by equation 2.35. The MAX operation would therefore generate $(m_0 n_0 m_1 n_1)$ polynomials of degree six which can be summed together to get the CDF of the arrival time at the output of the gate as given by equation 2.36.

$$C_{out}(t) = \Sigma_{i,j} C_0^i(t) C_1^j(t) \tag{2.36}$$

In order to propagate this to the next fanout gate, we again perform piecewise linearization of the CDF. The number of lines that this CDF is decomposed into depends on the error budget allocated to output linearization of this gate. We again repeat the iterative decomposition until the error budget is met.

Hierarchical Quadratic Approximation

The approximation of the CDFs and PDFs can also be done using hierarchical quadratic modeling [44, 45]. This has an advantage over linear approximation since quadratic approximation has lesser error. In this work, we apply the philosophy of hierarchical quadratic modeling in which the approximation is refined hierarchically till the approximation error is within the allocated error budget. We construct a minimal equidistant hierarchical grid structure as shown in figure 2.13(a) for each hierarchy level i. In this work we limit the maximum number of hierarchical levels to four. Each hierarchical level doubles the number of approximation quadratic

polynomials used from the previous level. Between these approximation points the input signal is approximated as a quadratic such that the error in approximation is minimum. If the overall error is more than the assigned budget then another level approximating points is added.

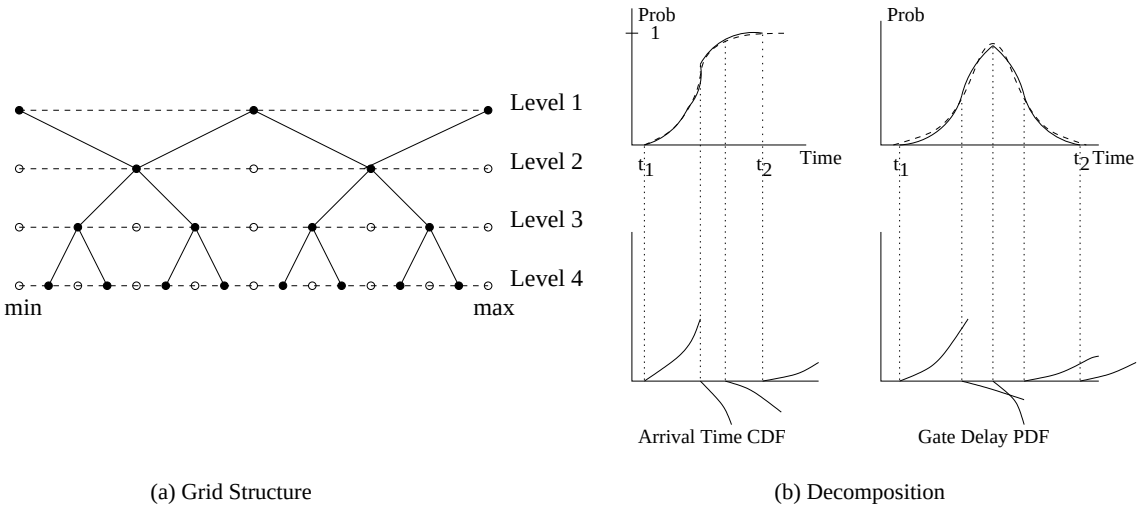


Figure 2.13: Grid Structure and Quadratic Decomposition

Details about the quadratic approximation techniques on hierarchical basis can be found in [44, 45]. Given a distribution as a CDF or a PDF, we can decompose it into piecewise quadratic function analogous to the piecewise linear case as shown in figure 2.13(b). The SUM operation would now be applied to piecewise quadratic CDF (with say n quadratics) and piecewise quadratic PDF (with say m quadratics) and result in mn polynomials of degree five. Similar to the linear case, we can derive a closed form expression for this convolution the details of which are omitted for brevity. We can compute the MAX operation on the two CDFs C_0 (say with ${}_0n_0$ CDFs after SUM) and C_1 (say with m_1n_1 CDFs after SUM) by taking the product of every pair of degree five polynomials that were obtained for both the CDFs after

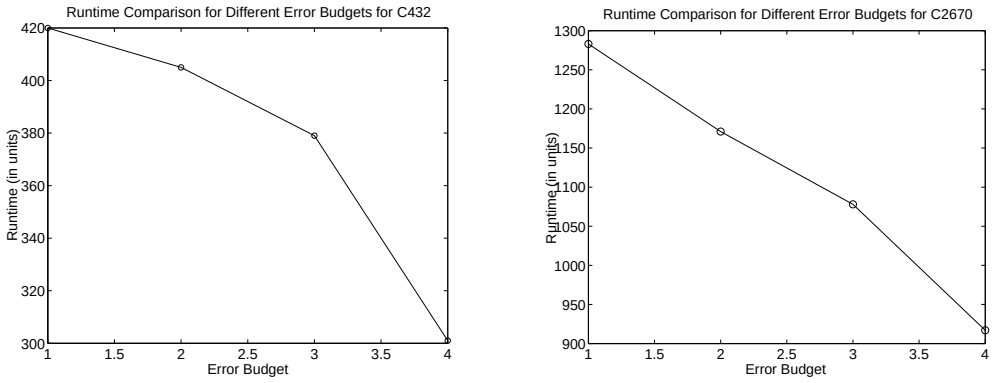
Benchmark	Accurate	Fixed 3 line	Error	Linear Budgeting	Error	Quadratic Budgeting	Error
C432	758	92	30.35	420	16.46	601	3.40
C499	1407	176	37.71	679	27.09	1056	9.45
C880	1160	151	13.87	487	8.75	863	1.11
C1908	1793	218	40.15	889	30.97	1249	2.31
C2670	2850	423	10.77	1283	4.36	1966	0.69
C3540	4071	500	11.02	1918	6.49	3146	1.171
C6288	11935	1930	13.47	5985	5.67	7473	2.71
C7552	9249	1467	5.65	3201	1.34	6562	0.48

Table 2.1: Runtime and Error Comparison

the SUM operation as given by equation 2.35. The MAX operation would therefore generate $(m_0 n_0 m_1 n_1)$ polynomials of degree ten which can be summed together to get the CDF of the arrival time at the output of the gate as given by equation 2.36. Although degree ten polynomials may sound too complicated, these are just close form expressions and could be implemented very easily. The output CDF needs to be approximated once again into a piecewise quadratic simplification. This approximation could be done depending on the error budget allocation for this gate.

2.4.4 Experimental Results

The statistical timing analysis framework with the proposed error budgeting paradigm was implemented in SIS [37]. A topological traversal over the circuit is done in the first step to generate the error budgeting constraints using the LP formulations discussed in section 2.4.2. We use CPLEX to solve the error budgeting problem and get an error budget for each step of approximation in statistical timing analysis. We have used the ISCAS/MCNC benchmarks in SIS for our experiments. The arrival time distributions at the primary inputs were taken to be Gaussian (in

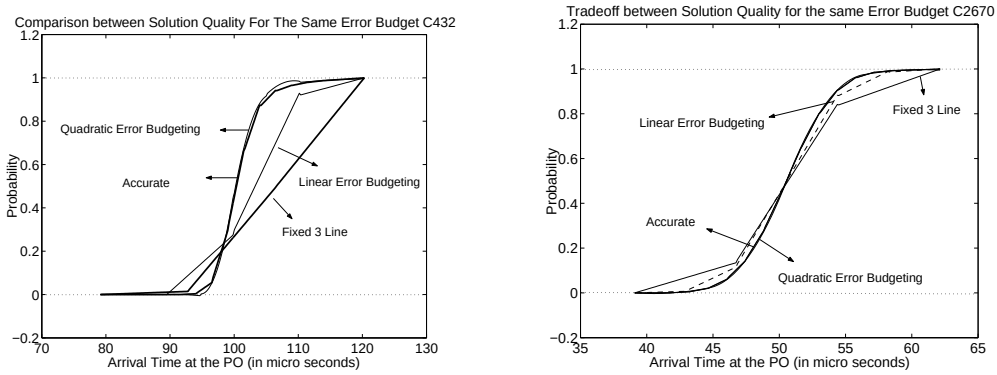


(a) Benchmark C432 (b) Benchmark C2670

Figure 2.14: Runtime Results

CDF form) and the gate delay distributions were taken to be Gaussian as well (in PDF form). We have ignored the global Corellations in this work and reconvergent fanouts were handled similar to [10]. Since our error budgeting approach uses an adaptive scheme to approximate each distribution depending on its corresponding allocated error budget, we limit the maximum number of segments used to make piecewise linear approximation to 16 lines and the minimum segments to be 3 lines ([10] uses fixed 3 line scheme). Piecewise quadratic approximations have maximum 4 hierarchy levels (or 8 quadratic polynomials). We generate an accurate CDF for the output arrival time for each benchmark to make comparisons in runtime, error budget and the quality of solution between our adaptive approach and the 3 line fixed linearization approach proposed in [10].

Table 2.1 shows the runtime and error comparison between fixed 3-line approximation and our adaptive error budgeting scheme (both linear and quadratic).

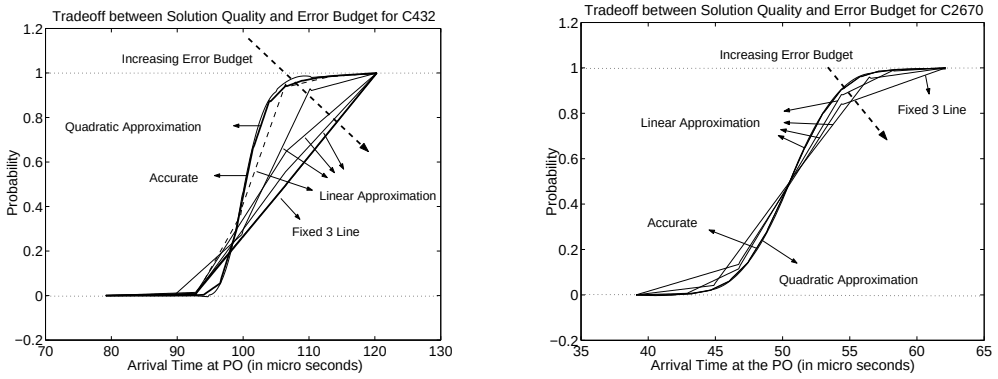


(a) Benchmark C432 (b) Benchmark C2670

Figure 2.15: STA Results

Columns 2, 3, 5 and 7 give the runtimes for the accurate, fixed 3-line, linear and quadratic cases respectively. The comparisons for error are made with respect to the accurate case using equation 2.7. Our adaptive linear approximation scheme using error budgeting give solutions which are bounded by the fixed 3-line linearization scheme from [10] and the accurate solution both in terms of runtime and quality of solution. We also note that the runtime of quadratic approximation is lower than that of the accurate distribution while the solution quality obtained from the quadratic scheme is very close to the accurate one. This shows the efficiency of quadratic approximation. However, when compared with linear approximation, it has a higher runtime but better solution quality as well.

Figures 2.14(b) and 2.14(a) show the tradeoff between the error budget and runtime. Hence, we can exploit this tradeoff to reduce the runtime of statistical timing analysis.



(a) Benchmark C432

(b) Benchmark C2670

Figure 2.16: Error Budgeting Tradeoff

Figures 2.15(a) and 2.15(b) show the CDFs at the primary outputs for two different benchmarks. The error budget assigned to both the linear approximation scheme and the quadratic approximation scheme were the same. We can see from the figures that linear approximation schemes using error budgeting gives better solution quality as compared with fixed 3-line approximation. For the same error budget, quadratic approximation scheme gives us better solution quality but at the cost of a higher runtime when compared with linear approximation scheme. We can clearly see that the solution quality of the quadratic approximation is very close to the accurate distribution for both cases but the runtime are better by 20.7% for C432 and 31.1% for C2670 respectively as shown in table 2.1. These observations clearly bring out the effective of the quadratic scheme over the linear scheme in terms of the solution quality. The proposed concept of error budgeting is effective in saving runtime while preserving the solution quality.

The tradeoff between error budget and solution quality obtained in statistical timing analysis is another key observation from the experiments. Now we try to study the effect of changing the assigned error budget during statistical timing analysis. Figures 2.14(b) and 2.14(a) show the effect of increasing the error budget on runtime. From figures 2.16(a) and 2.16(b), we can see that as the error budget increases, the solution quality from linear approximation decreases. Hence there is a direct tradeoff between the error budget and the corresponding solution quality and runtime.

2.5 A General Framework for Accurate Statistical Timing Analysis Considering Correlations

Statistical Timing Analysis has become a widely researched area with increasing impact of process variations on deep-submicron designs. The growing sources of variations along with the delay correlations they introduce in the design make it increasingly hard to perform fast and accurate timing analysis. Traditional design-corner based static timing analysis has become inaccurate due to pessimistic timing yield estimates. Monte-Carlo based statistical timing approaches become expensive in the presence of such large number of sources of variability. The central idea in STA is to capture the variability by modeling delays as distributions and performing timing analysis statistically on these distributions while capturing possible correlations that could exist between gate delays.

A lot of recent work in statistical timing analysis tries to consider the impact of process variations in performance analysis. Some approaches propose bounds on the statistical timing information [3, 2, 79] which can be computed efficiently for quick statistical timing estimation. Other approaches explicitly compute the timing statistically, making approximations at every step for curtailing the data explosion and improving the runtime. The authors in [26] propose a first order approximate delay model that takes into account both the correlated and independent randomness from different sources of variation. A similar strategy is presented in [42], where the authors present an efficient PERT-like traversal based statistical timing algorithm which considers the effects of the correlations of intra-die parameter vari-

ations by imposing an approximation similar to [26]. A moment based approach for capturing correlations is presented in [60]. In this work, we present a novel STA scheme that considers non-linear (quadratic) gate delay model and does not make any assumption about the nature of the underlying parameter variations. Some other recent works have tried to address the problem of non-linear, non-Gaussian STA ([127, 30, 43]) as well and a preliminary version of this work has been proposed in [119].

In this work we present a novel and general framework for accurate STA. The current approaches represent gate delay as a function of the underlying parameters which are usually taken to be independent principal components. In our approach we model each gate delay and arrival time distribution as a quadratic polynomial using Taylor-series expansion on the independent principal components. We do not make any assumptions about the distribution of the principal components (and consequently the gate delays and arrival times in the circuit). We impose a quadratic polynomial based gate delay model in the scheme. Any arbitrary distribution will work in our general framework. In this work we also present a strategy for computing the MAX of multiple arrival time signals which are also modeled as quadratic polynomials in the principal components. Using regression, we approximate the result of MAX back to a quadratic polynomial with minimum impact on error. Since all timing variables are approximated as quadratic polynomials in the global principal components, the correlations are inherently considered. The computation complexity of STA increases as we move from a linear modeling scheme [26, 42] to a quadratic modeling scheme. In order to address the runtime increase, we also

propose a novel linear-regression driven quadratic modeling STA scheme.

We ran experiments with two sets of underlying parameter distributions. In the first set of experiments, we assumed that the global parameters had a uniform distribution. The results have shown that the proposed linear regression driven quadratic gate delay and arrival time modeling based STA scheme has on an average an rms error of 0.0016 in the output CDF as compared to 0.0453 from linear gate delay and arrival time modeling (which is done by most existing STA schemes) when compared with accurate Monte Carlo CDFs. In the second set of experiments, we assumed that the global parameters had a Gaussian distribution. The results have shown that the proposed quadratic gate delay and arrival time modeling scheme has on an average an rms error of 0.0014 in the output CDF as compared to 0.0319 from linear gate delay and arrival time modeling when compared with accurate Monte Carlo CDFs. This clearly brings out the effectiveness of quadratic modeling of gate delays and arrival times to better capture the variability in timing due to parameter variations. The average runtime speedups for the linear-driven quadratic scheme over Monte-Carlo was $51.5x$, while that from linear scheme was $56.5x$. We also make experimental comparisons with the non-linear non-Gaussian STA scheme proposed in [43].

2.5.1 Modeling Parameter Variations and Spatial Correlations

In this section we will discuss the methodology that we impose for modeling the statistical correlations between the gate delay variables. We assume that the

gate delay is dependent on a number of location-dependent parameters which are assumed to be mutually independent random variables. Let P_i, Q_i and R_i denote three such parameters (although our approach is very general and can be trivially extended to having more sources of variations also). Therefore, the delay of a gate i can be modeled as a function of these independent parameters as given by equation 3.60:

$$D_i = F(P_i, Q_i, R_i) \tag{2.37}$$

We note here that F can be a non-linear function of the parameters. The underlying variables P_i, Q_i, R_i and the corresponding delay can have any distribution. As has been indicated in several other statistical timing techniques, spatial correlation would exist between delay variables of different gates due to spatial proximity.

Spatial Correlation Modeling

Let us suppose that we are given a placed netlist as shown in figure 2.17. We impose a uniform grid on the placement to partition the gates into spatial regions. Let us now consider the parameter P and assume that its variation can be represented as a linear combination of four independent random components namely $P1$, $P2$, $P3$ and $P4$ that are zero mean and finite variance. These four random variables correspond to the four corners of the chip (as illustrated in figure 2.17). For any gate j, we model its corresponding parameter P_j as given by equation 2.38:

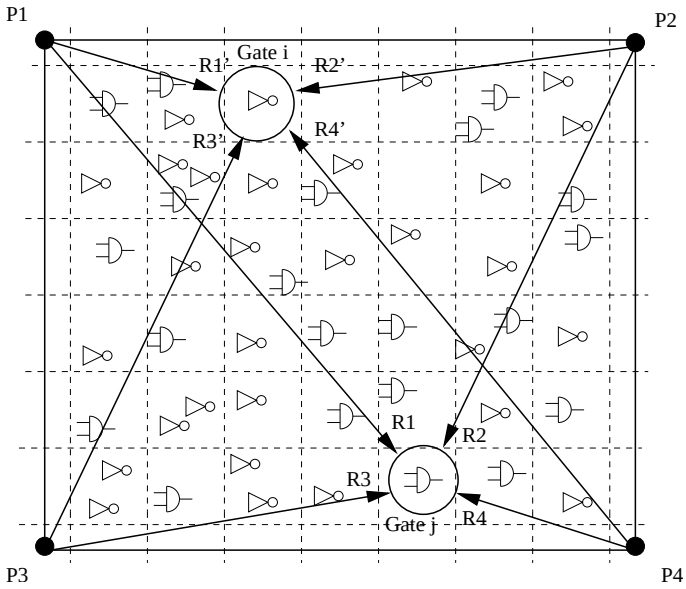


Figure 2.17: Grid-Based Spatial Correlation Model

$$P_j = a_1 P1 + a_2 P2 + a_3 P3 + a_4 P4 + a_0 \tag{2.38}$$

where a_0 is the nominal value of parameter P_j. For any gate j in the netlist, we can compute the grid-based radial distance for the gate from the corners of the placement. This is represented by $R1$, $R2$, $R3$ and $R4$ for gate j as shown in the figure. The coefficients a_1, a_2, a_3 and a_4 are dependent on these radial distances.

The underlying random variables $P1, P2, P3, P4$ can have any arbitrary distribution depending on the distribution of the parameter P_j. Therefore, we can see that if two gates i and j are far apart, they will get different contributions from each of the four components $P1$, $P2$, $P3$ and $P4$ and will have a weak correlation. If they are placed close together, then their coefficients will be similar and strong correlation will exist between them. In this way, we model spatial correlations for each of the remaining parameters in the system (Y and Z in this case).

There are other schemes in current literature [1, 42, 26] that present different spatial correlation models, but there is no strong validation of any existing model. Our scheme has the flexibility to allow any compact and efficient spatial correlation model that may be validated in future research to be used in this framework.

Gate Delay Modeling

In this work we propose a gate delay modeling scheme that represents each gate delay (and arrival time) as a quadratic polynomial in the underlying parameters. Unlike the existing schemes [42, 26], this allows us to consider non-Gaussian and non-linear dependence of delay on the underlying parameters. A similar quadratic modeling scheme has also been proposed in [127]. We have represented our gate delay as a function of the independent parameters as given by equation 3.60. Each of P_i, Q_i and R_i can be represented as a linear combination of their underlying random components as given by equation 2.38. Hence, we can represent our gate delay as a function of these variables as:

$$D_i = G_i(P1, P2, P3, P4, Q1, Q2, Q3, Q4, R1, R2, R3, R4) \tag{2.39}$$

For simplification in representation, let us represent these variables as $Y1$, $Y2$, $Y3$, $Y4$, $Y5$, $Y6$, $Y7$, $Y8$, $Y9$, $Y10$, $Y11$ and $Y12$ respectively. We can use Taylor-series expansion about the mean values on this relation and obtain gate delay D_i as a sum of a series of multiple-order components as given by equation 2.40. The nominal values for the gate delay happens when all Y_i variables are zero (essentially

no variance). Therefore $D_i(nominal) = G_i(0)$.

$$G_i = G(0) + \sum_{k=1}^{12}(Yk)G'(0) + 1/2!(\sum_{k=1}^{12}(Yk))^2G''(0)..... \quad (2.40)$$

The approach in [42] presents a similar strategy in which the delay for each gate is simplified according to Taylor series. Their approach however arbitrarily ignores the higher order polynomial terms and simply represents each gate delay as a a linear combination of the random variables (the Y_i terms in our case). Such a simplification is shown below

$$D_i = c_1Y1 + c_2Y2 + c_3Y3 + c_4Y4 + c_5Y5 + c_6Y6 + c_7Y7 + c_8Y8 + c_9Y9 + c_{10}Y10 + c_{11}Y11 + c_{12}Y12 + G_i(0) \quad (2.41)$$

Typical gate delay models have terms which illustrate a high degree of non linear sensitivity. Such a linear approximation can inject a large amount of error in gate delay modeling (and therefore the statistical timing estimate) itself. In this work we choose not to ignore the higher order terms in the expanded Taylor series. Therefore, we model the gate delays as a quadratic in the global variables Y_i. Note that this quadratic also has cross terms of the form Y_iY_j etc. A general quadratic representing the gate delay would have the following structure

$$D_i = c_1Y1 + c_2Y2 + ... + c_{12}Y12 + c_{13}Y1^2 + ... + c_{24}Y12^2 + \ 66\ degree - 2\ cross - terms\ +\ G_i(0) \quad (2.42)$$

All delay variables in the circuit would share the same global variables Y_i. This would enable effective capturing of the correlations between them.

Additionally, it is known that there is also uncorrelated randomness at each gate in the circuit [26]. We denote this uncorrelated randomness as another random variable R. Every gate would have its own random variable R that is independent of the other variables. In this work we assume that the uncorrelated randomness variable R for each gate has a standard Gaussian distribution. Our gate delay model is able to consider both correlated and uncorrelated components of variations as has been illustrated in this section. Therefore, the complete gate delay model for a gate i can be represented as

$$D_i = c_1 Y1 + c_2 Y2 + ... + c_{12} Y12 + c_{13} Y1^2 + ... + c_{24} Y12^2 + \ 66\ degree - 2\ cross - terms\ +\ G_i(0) + sR_i \quad (2.43)$$

where s is a constant coefficient denoting the sensitivity of the gate delay D_i to the uncorrelated randomness R_i for the gate.

2.5.2 Statistical Timing Analysis Framework

We will now describe our general STA framework. We use a block-based STA approach that traverses the circuit topologically from the primary inputs to the primary outputs. There are two basic operations that are performed at each gate during this traversal. We first perform a SUM operation on the arrival time at a fanin and the corresponding gate delay. This SUM operation is repeated for each

fanin of the gate. We then perform the MAX operation on the result of the already computed SUM operations. This gives us the arrival time at the output of the gate. As described in section 3.1.1, each gate delay is represented as a quadratic in the independent/global parameters. Following a similar strategy we would like to approximate each arrival time signal as a quadratic too. The approach in [26] proposes a similar strategy for representing all arrival time signals as linear combinations of global variables. At the end of the topological traversal of the circuit, the STA data has been generated. Let us now try to understand the two basic operations that are performed repeatedly in STA. Figure 2.18 shows a typical gate in the circuit that has K fanins and a quadratic gate delay representation D. The arrival time at fanin i of the gate is denoted by A_i, which is also a quadratic representation similar to D. We will use the notation $quad(Y1, .., Y12)$ to represent a quadratic in the variables $Y1, Y2, ..., Y12$ in the rest of this work.

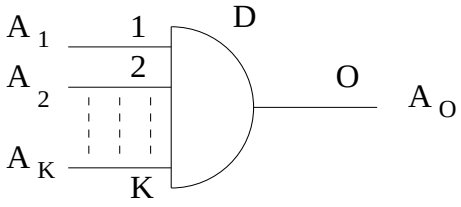


Figure 2.18: SUM and MAX Computation

$$D = quad(Y1, Y2, ..., Y12) + s_0 R0 \tag{2.44}$$

$$A_1 = quad(Y1, Y2, ..., Y12) + s_1 R1 \tag{2.45}$$

... ...

$$A_K = quad(Y1, Y2, ..., Y12) + s_K RK \tag{2.46}$$

SUM Operation

Since arrival time and gate delay are both quadratic polynomials in the same independent parameters, the result of the SUM operation is also a quadratic polynomial. The coefficient of each term in the resulting quadratic polynomial is the sum of the coefficients of the corresponding terms in A_i and D. However, we note that there are uncorrelated randomness terms (denoted by $R0$ and $R1$) that also need to be combined and represented back into the quadratic model as denoted by equation 2.43. Since the two random variables $R0$ and $R1$ are standard-normal, we choose to approximate them as another standard-normal random variable R. We can compute the coefficient s such that the variance of $R0$ and $R1$ is preserved as shown below. The entire SUM operation can therefore be represented as

$$
\begin{aligned}
A_{1o} &= A_1 + D && (2.47)\\
&= quad(Y1, ..., Y12) + s_0 R_0 + s_1 R_1 && (2.48)\\
&= quad(Y1, ..., Y12) + sR && (2.49)\\
where \quad & s^2 = s_0^2 + s_1^2
\end{aligned}
$$

Similarly, for each fanin i, we denote the result of the SUM operation by A_{io}:

$$A_{1o} = A_1 + D \tag{2.50}$$

... ...

$$A_{Ko} = A_K + D \tag{2.51}$$

We note that this is an accurate computation in the underlying parameter variations and the only approximation that has been made in the uncorrelated random component as shown above.

MAX Operation

We perform a MAX of K quadratics to get the arrival time signal A_o at the output of the gate. We would like to represent A_o as a quadratic too. Since all timing variables are represented as a quadratic in global variables, the correlations are effectively captured.

$$A_o = MAX(A_{1o}, A_{2o}, ..., A_{Ko}) \quad (2.52)$$

$$= quad(Y1, Y2, ..., Y12) + s_o Ro \quad (2.53)$$

It is known that the MAX operation introduces the complexity in STA. It is very hard to efficiently generate an accurate result of the MAX operation. We propose a regression based strategy to compute the resulting quadratic A_o by performing least square fitting. Assuming we know the degree of the quadratic that we want A_o to be approximated in, least square fitting will try to find the best quadratic that has the smallest error with the actual data of the MAX operation. We are trying to approximate A_o with a quadratic as indicated in equation 2.54. We need to evaluate all coefficients such that the resulting quadratic has smallest error when compared with the actual MAX data. We note that equation 2.54 has K terms for the uncorrelated randomness terms denoted by R. Since we are trying to perform least square fitting on K arrival times, from each of them, we get one uncorrelated randomness term R. However, as will be demonstrated later in the section, we will recombine all these K terms to represent A_o back in the form as shown above in equation 2.53.

$$A_o = c_1 Y1 + c_2 Y2 + ... + c_{12} Y12 + c_{13} Y1^2 + ... + c_{24} Y12^2 + \ 66 \ degree\_2 \ crossterms \ + c_{91} + \sum_{i=1}^{K} s_i Ri \quad (2.54)$$

Now we will formalize the regression strategy that is used to compute these

coefficients. Let us assume that we are given a set of n sampling vectors for the parameters $(Y1, ..., Y12, R1, .., RK)$ and denote the ith sampling vector as $(Y1_i, ..., Y12_i, R1_i, .., RK_i)$ (these n samples will not be a very large set). We can evaluate the exact value of the MAX result at these n sampling vectors. This could be done by evaluating all the quadratics A_{io} and calculating their MAX. Let the ith value be represented by z_i. We can define a residual Res for least square fitting as

$$Res^2 = \sum_{i=1}^{n} \Big[z_i - (c_1 Y1_i + ... + c_{12} Y12_i + c_{13} Y1_i^2 + ...+ \\ c_{24} Y12_i^2 + \ 66\ degree\_2\ cross\_terms\ +\ c_{91} + \sum_{j=1}^{K} s_j Rj) \Big]^2 \quad (2.55)$$

This residue essentially is the root mean square error between the actual data of MAX z_i and the one predicted by the quadratic. In order of minimize the residual, we evaluate the partial derivative wrt. each coefficient in the quadratic and equate the result to zero. This can be represented as :

$$\frac{\partial(Res^2)}{\partial c_1} = -2\sum_{i=1}^{n}\Big[z_i - (c_1 Y1_i + ...)\Big]Y1 = 0 \quad (2.56)$$

$$\frac{\partial(Res^2)}{\partial c_2} = -2\sum_{i=1}^{n}\Big[z_i - (c_1 Y1_i + ...)\Big]Y2 = 0 \quad (2.57)$$

$$.... = ...$$

$$\frac{\partial(Res^2)}{\partial c_{13}} = -2\sum_{i=1}^{n}\Big[z_i - (c_1 Y1_i + ...)\Big]Y1^2 = 0 \quad (2.58)$$

$$.... = ...$$

$$\frac{\partial(Res^2)}{\partial c_{91}} = -2\sum_{i=1}^{n}\Big[z_i - (c_1 Y1_i + ...)\Big]1 = 0 \quad (2.59)$$

$$\frac{\partial(Res^2)}{\partial s_1} = -2\sum_{i=1}^{n}\Big[z_i - (c_1 Y1_i + ...)\Big]R1 = 0 \quad (2.60)$$

$$.... = ...$$

$$\frac{\partial(Res^2)}{\partial s_K} = -2\sum_{i=1}^{n}\Big[z_i - (c_1 Y1_i + ...)\Big]RK = 0 \quad (2.61)$$

We can re-organize these to get equations :

$$c_1\sum_{i=1}^{n}Y1_iY1_i + c_2\sum_{i=1}^{n}Y2_iY1_i + ... + c_{13}\sum_{i=1}^{n}Y1_i^2Y1_i + ... + c_{91}\sum_{i=1}^{n}Y1_i + s_1\sum_{i=1}^{n}R1_iY1_i + ... + s_K\sum_{i=1}^{n}RK_iY1_i = \sum_{i=1}^{n}z_iY1_i \quad (2.62)$$

$$c_1\sum_{i=1}^{n}Y1_iY2_i + c_2\sum_{i=1}^{n}Y2_iY2_i + ... + c_{13}\sum_{i=1}^{n}Y1_i^2Y2_i + ... + c_{91}\sum_{i=1}^{n}Y2_i + s_1\sum_{i=1}^{n}R1_iY2_i + ... + s_K\sum_{i=1}^{n}RK_iY2_i = \sum_{i=1}^{n}z_iY2_i \quad (2.63)$$

....

$$c_1\sum_{i=1}^{n}Y1_iY1_i^2 + c_2\sum_{i=1}^{n}Y2_iY1_i^2 + ... + c_{13}\sum_{i=1}^{n}Y1_i^2Y1_i^2 + ... + c_{91}\sum_{i=1}^{n}Y1_i^2 + s_1\sum_{i=1}^{n}R1_iY1_i^2 + ... + s_K\sum_{i=1}^{n}RK_iY1_i^2 = \sum_{i=1}^{n}z_iY1_i^2 \quad (2.64)$$

....

$$c_1 \sum_{i=1}^{n} Y1_i + c_2 \sum_{i=1}^{n} Y2_i + ... + c_{13} \sum_{i=1}^{n} Y1_i^2 + ... \\ + c_{91} n + s_1 \sum_{i=1}^{n} R1_i + ... + s_K \sum_{i=1}^{n} RK_i = \sum_{i=1}^{n} z_i \quad (2.65)$$

$$c_1 \sum_{i=1}^{n} Y1_i R1_i + c_2 \sum_{i=1}^{n} Y2_i R1_i + ... + c_{13} \sum_{i=1}^{n} Y1_i^2 R1_i + ... \\ + c_{91} \sum_{i=1}^{n} R1_i + s_1 \sum_{i=1}^{n} R1_i R1_i + ... + s_K \sum_{i=1}^{n} RK_i R1_i = \sum_{i=1}^{n} z_i R1_i \quad (2.66)$$

....

$$c_1 \sum_{i=1}^{n} Y1_i RK_i + c_2 \sum_{i=1}^{n} Y2_i RK_i + ... + c_{13} \sum_{i=1}^{n} Y1_i^2 RK_i + ... \\ + c_{91} \sum_{i=1}^{n} RK_i + s_1 \sum_{i=1}^{n} R1_i RK_i + ... + s_K \sum_{i=1}^{n} RK_i RK_i = \sum_{i=1}^{n} z_i RK_i \quad (2.67)$$

We can combine these equations to give a more compact matrix representation as:

$$\begin{pmatrix} \sum_{i=1}^{n} Y1_i Y1_i & \dots & \sum_{i=1}^{n} Y1_i & \sum_{i=1}^{n} R1_i Y1_i & \dots & \sum_{i=1}^{n} RK_i Y1_i \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} Y1_i Y1_i^2 & \dots & \sum_{i=1}^{n} Y1_i^2 & \sum_{i=1}^{n} R1_i Y1_i^2 & \dots & \sum_{i=1}^{n} RK_i Y1_i^2 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} Y1_i & \dots & n & \sum_{i=1}^{n} R1_i & \dots & \sum_{i=1}^{n} RK_i \\ \sum_{i=1}^{n} Y1_i R1_i & \dots & \sum_{i=1}^{n} R1_i & \sum_{i=1}^{n} R1_i R1_i & \dots & \sum_{i=1}^{n} RK_i R1_i \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} Y1_i RK_i & \dots & \sum_{i=1}^{n} RK_i & \sum_{i=1}^{n} R1_i RK_i & \dots & \sum_{i=1}^{n} RK_i RK_i \end{pmatrix}$$

$$\times \begin{pmatrix} c_1 \\ \dots \\ c_{91} \\ s_1 \\ \dots \\ s_K \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} z_i Y1_i \\ \dots \\ \sum_{i=1}^{n} z_i \\ \sum_{i=1}^{n} z_i R1_i \\ \dots \\ \sum_{i=1}^{n} z_i RK_i \end{pmatrix}$$

Essentially, we have represented the quadratic regression as the system $YC = Z$ where we need to solve for the C matrix. Any well known technique for solving a

system of matrices can be used here. This approach essentially selects the coefficients in such a way that the quadratic approximation of A_o has minimum error with the real data set z_i. This quadratic re-approximation is performed every time a MAX operation is computed.

We have computed A_o as a quadratic in the underlying parameters (denoted by Yi) and the uncorrelated randomness terms (denoted by Rj) as shown by equation 2.68. Since we know that the variables representing the uncorrelated randomness are standard-normal, we can combine and re-approximate them as one standard-normal variable as shown by equation 2.69. We can compute the coefficient s_o such that we preserve the variance of the uncorrelated randomness terms in 2.68 by using $s_o^2 = \sum_{j=1}^{K} s_j^2$. Therefore, as shown by equation 2.69, we can represent the result of the MAX operation back into the out timing model as shown.

$$A_o = quad(Y1, Y2, ..., Y12) + \sum_{j=1}^{K} s_j Rj \tag{2.68}$$

$$A_o = quad(Y1, Y2, ..., Y12) + s_o Ro \qquad where \qquad s_o^2 = \sum_{j=1}^{K} s_j^2 \tag{2.69}$$

The regression strategy used in MAX operation has a computational complexity that depends on the size n of the sampling values. Increasing the number of samples at each MAX operation increases the computational cost of this operation but improves the accuracy of the quadratic fit.

We also point out here that the generality of our STA approach to handle all kinds of parameter variation distributions, gate delay distributions and arrival time

distributions is made possible by not making any distribution based approximation in the MAX operation. Quadratic regression can be applied to any arbitrary distribution of the parameter variables YK and the accuracy controlled through the number of sampling vectors used.

After the topological traversal of the circuit, the arrival time at the primary output is represented as a quadratic in global variables. It can be seen that we have presented a generic statistical timing methodology that is not constrained by any assumptions on underlying distribution.

Strategy for Generating Samples for Regression

We would like to elaborate on the sampling strategy that has been used in our proposed STA scheme. The MAX operation as discussed in subsection 2.5.2 performs least square fitting based regression to compute the output arrival time A_o. In order to implement the regression strategy, we need to compute a set of n sampling vectors for the variables representing the parameters $(Y1, ..., Y12, R1, .., RK)$.

Sampling can be done in two ways, the first one being distribution dependent probabilistic sampling. If we are given the distribution of each of the random variables $(Y1, ..., Y12, R1, .., RK)$, we can use the density functions to generate samples. Regression performed through these samples would be a probabilistic fit which has the potential to be very accurate as compared to a distribution independent sampling scheme. A second way for sampling is to perform a distribution independent sampling which we call algebraic sampling. Here, we do not make any assumption

about the distribution of the random variables $(Y1, ..., Y12, R1, .., RK)$. We generate samples along the range of the variables without taking into consideration the probability of the occurrence of that sample. Effectively, this is the same as performing a uniform distribution based sampling. As opposed to probabilistic sampling, in this case we can end up trying to minimize the least square fitting error at points that have a very low probability of occurrence.

We have chosen to perform algebraic sampling in this work and our experimental results show that we are extremely accurate compared with Monte-Carlo simulations. Apart from being independent of the distribution of the parameters, we are also able to use fewer samples to capture the distribution making our scheme more efficient.

2.5.3 Reducing Complexity in Quadratic Regression

We note that the computational complexity in quadratic STA comes primarily from the MAX operation as described in section 2.5.2. Hence, this step becomes the run-time determining step of the STA scheme. Ideally, we would like to maintain the accuracy obtained from using a quadratic models while keeping a runtime that is comparable to an STA scheme with linear delay/arrival time models. The advantage of using regression is the generality in the scheme to handle timing distributions of any nature (not Gaussian only) and the mathematical accuracy inherent in regression. In order to achieve the desired level of accuracy as well as runtime behavior, we propose a scheme that uses linear-modeling based STA to drive the

quadratic STA.

Linear Regression Driven Quadratic STA

Quadratic modeling based STA is more accurate in generating the PDF/CDF of arrival time distribution because of two primary reasons: firstly, because quadratic gate delay modeling is better able to capture the nature of distribution due to the underlying parameter variation and secondly, because quadratic arrival time modeling is able to represent the PDF/CDF more accurately than linear modeling. However, the mean and variance of the arrival time distributions are captured with reasonable accuracy in the linear modeling based STA. We will now propose a quadratic modeling based STA technique that is driven by linear modeling based STA (which has lower runtime).

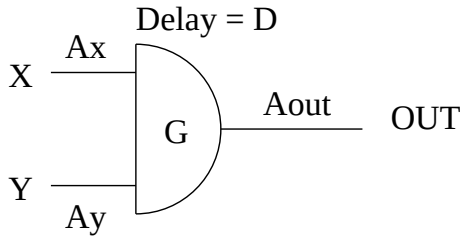


Figure 2.19: STA technique at Gate G

We traverse the circuit topologically and at each gate, we run linear STA and then use linear STA results to drive quadratic STA. Linear STA corresponds to performing linear regression assuming a linear model for arrival time and gate delay. Thus, we generate and store both linearly and quadratically modeled timing values at each gate. Let us suppose we are evaluating the arrival time at output of gate G

with two fanins (X and Y) as shown in figure 2.19. For each input X and Y, we are given both linear and quadratic modeling values for the signal arrival times A_x and A_y respectively. Let us denote the linear arrival times as A_x^l and A_y^l respectively and the quadratic arrival times as A_x^q and A_y^q respectively. The linear and quadratic models for gate delays are given as D^l and D^q respectively.

The linear arrival time A_{out}^l at the output of gate G is given by:

$$A_{out}^l = MAX(A_x^l + D^l, A_y^l + D^l) \tag{2.70}$$

During linear STA we perform regression based MAX operation based on linear gate delay and arrival time models as given by equation 2.70. In section 2.5.2, we have discussed the details of the proposed regression based STA scheme. This enables the time consuming regression in the MAX step (section 2.5.2) to be much faster than the quadratic case. The linear regression output gives us the arrival time (A_{out}^l) at the output of gate G as a linear combination of parameters:

$$A_{out}^l = c_0 + c_1 Y1 + c_2 Y2 + ... + c_{12} Y12 + s_o Ro \tag{2.71}$$

where Y1, Y2,....., Y12 are the independent parameter variables as discussed in section 3.1.1. We know the distribution of these random variables and hence can calculate the mean and the variance of the arrival time A_{out}^l as:

$$Mean(A_{out}^l) = c_0 + c_1 * Mean(Y1) + ... + c_{12} * Mean(Y12) \tag{2.72}$$

$$Var(A^l_{out}) = c_1^2 * Var(Y1) + c_2^2 * Var(Y2) + ... + c_{12}^2 * Var(Y12) + s_o^2 \quad (2.73)$$

We will now assume that the mean and variance of the output arrival time after linear regression is accurate. We will run quadratic STA by matching the mean and variance (first two moments) of the quadratic arrival time with the linear regression output. Let us now understand the scheme in more detail.

The quadratic arrival time at the output of gate G (say A^q_{out}) is given by:

$$A^q_{out} = MAX(A^q_x + D^q, A^q_y + D^q) \quad (2.74)$$

where A^q_x and A^q_y are the signal arrival times at the input-pins X and Y respectively and D^q is the quadratic gate delay. Now let us suppose that we know the probability p such that arrival time $(A^q_x + D^q) \geq$ arrival time $(A^q_y + D^q)$. We can calculate the probability $p = Prob(A^q_x + D^q \ \geq \ A^q_y + D^q)$ during the linear STA run at gate G.

We can run quadratic STA on gate G by utilizing this probability p to generate an output quadratic A^q_{out}, which will then be scaled to match its first two moments to the values evaluated from linear regression based STA as given by equations 2.72 and 2.73. Let the output arrival time quadratic A^q_{out} be generated as follows:

$$A^q_{out} = p * (A^q_x + D^q) + (1 - p) * (A^q_y + D^q) \quad (2.75)$$

where A^q_x and A^q_y are the quadratic arrival times of the signal at the fanin pins X and Y (which have already been calculated previously). After this step, we

need to match the variance of A^q_{out} to the variance of A^l_{out} from linear regression. A quadratic A^q_{out} can be given by:

$$A^q_{out} = c_0 + c_1 Y1 + ... + c_{12} Y12 + c_{13} Y1^2 + ... + c_{24} Y12^2 + \ 66\ degree - 2\ cross - terms + s_x Rx + s_y Ry \quad (2.76)$$

We can again combine the uncorrelated random terms (Rx and Ry) into one term (Ro) by matching their variance as shown before

$$A^q_{out} = c_0 + c_1 Y1 + ... + c_{12} Y12 + c_{13} Y1^2 + ... + c_{24} Y12^2 + \ 66\ degree - 2\ cross - terms + s_o Ro \quad (2.77)$$

Since we know the distribution of each underlying parameter variation ($Y1$ to $Y12$), we know their mean and variance values. We can evaluate the mean and variance of A^q_{out} as follows:

$$Mean(A^q_{out}) = c_0 + c_1 * Mean(Y1) + ... + c_{12} * Mean(Y12) + c_{13} * Mean(Y1^2) + ... other\ terms \quad (2.78)$$

$$Var(A^q_{out}) = c_1^2 * Var(Y1) + ... + c_{12}^2 * Var(Y12)$$
$$+ c_{13}^2 * Var(Y1^2) + ... + c_{24}^2 * Var(Y12^2)$$
$$+ c_{25}^2 * Var(Y1 * Y2) + ...cross\ terms + s_o^2$$
$$+ 2c_1c_2 * Cov(Y1, Y2) + 2c_1c_3 * Cov(Y1, Y3)$$
$$+ ..... \ all\ other\ covariance\ terms \quad (2.79)$$

We will first match the variance of A^q_{out} (from equation 2.79) with that of A^l_{out} (from equation 2.73) by scaling A^q_{out} with a factor α such that:

$$\alpha^2 = Var(A^l_{out})/Var(A^q_{out}) \quad (2.80)$$

$$A^{p'}_{out} = \alpha * A^q_{out} \quad (2.81)$$

The mean of the new scaled quadratic will be:

$$Mean(A^{q'}_{out}) = \alpha * Mean(A^q_{out}) \quad (2.82)$$

Hence, to match the mean of the quadratic arrival time expression with that obtained from linear regression (equation 2.72), we can add a constant factor β to the constant term c_0 of $A^{q'}_{out}$ such that:

$$\beta = Mean(A^l_{out}) - Mean(A^{q'}_{out}) \quad (2.83)$$

$$c'_0 = c_0 + \beta \quad (2.84)$$

Hence the final quadratic arrival time at the output of gate G can be given by A_{out}^{quad}:

$$A_{out}^{quad} = \alpha * A_{out}^{q} + \beta \tag{2.85}$$

This completes our linear regression driven quadratic STA technique. We have avoided the complexity of solving a large quadratic regression problem at each gate (during the MAX operation) by solving a smaller linear regression problem and then performing moment matching (first two moments) as explained in this section. The runtime complexity of this scheme will be of the order of the runtime for linear regression.

2.5.4 Experimental Results

The proposed STA framework was implemented in SIS [37]. For the gate delay model in equation 2.86, we assumed that threshold voltage (V_{th}) is the underlying sources of variability. We used an academic placement tool (CAPO [5]) to get a valid placement for each benchmark. This placement information was used to generate the V_{th} variations at each gate as indicated in equation 2.38. This automatically captures correlations due to spatial proximity. We imposed a 15% variability on V_{th} with a mean value of 0.5V. All experiments were run on a Sunblade 150 machine with 512mb RAM.

$$D_i \propto \frac{C_L V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \tag{2.86}$$

Benchmark	Monte Carlo	Linear STA		Quadratic STA		Linear-Driven Quadratic STA	
	Runtime	Runtime	Speedup	Runtime	Speedup	Runtime	Speedup
C432	1919	129	14.9	382	5.0	134	14.3
C499	6132	257	23.9	773	7.9	270	22.7
C880	5171	231	22.4	679	7.6	241	21.5
C1355	6030	251	24.0	777	7.8	270	22.3
C1908	8310	272	30.6	815	10.2	282	29.5
C3540	39805	714	55.7	2069	19.2	758	52.5
C5315	86575	985	87.9	2084	41.5	1088	79.6
C6288	223945	1715	130.6	4838	46.3	1898	117.9
i2	3052	147	20.8	428	7.1	148	20.6
i4	2286	88	26.0	258	8.9	94	24.3
i5	2975	92	32.3	271	10.9	96	30.9
i6	15933	255	62.5	763	20.9	271	58.8
i7	28977	355	81.6	1014	28.6	375	77.3
i8	36183	620	58.4	1804	20.1	659	54.9
i9	36183	357	47.2	1048	16.1	383	44.0
i10	223021	1378	161.8	3862	57.7	1554	143.5
Average			55X		20X		51X

Table 2.2: Runtime Comparison wrt Monte Carlo (Global Parameters have a Uniform Distribution)

The quadratic model for each gate delay was generated using best fit regression with Monte Carlo data. The Monte Carlo data for the gate delay was calculated using the delay model indicated in equation 2.86 with different parameter instances. In order to compare the runtime and error of our STA results, we generated accurate timing CDFs for each benchmark using equation 2.86 for gate delays through Monte Carlo simulations.

We experimented with the following cases:

1. Using linear gate delay and arrival time models, we performed regression based STA (as described in section 2.5.2). This approach is similar to the one proposed by state of the art STA techniques like [26, 42].

2. Using quadratic gate delay and arrival time models, we performed regression based STA (as described in section 2.5.2).

3. We performed quadratic STA using our proposed linear regression driven quadratic STA scheme (as described in section 2.5.3). The aim here is to show that we can maintain the accuracy obtained from quadratic models while keeping the runtime comparable to linear regression based STA.

We performed two sets of experiments, assuming the underlying global parameters to have a uniform and Gaussian distribution respectively.

Uniform Distribution of Global Parameters

In this set of experiments, all global parameters were assumed to have a uniform distribution. We imposed a 15% variability on V_{th} with a mean value of 0.5V.

Benchmark	Linear	Quad	Lin-Quad
	(rms err)	(rms err)	(rms err)
C432	0.0474	0.0019	0.0049
C499	0.0496	0.0025	0.0031
C880	0.0489	0.0012	0.0012
C1355	0.0499	0.0013	0.00013
C1908	0.0479	0.0010	0.0011
C3540	0.0476	0.0009	0.0009
C5315	0.0434	0.0026	0.0029
C6288	0.0437	0.0011	0.0011
i2	0.051	0.0013	0.0014
i4	0.046	0.0013	0.0023
i5	0.00475	0.0009	0.0009
i6	0.0472	0.0008	0.0009
i7	0.0504	0.0011	0.0011
i8	0.0485	0.0009	0.0010
i9	0.0498	0.0010	0.001
i10	0.0485	0.0008	0.0009
Average	0.0453	0.0013	0.0016

Table 2.3: RMS Error Comparison wrt Monte Carlo CDFs (Global Parameters have a Uniform Distribution)

Tables 2.2 and 2.3 present the experimental results. All runtime and error comparisons are made wrt. Monte Carlo simulations. In table 2.2, columns 2, 3, 5 and 7 present the runtime for Monte-Carlo, linear STA, quadratic STA and linear-driven quadratic STA respectively. The corresponding speedups are given in columns 4, 6 and 8 respectively. It can be seen that on an average, we get $55x$, $20x$ and $51x$ speedup compared with Monte Carlo runtime for the three schemes respectively. We note that the runtime complexity of the novel linear-driven quadratic regression scheme is similar to that of linear regression driven STA with the benefit of allowing us to use a more accurate quadratic modeling framework.

Table 2.3 presents the root mean square (rms) error in the output CDFs from the three schemes as compared with the accurate CDFs from Monte-Carlo simulations. On an average there is 0.0453, 0.0013 and 0.0016 units of rms error in the CDFs obtained from linear STA, quadratic STA and linear-driven quadratic STA respectively. It is evident that quadratic modeling gives us more accuracy by capturing the variability better during STA. We also note that the error obtained from linear-driven quadratic STA is also very similar to that of quadratic STA. This points out the effectiveness of our proposed scheme in being able to capture the accuracy provided by quadratic modeling of gate delays and arrival times while being able to maintain a runtime complexity similar to that of linear STA.

Even though the rms error numbers are small in magnitude, they can make a significant impact on the CDF. For example, the average rms error in linear regression scheme is 0.045 units, so if we are looking at the 50 percentile point on the accurate CDF, the predicted CDF potentially be showing a value of either

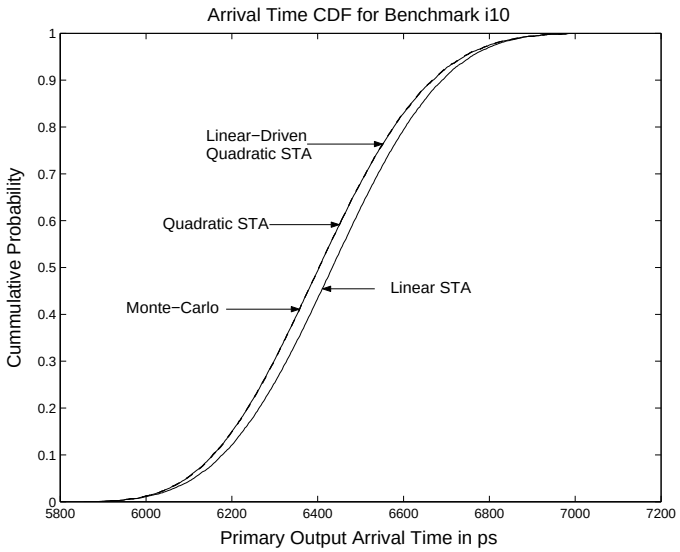


Figure 2.20: CDF Result for i10 at a primary output

0.455 or 0.545, which is a very significant difference from the actual value of 0.5. The impact of this inaccuracy on decisions made on the design during optimization phase using these CDFs could be very drastic. Figure 2.20 depicts the CDF at the output of benchmark $i10$. The CDFs obtained from quadratic regression based STA and linear-driven quadratic STA are almost co-inciding with that obtained from Monte-Carlo and have a very small rms error. Hence, linear regression driven quadratic STA provides high accuracy as compared with Monte Carlo even though its runtime is comparable with that of linear regression.

Gaussian Distribution of Global Parameters

In order to evaluate the accuracy of our scheme on different distributions of the underlying parameter variations, we also ran a set of experiments assuming the parameters to have Gaussian distribution. We imposed a 15% variability on V_{th} with

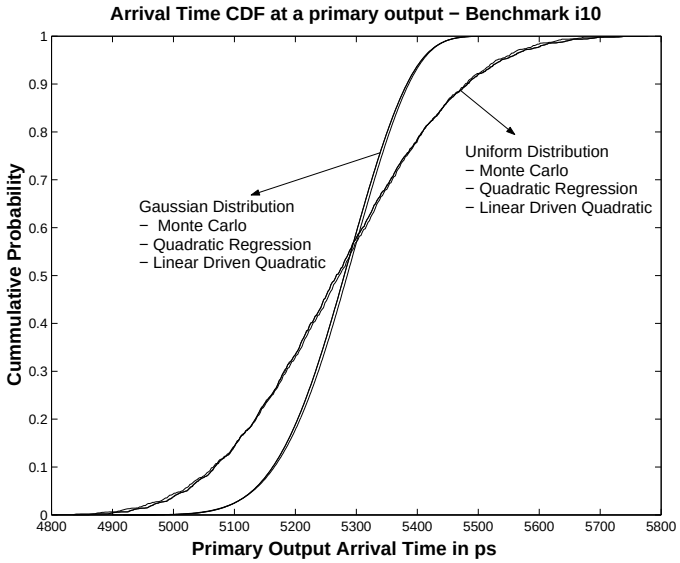


Figure 2.21: CDF Result for i10 at a primary output

a mean value of 0.5V, which implies that the 3σ interval was taken to be $0.15 * 0.5$V. All other experimental conditions were kept the same as before.

In table 2.4, columns 2, 3, 5 and 7 present the runtime for Monte-Carlo, linear STA, quadratic STA and linear-driven quadratic STA respectively. The corresponding speedups are given in columns 4, 6 and 8 respectively. It can be seen that on an average, we get $58x$, $21x$ and $52x$ speedup compared with Monte Carlo runtime for the three schemes respectively.

Table 2.5 presents the *rms* errors in the output CDFs from the three schemes as compared with the accurate CDFs from Monte-Carlo simulations. On an average there is 0.0319, 0.0011 and 0.0014 units of *rms* error in the CDFs obtained from linear STA, quadratic STA and linear-driven quadratic STA respectively.

Figure 2.21 shows the comparison of the CDFs obtained at one of the primary outputs for benchmark $i10$. It can clearly be seen that our proposed schemes (both

Bench	Monte Carlo	Lin STA		Quad STA		Lin-Driven Quad STA		STA from [43]	
mark	Runtime	Runtime	Speedup	Runtime	Speedup	Runtime	Speedup	Runtime	Speedup
C432	2010	135	14.9	388	5.1	144	13.9	369	5.4
C499	6111	269	22.7	762	8.0	281	21.7	711	8.6
C880	5398	227	23.8	691	7.8	264	20.4	621	8.7
C1355	6211	248	25.0	773	8.0	328	18.9	688	9.0
C1908	8240	262	31.5	828	9.9	279	29.5	729	11.3
C3540	40136	710	56.5	2018	19.9	746	53.8	1925	20.8
C5315	86200	1002	86.0	2155	40.0	1117	77.2	2452	35.2
C6288	225018	1792	125.6	4892	45.9	1853	121.4	4207	53.5
i2	3078	153	20.1	426	7.2	1464	2.1	390	7.9
i4	2262	91	24.9	252	8.9	104	21.8	271	8.3
i5	3077	99	31.1	288	10.7	110	27.9	251	12.3
i6	16193	249	65.0	771	21.0	274	59.1	689	23.5
i7	29045	372	78.1	1068	27.2	391	74.3	898	32.3
i8	36330	631	57.6	1839	19.8	685	53.1	1643	22.1
i9	35941	369	97.4	1036	34.7	388	92.6	882	40.7
i10	223936	1401	159.8	3911	57.3	1573	142.4	3571	62.7
Average			58X		21X		52X		23X

Table 2.4: Runtime Comparison wrt Monte Carlo (Global Parameters have a Gaussian Distribution)

quadratic regression based STA and linear regression drive quadratic STA) are very accurate as compared with Monte-Carlo simulations. These results support our claim that our scheme works well for all kinds of underlying distributions of global parameters.

Comparison with Work in Current Literature

In recent literature, several researchers have proposed different schemes to address the problem on non-linear non-Gaussian statistical timing analysis [127, 30, 43]. An initial version of our work has also been proposed in [119]. In [43], the authors use a canonical timing model where they represent the arrival time as a sum of

linear Gaussian parameters and a function of non-linear/non-Gaussian parameters. In order to compute the MAX operation they perform numerical integration on the non-linear non-Gaussian terms of the canonical model.

We implemented the scheme as proposed in [43]. For our experiments, we assumed that all the independent global parameters had a Gaussian distribution. This was done in order to allow the scheme in [43] to model some parameters to be linear-Gaussian and others to be non-linear or non-Gaussian. The STA scheme proposed in our work does not make any such assumption. We use a canonical quadratic timing model as explained earlier in this work, while the scheme in [43] proposes to model the non-linear non-Gaussian parameter dependencies through a non-linear function. In order to make a fair comparison between the two schemes, we assume that this non-linear function is also a quadratic function in the non-linear non-Gaussian parameters. This would ensure that we do not add any extra computational complexity to the scheme proposed in [43]. For each parameter that had a non-linear dependence, we used 7 samples for numerical integration during the MAX operation. We also used the same number of samples in our regression based scheme to keep a fair comparison in terms of computational complexity. Results for runtime and rms error comparison are given in the last columns of tables 2.4 and 2.5 respectively. In general, we can see that the average runtime speedups that we gets from [43] are comparable with that obtained from our pure quadratic based regression scheme. This is expected because the both the schemes are using numerical techniques and the same quadratic delay models. However, our linear regression driven quadratic STA approach gives significant runtime improvements

Benchmark	Linear	Quad	Lin-Quad	[43]
	(rms err)	(rms err)	(rms err)	(rms err)
C432	0.0327	0.0007	0.0018	0.0027
C499	0.0346	0.0009	0.0009	0.0012
C880	0.0338	0.0009	0.0009	0.0035
C1355	0.0345	0.0008	0.0012	0.0013
C1908	0.0333	0.0008	0.0009	0.0012
C3540	0.0331	0.0009	0.0009	0.0011
C5315	0.0252	0.0063	0.0082	0.0091
C6288	0.0304	0.0007	0.0007	0.0073
i2	0.0351	0.0009	0.0009	0.0010
i4	0.0312	0.0010	0.0011	0.0017
i5	0.0279	0.0006	0.0007	0.0007
i6	0.0328	0.0008	0.0008	0.0009
i7	0.0349	0.0009	0.0009	0.0009
i8	0.0314	0.0008	0.0008	0.0010
i9	0.0344	0.0008	0.0009	0.0010
i10	0.0252	0.0006	0.0006	0.0007
Average	0.0319	0.0011	0.0014	0.0022

Table 2.5: RMS Error Comparison wrt Monte Carlo CDFs (Global Parameters have a Gaussian Distribution)

over [43] as is evident from the results in table 2.4. The rms error obtained in the two schemes are comparable as is evident from table 2.5. Hence, as the results indicate we have a faster approach that has comparable error with the scheme presented in [43].

As the technology nodes are scaling, the delay dependence on parameters becomes more non-linear. Variability in parameters may not always be accurately modeled as a Gaussian distribution. In the most general case, where we assume that all parameters can have either a non-Gaussian distribution or a non-linear delay dependence, the scheme in [43] would essentially boil down to numerical computations similar to a Monte-Carlo based STA scheme. This brings out the generality in our approach where no such assumptions have been made.

Chapter 3

Variability-Aware Design Optimization: Design Time Techniques

Variability-Aware design synthesis and optimization has gained a lot of attention in recent years. As discussed in the earlier chapters, randomness due to fabrication and environmental uncertainty causes severe degradation of design performance, yield and power. Traditional deterministic techniques for handling variability are not able to efficiently adapt to managing the variability problem in nanoscale designs. There is a shift of paradigms in trying to develop variability-aware design synthesis and optimization schemes, where the idea is to explicitly consider the impact of randomness during the design flow in order to maximize the likelihood of the design in meeting its constraints.

There is a lot of work that has been done in the area of statistical/probabilistic design optimization. Several researchers have worked on the problem of variability-aware buffer insertion [115, 7, 69] where the classical physical synthesis optimization step of interconnect optimization is re-evaluated in a probabilistic paradigm to consider uncertainty in both wire-lengths as well as parasitics. Statistical gate sizing [117, 8, 73, 67, 80, 32] has also been an active area of research in recent years and several works have proposed techniques to perform gate sizing while considering the impact of variability for yield improvements. With leakage power becoming a

significant challenge in nanoscale designs due to its exponential sensitivity to variability, a lot of research has also focused on statistical leakage power optimization [97, 89, 77, 73, 83].

In this chapter, we present our research contributions in the area of design-time optimization considering variability. We have looked at several different problems that are applicable during physical synthesis and also develop general frameworks that can be used to model several optimization problems in VLSI-CAD. We specifically present two research problems in this chapter, namely:

1. **A Probabilistic Approach to Buffer Insertion**

   This work presents a formal probabilistic approach for solving optimization problems in design automation. Prediction accuracy is very low especially at high levels of design flow. This can be attributed mainly to unawareness of low level layout information and variability in fabrication process. Hence a traditional deterministic design automation approach where each cost function is represented as a fixed value becomes obsolete. A new approach is gaining attention [98, 25, 6, 14, 75, 10, 27, 2] in which the cost functions are represented as probability distributions and the optimization criteria is probabilistic too. This design optimization philosophy is demonstrated through the classic buffer insertion problem [76]. Formally, we capture wirelength as probability distributions (as compared to the traditional approach which considers wirelength as fixed values) and present several strategies for optimizing the probabilistic criteria. During the course of this work many problems are proved to be

NP-Complete. Comparisons are made with the Van-Ginneken "optimal under fixed wire-length" algorithm. Results show that the Van-Ginneken approach generated delay distributions at the root of the fanout wiring tree which had large probability (0.91 in the worst case and 0.55 on average) of violating the delay constraint. Our algorithms could achieve almost 100% probability of satisfying the delay constraint with similar buffer penalty. Although this work considers wirelength prediction inaccuracies, our probabilistic strategy could be extended trivially to consider fabrication variability in wire parasitics.

2. **Monte-Carlo Driven Stochastic Optimization Framework for Handling Fabrication Variability**

Increasing effects of fabrication variability have inspired a growing interest in statistical techniques for design optimization. Most techniques that have been proposed in existing literature make assumptions on the nature of the distribution (generally taken to be Gaussian) of the circuit parameters that are affected by variability. This is done in order to exploit the existing analytical results that can be used to solve the problem efficiently. But as the impact of fabrication variability increases, we need to develop efficient techniques that can capture the true distribution of the parameters. One way to accomplish this is to develop efficient Monte-Carlo driven stochastic optimization techniques. In this work, we investigate the classic linear programming problem as applied to VLSI CAD from a stochastic perspective. To this end, we investigate

stochastic programming formulations and present techniques like Successive Sample Mean Optimization SSMO and Stochastic Decomposition to solve the same. Stochastic programming presents a very strong framework for solving linear programs in which the parameters behave as random variables. In this framework we consider Binning-Yield Loss as an optimization objective. The proposed formulation can be solved to get provably optimal solution under a convex binning-yield loss function. We modeled the MTCMOS sizing problem [116] as a stochastic program and solved it using 1) traditional deterministic linear programming ignoring variability, 2) SSMO and 3) Stochastic Decomposition. Results showed that Stochastic Decomposition generates solutions that always satisfied the timing constraints even in presence of variability. SSMO was very slow and did not converge in many cases while the deterministic techniques violated the timing constraints with a 48% probability.

We will now discuss each of these works in more detail.

3.1 A Probabilistic Approach to Buffer Insertion

Design automation of integrated systems is essentially optimization driven by estimation. If the estimation of critical design objectives is inaccurate, the optimality of optimization will be limited. Unfortunately, estimation is always marred with inaccuracies which occur due to many factors. Unawareness of exact implementation information, unpredictable circuit behavior, fabrication variability are important ones among them. Traditionally, optimization in design automation has been deterministic since it assumes a fixed value to the pertinent cost function (like area, delay, power). Lately, a new optimization approach is gaining attention [98, 25, 6, 14, 75, 10, 27, 2] in which the cost functions are represented as probability distributions and the optimization criteria is probabilistic too. Such a design methodology would be able to address the issue of prediction uncertainties and fabrication variability in a much more robust fashion when compared with traditional deterministic approach. In this work, we present such an optimization methodology for the buffer insertion problem [76]. We address unpredictabilities posed by wirelength estimation and/or variation of interconnect properties due to fabrication variability [98] and illustrate the superiority of our probabilistic approach over traditional deterministic Van-Ginneken algorithm [76]. Accurate wire-length prediction is in itself an area of research and several interesting approaches have been proposed [12, 33, 34, 50, 52]. Probabilistic models for estimation of net-length distribution can be incorporated into our probabilistic buffer insertion algorithm to estimate the distribution of the wire-lengths of each segment of the wiring tree. More recently,

[9, 68] have proposed wire length estimation models which are directly applicable to this work on probabilistic buffer insertion.

The problem of buffer insertion deals with the placement of buffers at appropriate positions on the fanout wiring trees such that the delay at the driving gate is minimal. In the last few years a lot of work has been done on optimization using buffer insertion [48, 61, 123, 20, 28, 62, 19, 126]. Lukas van Ginneken [76] presented an optimal buffer placement algorithm for RC-Trees under the Elmore Delay model for wires. It was assumed that wirelength of individual segments in the wiring tree are known a-priori through some estimation engine. Estimating wirelength especially in a traditional top down design flow is extremely hard and error prone. This makes estimation of delay and capacitive loading of the individual wire segments extremely difficult. Even if the wirelength estimates are accurate, the fabrication variability/uncertainty makes accurate estimation of wire parasitics an intractable problem. Hence, the traditional deterministic approach possesses serious disadvantages. In this work, we extend Van-Ginneken's approach to consider wirelength estimation inaccuracy by modeling it as probability distributions. We propose a new probabilistic criteria of selecting the final solution. The new criteria computes and minimizes the probability of violating a given delay constraint. This work also presents three algorithms for performing buffer insertion when wire-lengths are assumed as distributions. These three approaches have different pruning criteria (from very relaxed to very strict) and varying runtime complexities. Several sub-problems were proved NP-Complete indicating that finding an optimal solution in polynomial time under the probabilistic wire-length assumption is very hard.

Experiments were conducted on large benchmarks with state of the art technology parameters. Comparisons were made with the Van-Ginneken approach [76]. Results showed that the Van-Ginneken approach generated delay distributions at the root of the wiring tree which had large probability (0.91 in the worst case and 0.55 on average) of violating the delay constraint. Our algorithms could achieve 100% probability of satisfying the delay constraint with similar buffer penalty. This is a very strong result since our approach ensures that the delay constraint will always be satisfied.

Although in our approach we address wire-length predication inaccuracies, our algorithms can be trivially extended to the case where the estimation of wire parasitics is also inaccurate due to fabrication variability.

3.1.1 Motivation

The Probabilistic Paradigm

Automation of integrated systems is marred with estimation inaccuracies which occur due to a combination of many factors. Unawareness of exact layout information like routing, placement, exact logic structure are prominent reasons. Lately fabrication uncertainties have also begun to get considerable weight primarily due to increasing complexity and scaling of the fabrication process. In the light of such unpredictabilities, a traditional deterministic approach towards design automation becomes incapable and obsolete. Basically, a deterministic approach assigns a fixed value to the cost function (like area, delay, power) and does not consider the error

associated with the estimation of this cost function. Hence, very little could be said about the optimality of the final design especially if the estimation was erroneous. This calls for the development of a probabilistic approach towards design optimization. Such an approach models the cost functions as probability distributions and optimizes the design probabilistically, hence maximizing the likelihood of satisfying design constraints. Many researchers have suggested the importance of such an approach [2, 25, 98, 14, 75, 6] since estimation inaccuracies (both due to fabrication variability and layout unawareness) are becoming major bottlenecks in design closure. The main advantage of such an approach would be faster design closure, better fabrication yield (since fabrication variability would have been accounted for during designing) and improved robustness.

In this work we present such a probabilistic approach for the classic buffer insertion problem. We revisit the traditional deterministic buffer insertion approach proposed by Van-Ginneken and reformulate the problem probabilistically. In this work we assume the source of unpredictability to be the inaccuracy in wirelength estimation. Hence we model the wire lengths as probability distributions. We do not assume any particular distribution for the wire-lengths and present a generic approach that is valid for any form of distribution. Our approach can be trivially extended to the case when the parameters of the wires (and not the length itself) change due to fabrication variability.

Wire-Length Distribution Model

Our proposed probabilistic buffer insertion algorithm assumes that we have a wire-length distribution prediction model for each net in the wiring tree. We note here that we propose a generic approach for probabilistic buffer insertion that is not dependent on the underlying distribution of the wire-lengths of each net. Wire-Length prediction is an issue even if we are given a placed netlist. The work in [9] illustrates the importance of accurate wire-length prediction and discusses various issues associated with accurate wire-length prediction. In [9], the authors propose an empirical and parameterizable model for estimating the probability distribution of wire-length for each net in a placed netlist. This model is simple and fast to compute. It takes a placed netlist and router parameters as input and provides the probability distribution of wire-length for each net. Such a wire-length model is totally different from the traditional wire-length estimation models that give a fixed wire-length estimate for each net. This model is empirical in nature and has been validated on state of the art commercial (Cadence QPlace and WRoute) and academic tools (Parquet [94] and Labyrinth [85]). The model has two distinct subcomponents. The first part estimates the wire-length that has the largest probability of occurrence for a given net and the second part estimates the value of this largest probability. These two are combined together to generate the overall probability distribution function for each net. The model proposed in [9] can be given as:

$$P(x) = \begin{cases} 0 & x \leq BB \\ (\frac{P}{PL-BB})(x - BB) & BB < x < PL \\ P \times e^{-l(x-PL)} & x \geq PL \end{cases} \tag{3.1}$$

Here $P(x)$ is the probability for the net having a length x, PL is the Peak Location of the distribution, P is the value of the distribution at the peak location, BB is the half-perimeter bounding box estimate and l is a parameter that captures the rate of decay of the exponential region of the distribution. The probability distribution is modeled as an increasing ramp for lengths between bounding box to the peak location and after that it follows an exponentially decaying trend. The parameters in this model need to be determined to get the wire-length distribution, the details of which are given in [9]. For brevity, we will not go into further details about the model in this work. We note that this model can be applied to our proposed probabilistic buffer insertion framework and has been used in the experiments to generate a wire-length distribution for each net in the wiring tree.

Traditional Buffer Insertion

The Buffer Insertion Problem can be formally stated as:

Given the fanout wiring tree with parasitic resistances and capacitances, wire-lengths, potential buffer locations, sink required times, sink capacitive loads and a delay constraint at the driving gate, the problem is to place buffers into the tree such

that the required arrival time at the input of the driving gate is maximum. We also consider the optimization of the number of buffers used to satisfy the delay constraint.

The buffer insertion problem formalized by [76] models the fanout wiring tree as a set of distributed RC sections. The Elmore Delay model [122] is used to compute the delay of such a wiring tree. Figure 3.19 illustrates a typical wiring tree. Each of the individual wire segments is characterized by parasitic resistances and capacitances (like R3 and C3). These depend on the length of the corresponding wire. A subtree rooted at node k is represented by two numbers: the required arrival time T_k and capacitive loading L_k.

Adding a wire of length l at the root of a subtree affects the T_k,L_k values as :

$$T'_k = T_k - rlL_k - (1/2)rcl^2 \tag{3.2}$$

$$L'_k = L_k + cl \tag{3.3}$$

When a buffer is added at the root of the subtree :

$$T'_k = T_k - D_{buf} - R_{buf}L_k \tag{3.4}$$

$$L'_k = C_{buf} \tag{3.5}$$

When two subtrees rooted at n and m are merged into one subtree :

$$T'_k = min(T_n, T_m) \tag{3.6}$$

$$L'_k = L_n + L_m \tag{3.7}$$

These equations can be used to compute the required arrival time T_o at the root of the wiring tree. For brevity we have omitted the detailed description of this

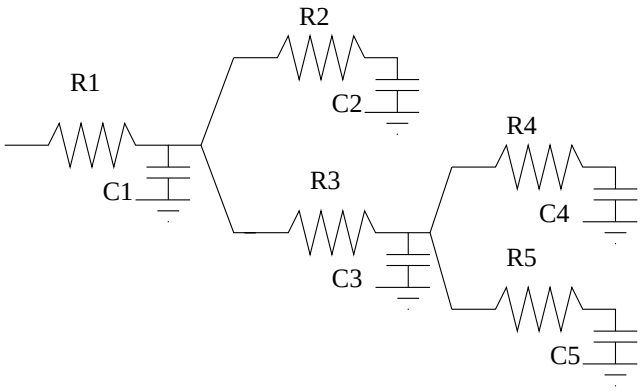


Figure 3.1: RC Tree Network

delay model. This is a very popular delay model for capturing wire-delays in modern layout driven optimization systems. A lot of research has been done on the buffer insertion problem [58, 128, 40, 15, 70] which is especially useful for large global nets like clock trees. Most of these approaches use a dynamic programming based approach in which the wiring tree is traversed topologically from sinks to source while storing an optimal solution set. Next we describe Van-Ginneken approach to buffer insertion which solves the problem optimally for a fixed wiring topology and buffer placement locations.

The Van-Ginneken Algorithm

Using the delay model described above, Van-Ginneken proposed his buffer placement algorithm [76]. The input to his strategy was a wiring tree with estimated parasitics and a set of possible buffer locations, fanout capacitive loadings and required arrival times. The wire parasitics in turn are dependent on the wire-

lengths which are assumed to be prespecified. The problem is to decide buffers locations at the prespecified positions such that the required time at the root is maximized. The Van-Ginneken strategy is optimal under the Elmore Delay model. His algorithm traverses the wiring tree topologically from primary outputs to primary inputs. At each possible buffer location, it evaluates the possibility of adding a buffer and its effect on the required time and capacitive loading at that node. This is followed by local pruning of the generated solutions. The pruning criteria removes those solutions from the set of potential solutions which have another solution with better values of both required time and capacitive loading. Lukas Van-Ginneken[76] proved that this pruning criteria generates an optimal solution at the root in polynomial time.

Shortcomings of Existing Approach

Existing approaches to Buffer Insertion (or any other problem in design automation) do not consider the uncertainties in estimating wire-lengths (or any other pertinent cost function). Focusing the discussion on Buffer Insertion, the Van-Ginneken approach assumes wire-lengths to be prespecified from some estimation engine as fixed values. In reality, no estimation engine can give accurate wire-length predictions. This is due to the unawareness of future optimizations and hence the state of the final design. This is also due to the sensitivity between various cost functions. For example, if congestion in a certain region is intolerable, then any strategy of optimizing congestion could have adverse effects on wire-length. Using

a fixed value of wire-length therefore cannot capture the real variations involved.

In this work we relax the assumption of having fixed wire-length estimates and propose a new buffer insertion strategy which probabilistically models lengths and optimizes the underlying distribution. We are considering the post-placement, pre-routing stage of design flow where the exact wire-lengths are unknown even though the wiring topology is assumed to be known. Without the detailed routing information, it is not possible to evaluate accurate estimates for length of each wire-segment. But before we delve into the details of our algorithm, we quantitatively illustrate the shortcomings in making a fixed wire-length estimate.

Let us assume that we know the distribution of length of the wire-segments. The Van-Ginneken algorithm does not consider distributions, hence we consider the two ways of providing fixed length values to the algorithm:

1. Average length of the distribution

2. Worst (longest) length in the distribution

We conducted experiments with these two wire-length estimates. The Van-Ginneken algorithm was given these estimates and the buffered solution was generated. On this buffered tree, we imposed the real distribution of wire-lengths. Using these distributions, the delay distribution at the root was computed.

Figure 3.2 illustrates the results when the average values from the wire length distribution were given as wire-length estimates. The figure reports the variation in delay at the root for the result generated by the Van-Ginneken algorithm. These plots are shown for different wirelength statistical distributions (tailed Gaussian and

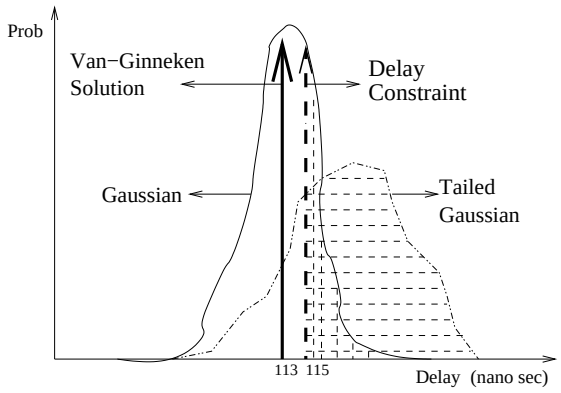


Figure 3.2: Mean Value vs. Actual Delay Distribution

Gaussian). The bold arrow illustrates the delay estimated by the Van-Ginneken algorithm. In reality the delay values at the root are distributions. It can be seen that there is a large portion of these distributions whose delay is greater than that estimated by the Van-Ginneken algorithm. Let us suppose we have a delay constraint illustrated by the dotted line. According to the Van-Ginneken solution this constraint is satisfied, but in reality, there is a large portion of the delay distribution that violates this constraint. This clearly shows that the deterministic fixed wirelength approach can result in failure of design closure.

We also assigned the worst case length values as estimates. Figure 3.3 presents the data for Gaussian wire-length distributions. Such an estimation strategy could be an overestimate and hence an overkill. It can be seen that for the same delay constraint as in figure 3.2, the Van-Ginneken solution (shown in a bold arrow) will not be able to satisfy the constraint. But in reality (by observing the delay distribution of the Van-Ginneken buffered tree), we find that there is a large area which lies within the constraint (the shaded area). Hence the Van-Ginneken result

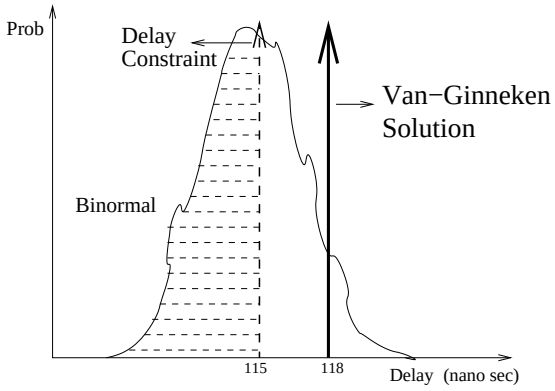


Figure 3.3: Worst Case Length Estimate

would be an overkill both in terms of number of buffers and delay.

What we observed so far was that using fixed values to estimate cost functions does not accurately address the issue of unpredictabilities. In reality, these cost functions should be modeled as distributions and the algorithms should be reformulated to consider these distributions. In this work we present such an approach for the buffer insertion problem.

3.1.2 Probabilistic Buffer Insertion: Metrics

The previous section illustrated the importance of considering probability distributions of cost functions during optimization. Let us consider the following situation in this light. Given a wiring tree with possible buffer locations and a delay constraint at the root, the problem is to place buffers such that the delay constraint is satisfied. Consider two given solutions at the root, each corresponding to different distributions (as illustrated in figure 3.4). The figure also illustrates the delay constraint that needs to be satisfied. Here the distribution with larger spread has

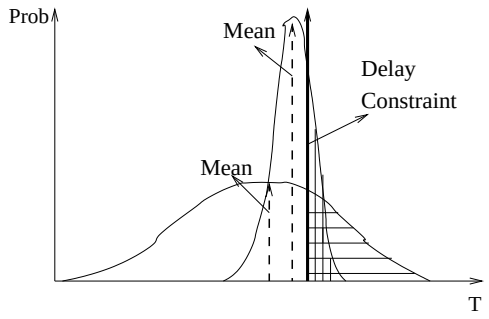


Figure 3.4: Spread in Distribution

lesser mean value for delay compared with the distribution with smaller spread. A traditional approach would choose the distribution with smaller mean (see figure 3.4). Clearly this solution has a large area outside the delay constraint. Hence it has a larger probability of failure. In this situation the second solution with a smaller spread should be the choice. This observation can be formally outlined as follows

$$Minimize \sum_{d \geq D_{cons}} p(d) \tag{3.8}$$

Here

d : Delay p(d): Probability of delay d

Basically, we would like to choose a solution that minimizes the total probability of the delay constraint D_{cons} not being satisfied. This is a probabilistic selection criteria.

3.1.3 Probabilistic Buffer Insertion: Algorithms

In the next few sections we will describe algorithms to optimize the criteria outlined above. The input to our algorithms is a wiring tree with parasitic resistances and capacitances, distributions of wire-lengths (instead of fixed values), possible buffer locations, sink required times, sink capacitive loads and a delay constraint at the root. The root is assumed to be a Nand gate which drives the wiring tree. The delay constraint needs to be satisfied at the input of this gate. We would like to point out here that we have not assumed anything about the distribution of wire-lengths in our formulations. We propose a generic probabilistic optimization approach that will work for any given underlying distribution. We modify the Van-Ginneken approach to consider probability distributions of wire-lengths, the details are outlined below.

The Global Algorithm

The **Global Algorithm** approaches the problem similar to the Van-Ginneken strategy. The RC-Tree network is traversed topologically from sinks towards source. At the root of each subtree (internal node in the network), the set of possible solutions are computed by merging the solutions of fanout-subtrees. Just like the Van-Ginneken approach each solution comprises of two entities: T and L. T corresponds to the required arrival time at the root and L corresponds to the capacitive loading. The difference lies in the fact that since wire-lengths are considered as distributions, both T and L will be distributions too. We do not make any assumption

about the type of distribution of the wire-segments and present a generic approach that is applicable to any distribution. Referring to figure 3.5(b), at any internal node i we have the option of whether to place a buffer or not. In order to compute the potential solutions at i, the solutions at its fanout nodes j and k are propagated to i (by adding the delay distribution of the wires (x and y which connect them to i respectively). This can be done using equations 3.2 and 3.3 (although all variables are distributions now). Now we generate the possible solutions at node i by merging these modified solutions from the fanout nodes using equations 3.6 and 3.7. If node i has two fanouts with m and n solutions each, then there could be a total of mn solutions at i. The possibility of buffer placement at node i, makes this solution set to become $2mn$. Each buffer is characterized by an input capacitance C_{buf}, and internal delay D_{buf} and an output impedance R_{buf}. Adding a buffer modifies the solutions at node i according to equations 3.4 and 3.5. All these equations would be applied on probability distributions of L and T instead of fixed values.

The total solutions at node i can become very large (non-polynomial in problem size). Hence this solution set needs to be pruned. Van-Ginneken has a very effective pruning criteria in which both polynomiality and optimality were achieved [76]. Since the L and T values are not fixed, the Van-Ginneken pruning criteria cannot be used. We propose three probabilistic pruning strategies with varying complexities in the next few sub-sections. After pruning, we are left with a reduced set of solutions at node i. A dynamic programming implementation computes the delay distribution of all potential solutions at the input of the gate driving the RC-Tree. The solution which has the highest likelihood of meeting the delay constraint

is chosen as the final solution of the buffering problem(using equation 3.8).

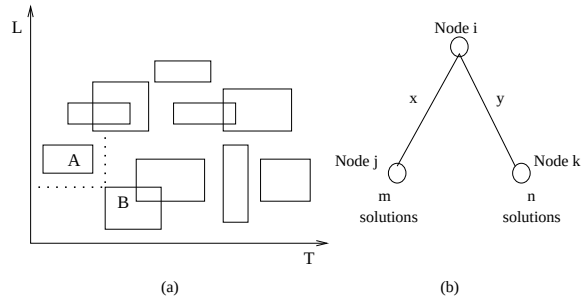


Figure 3.5: Distribution of Potential Solutions at a node

Pruning Strategies

We now propose three different pruning strategies for the Global Algorithm:

1. Criteria 1

   Let us assume that at a given sub-tree we have all the possible solutions (including the possibility of adding a buffer at the root of sub-tree). Let us make the following observation. A specific solution i at the root k of a subtree is actually a distribution in required time T_k^i and a distribution in capacitive loading L_k^i. Taking a worst case scenario, the possible points in this solution space could take the minimum value of the required time distribution (T_k^i) and minimum value of capacitive loading L_k^i or the maximum value of the required time distribution and maximum value of capacitive loading. These two scenarios define the boundary of our solution space for the distribution of

the potential solution. On a two dimensional plane with x-axis as T and y-axis as L, this solution can be represented as a rectangle in the worst case. The length of the rectangle is bounded by the smallest and largest required time values in the corresponding distribution. Similarly, the width of the rectangle is defined by the range of capacitive loading. For any point (x,y) inside the rectangle (x being the required time axis, y being the loading axis), p(x)p(y) denotes the probability of having x as the required time and y as the loading. Note that the probability that a solution point lies outside this rectangle is zero for the corresponding solution. Also note that if this was a deterministic approach, each solution would be represented as a point instead of a rectangle, which is exactly the case in the Van-Ginneken algorithm [76]. Figure 3.5(a) illustrates this concept and shows the distribution of the possible solutions at a node k. Consider two potential solutions A and B as marked in figure 3.5(a) for node k. It can be seen that B is better than A both in terms of the required time and capacitive load distributions. The dotted lines show that A does not even partially overlap with B along both T and L axis. Hence we can conclude that solution A is guaranteed to be worse than B and can be pruned out.

Formally the steps can be outlined as follows :

(a) Given a root k of a subtree with a set of possible solutions (T,L) represented as rectangles

(b) Prune out a rectangle which is definitely worse in capacitive loading and

required time than another solution

(c) The remaining set of solutions are considered co-optimal

Under this pruning strategy, the worst case number of solutions at the root can be exponential in the total possible buffer locations. The strategy also ensures that the optimal solution stays in the co-optimal set generated at the root.

2. Criteria 2

Once again, we assume that we have all possible solutions at a subtree k. In this approach, we have a stricter selection criteria which prunes out more potential solutions. We determine the probabilistic relation between a pair of potential solutions. Each pair (A,B) can have three possible relations: A can probabilistically prune out B, B can probabilistically prune out A or (A,B) could be co-optimal. These relations are elaborated as follows. First we would like to outline that given two distributions X and Y, probability that $X \geq Y$ is defined as $\sum_{y \in Y}(p(y) \sum_{x \in X, x \geq y} p(x))$. Given the distribution for T and L values for A and B, probabilities for the three possible relations is computed as follows.

$$A\, prunes\, B : P(A \Rightarrow B) = P_T(A \geq B) \cdot P_L(A \leq B) \tag{3.9}$$

$$B\,prunes\,A : P(B \Rightarrow A) = P_T(B \geq A) \cdot P_L(B \leq A) \tag{3.10}$$

$$Co - exist : P(A \sim B) = (1 - P(A \Rightarrow B) - P(B \Rightarrow A)) \tag{3.11}$$

Equation 3.9 computes the probability for A being better than B both w.r.t. required time ($P_T(A \geq B)$) and loading capacitance ($P_L(A \leq B)$). These values could be easily computed since we know the L and T distributions. Similarly equation 3.10 gives the probability of B being better than A. Equation 3.11 gives the probability of co-existence. The dominant relation between A and B is given by

$$max(P(A \Rightarrow B), P(B \Rightarrow A), P(A \sim B)) \tag{3.12}$$

Hence the relationship between two solutions A and B is decided by the one with highest probability. Now let us instantiate a graph G= (V,E) with each solution as node in G and edges defined as follows. If A prunes B according to equation 3.12 then there is a directed edge from node A to node B. If two nodes do not have any edges then they are co-existing solutions according to equation 3.12. Also note that there can be at-most one edge between any two nodes. We apply our pruning criteria on such a directed graph.

The main aim of pruning is to reduce the given set of solutions while maintaining the quality. The cost of a node δ_i is defined by the total number of nodes that can be pruned out by this solution. This essentially corresponds to the out-degree of this node. We want to find a maximum set of nodes such that:

(a) All nodes in the set are co-optimal/independent

(b) Each node not in this set has a directed edge from at-least one of the nodes in this set (note the keyword from)

(c) The cost of the set $\sum_i \delta_i$ is maximum

This is a variation of the maximum independent set problem [41]. The variation is that each node not in the independent set must have an incoming edge from at-least one node in the independent set. The logic behind this constraint is that if a node (or a solution) is culled out, then there must be at-least one solution in the independent set that prunes this solution out (according to the probabilistic criteria described in the equations above). Moreover, we want to maximize the total cost of all nodes, since the cost signifies the quality of a node (larger the cost, more are the number of solutions it prunes out). We name this problem as the DIRECTED MAXIMAL INDEPENDENT SET problem.

Theorem: DIRECTED MAXIMAL INDEPENDENT SET problem is NP-Complete.

Proof: Transformation from DIRECTED-COVER problem. Refer to subsection 3.1.5 for complete proof.

□

Heuristic for Criteria 2

Algorithm 1 *Heuristic for Criteria 2*

INPUT: Directed Graph G=(V,E);

Compute the cost of each node n = outdegree(n)

A: Set of all nodes

While(A != $\emptyset$)

Choose highest cost vertex i from set A and add i to set B

Remove all solutions from set A that have an edge with i

Return B

The heuristic used for this criteria is described in Algorithm 1. We sort all nodes w.r.t their cost values and iteratively pick the node with highest cost. The final solution generated by this algorithm will not satisfy the directed edge constraint of the Directed Maximal Independent Set problem. In fact the problem remains NP-C even if the directed edge constraint is relaxed since it is an instance of traditional Independent Set problem. Another point to note is that this is a stricter pruning criteria (prunes out more potential solutions) w.r.t. Criteria 1 but still does not guarantee a polynomial set of solutions at the root. Next we describe an even stricter approach which is completely polynomial.

3.

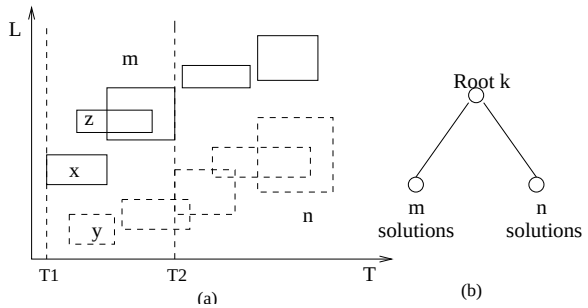


Figure 3.6: Generate Solutions at a Node from its Fanout Nodes

Criteria 3

This approach is focused on ensuring that the total number of solutions generated at the root of the wiring tree are polynomial in the possible buffer positions. Let us consider an internal node k with two subtrees (as shown in figure 3.6(b)). The two subtrees have m and n solutions respectively. In the worst case there will be $m \cdot n$ solutions at the root k after merging (assuming there is no buffer at the root). Merging two solutions from the left and the right subtrees essentially amounts to applying equations 3.6 and 3.7 to the distributions of the corresponding solutions. Let us suppose we are trying to merge solutions x and y in figure 3.6(a). The generated solution has a delay distribution starting from T1 (see figure 3.7(a)). Also whenever x is merged with any solution whose starting T value is greater than T1, the generated solution will start from T1. Since there can be at most $m+n$ distinct starting times in figure 3.6, the generated $m \cdot n$ solutions at root k will be clustered in at most $m + n$ starting times. This is illustrated in figure 3.7(a).

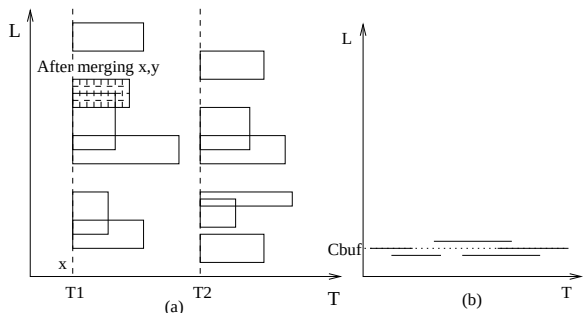


Figure 3.7: Total $m \cdot n$ Solutions After Merging

Now comes our pruning criteria. For each distinct starting time value (shown in figure 3.7(a)), we pick exactly one solution and prune the rest. Hence we have at most $m + n$ solutions at root k. The exact strategy of picking these solutions will be discussed in the subsequent paragraphs. Before that, let us consider the situation where we can add a buffer at root k. Hence for each of the $m \cdot n$ solutions, we have the choice of adding a buffer. Therefore the total number of solutions become $2 \cdot mn$. Note that for the buffered solutions the capacitance is no longer a distribution.This is illustrated in figure 3.7(b) where the capacitance is a fixed value. According to this pruning criteria, we pick exactly one of these $m \cdot n$ buffered solutions at root k. The selection criteria used is as follows. We compute the probability that a buffered solution is better than another using equation 3.9. For each solution, we get the cumulative probability values. Finally we choose the solution that has the largest value. This solution is probabilistically better than all the other buffered solutions. Hence the total number of solutions that we store at a subtree is O(m+n+1) (m+n for solution with no buffers and one with the buffer). This is a poly-

nomial quantity. Even the Van-Ginneken algorithm [76] was storing at most $m + n + 1$ solutions at a subtree. This results in a polynomial number of solutions at the root of the wiring tree. The proof of polynomiality is the same as the proof given in [76].

Theorem: Criteria 3 is polynomial in the problem size

Proof: Proof given in [76].

□

Now we go into the details of how $m+n$ solutions are chosen from $m \cdot n$ possible solutions (for the case where there are no buffers). As mentioned before there will be $m+n$ distinct starting T values (figure 3.7(a)). We choose exactly one solution from each distinct starting T value. This problem is modeled using *COMPLETE R-PARTITE MAX COST CLIQUE* problem [95]. The transformation is as follows. We instantiate a complete R-PARTITE graph G=(V,E) with the following properties. For each set of solutions that have the same starting T value, we instantiate a graph partition with a node for each solution. These nodes do not have edges between them. There are undirected edges between all other pair of nodes from different partitions. This generates a complete R-PARTITE graph with $m + n$ as R. The generated graph is shown in figure 3.8. Each edge has a cost which signifies the probability that the corresponding solutions are co-optimal. This can be computed using

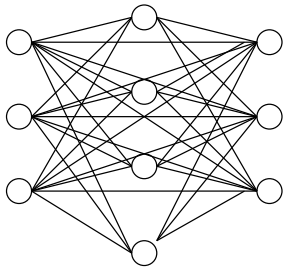


Figure 3.8: Complete R-Partite Max Cost Clique

equation 3.11. The problem is to find a clique in this graph with maximum cost. Note that the largest clique in this graph can be trivially generated by picking a node from each levels. The challenge is to generate the clique with maximum cost. Picking one node from each partition would ensure one solution for each starting T value is chosen. Hence the total number of chosen solutions are $R = m + n$. Larger the cost of the clique, higher the probability of co-optimality of all solutions. Hence a larger solution space could be represented by the same number of solutions.

Theorem: COMPLETE R-PARTITE MAX COST CLIQUE is NP-Complete

Proof: Transformation from 3SAT [41]. Refer to subsection 3.1.5 for complete proof.

□

Heuristic for Criteria 3

Algorithm 2 *Heuristic for Criteria 3*

INPUT: set of mn solution partitioned in $m+n$ distinct sets

Get the probability of co-existence between each solution pair i and j

Sort the $m+n$ sets depending on their starting T values

Loop(over all $m+n$ set)

For(each solution x in set i)

For(each solution y in set $i-1$)

Calculate the cost of the clique formed by adding x to the best clique at y

Choose the clique with maximum cost and store it. This is the cost of the best clique at x now

At the last set, among all x in this set, pick the clique with largest cost

We propose a heuristic (refer Algorithm 2) for the above problem formulation. We are given $m+n$ sets of solutions sorted in increasing order of T. We traverse these sorted set one by one and at each set we generate the possible cliques for each potential node in this current set by merging it with each of the best cliques generated at the previous set. From these potential solutions, the clique that gives the maximum cost is stored. When the next set is traversed, the possible solutions at the next set will use this best clique information. The is done iteratively till the last set is encountered. At this stage the clique with best cost is selected.

This completes the description of the three pruning criteria. Each of them has its own distinct property. **Criteria 1** ensures that the best solution will never be pruned out, but is not polynomial. **Criteria 2** has a methodology which enables more

Bench # Sinks	Van-Gin Sol (nano sec)	T_{cons} (nano sec)	Van-Gin P_e	# Bufs	Crit 1 P_e	# Bufs	Crit 2 P_e	# Bufs	Crit 3 P_e	# Bufs
54	121.2	122	0.198	14	0	13	0	15	0	13
96	102.4	103	0.563	19	0	19	0	19	0	20
216	585.9	586	0.467	23	-	-	0	23	0	20
360	117.7	118	0.917	70	-	-	0	73	0	72
468	582.6	583	0.531	45	-	-	0.025	41	0.213	32
590	390.8	391	0.618	77	-	-	0	80	0	79
720	722.3	723	0.596	61	-	-	0	63	0	59
890	845.5	846	0.573	60	-	-	0	59	0	57
1080	1598.3	1599	0.414	78	-	-	0	84	0	69
1260	5812.2	5813	0.639	95	-	-	0	93	0	94
Avg			0.552	-			0.003		0.021	

Table 3.1: Results from Experiments

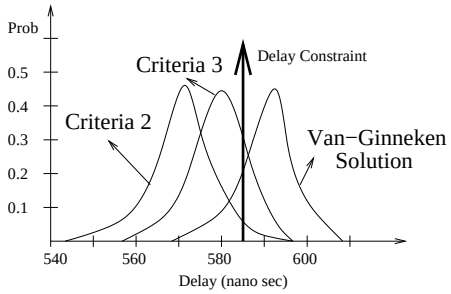


Figure 3.9: Comparison of Solutions for a Benchmark

pruning and hence is faster than **Criteria 1** but is still not provably polynomial. **Criteria 3** is strictly polynomial and has a very firm selection mechanism.

3.1.4 Experimental Results

The objective through experimental results was to illustrate the superiority of our approach over fixed wire-length assumption and also compare the quality of the three pruning criteria. For experimental purposes, we used large wiring trees with large number of sinks. Some of these trees were balanced and some were unbalanced

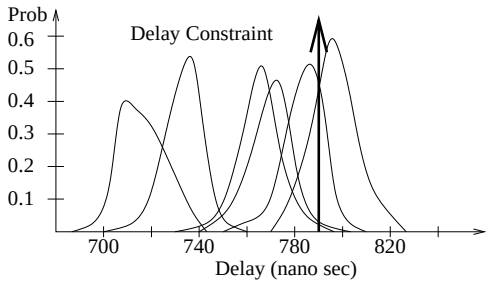


Figure 3.10: Delay Distribution of Solutions Satisfying a Delay Constraint

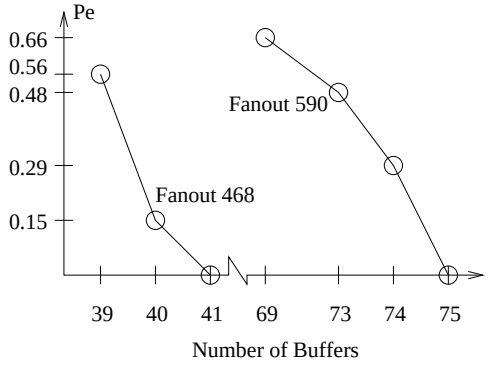


Figure 3.11: Trade-off Between Number of Buffers and Probability of Error

(close to lt-trees). The required arrival times at the sinks of the benchmarks were also chosen randomly. Values for r and c (wire parasitics) were chosen for 0.18 micron technology. Wire-lengths were taken as Gaussian distributions with mean varying between 100 to 1000 λ and variance lying within 10% of the mean. The Van-Ginneken algorithm was used to generate a valid buffer placement with largest required time at source. The delay constraint for the tree was set at a value slightly greater than the single delay value given by the Van-Ginneken Algorithm. This ensures that the Van-Ginneken estimate always satisfies the constraint.

The heuristics were implemented in SIS [37]. The wire-length distributions were taken to be discrete for simplicity of implementation. Hence, the equations mentioned in section 3.1.1 were applied to discrete distributions to get the corresponding distributions for delay and capacitive loading which were also discrete. We again point out that this implementation is generic and can work with any distribution of wire-lengths.

Table 3.1 illustrates the performance of our criteria as compared with fixed wire-length estimate buffer insertion [76]. The average wire-length of the distribution was provided as wire length estimate input to the Van-Ginneken algorithm. We chose a solution from the Van-Ginneken algorithm that satisfied the delay constraint and then generated the delay distribution for the solution. The second column of table 3.1 provides the output of the Van-Ginneken algorithm. It can clearly be seen from the results that fixed value estimates result in delay constraint violation with very high probabilities (P_e). Given a delay constraint P_e is computed using $P_e = \sum_{d \geq D_{cons}} p(d)$

On the other hand probabilistic buffer insertion results in the delay constraint being satisfied with a very high probability. Even though the total number of buffers between our approach and the Van-Ginneken approach remain more or less the same, it was observed that the buffer locations are different. Another observation that we made was that Criteria 1, which has a very relaxed pruning strategy was not practical for larger benchmarks since it did not give results in a reasonable run time. Results also show that Criteria 2 solutions have the least probability (P_e) of not satisfying the delay constraint without any major buffer penalty. We also observed that for most cases, the actual delay distribution found using our criteria were much better than those from the Van-Ginneken fixed value algorithm. This is illustrated in figure 3.9. Clearly, the best solution of the Van-Ginneken approach is worse than that generated by Criteria 2. There is an important inference here: **optimization using fixed average wire-length estimates does not generate a distribution with the smallest average delay**. This is a critical observation since optimization using average wire-length values does not optimize average delay at the root.

Figure 3.10 shows a typical delay distribution solution at the root using the probabilistic approach. It is evident that there are several possible solutions for a given delay constraint (shown by the arrow in figure 3.10) that have differing probabilities of satisfying the constraint. We observed from the experiments that a trade-off exists between probabilistically satisfying a delay constraint and the number of buffers used by that solution. Figure 3.11 shows such a trade-off for two benchmarks. This gives the designer the flexibility of taking a $Probabilistic - Risk$

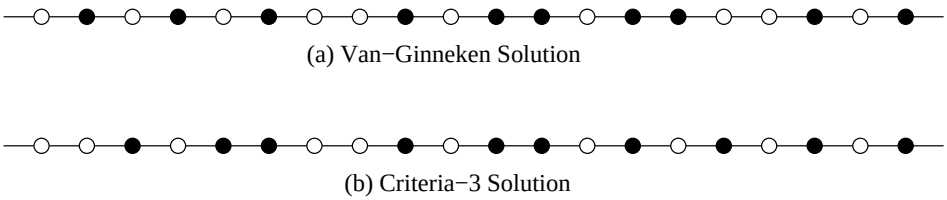


Figure 3.12: Buffered Solution for a 2 Terminal Net with 20 Potential Locations

of not satisfying the delay constraint while choosing a solution with fewer buffers.

Table 3.2 shows that the run time comparison between the three Criteria proposed in section 3.1.3. All values have been normalized w.r.t. the run time values for Criteria 2. Criteria 1 is not polynomial in complexity and hence has a very large run time for reasonably large benchmarks. It has a much weaker pruning criteria, but the quality of final solution could be very high since it retains all possible solutions which can potentially be better. But the runtime for Criteria 1 was unreasonably large making it impractical when compared with Criteria 2 and 3. As can be seen from table 3.2, run time for Criteria 2 is similar to that of Criteria 3 even though it is not polynomial in the worst case. This illustrates that Criteria 2 has a very good performance both in terms of run time and quality of the final solution. Run-time is a drawback in statistical approaches as compared with deterministic approaches since there is more computation that needs to be performed. We have implemented a generic probabilistic algorithm which does not assume anything about the underlying distributions. However, we can make intelligent assumptions about the distributions and use approximations to speed-up the probabilistic algorithm. Such an approach has been shown to work efficiently in case of Statistical Timing in [10, 27, 2] and is a direction of future work for probabilistic buffer insertion.

Bench Sinks	Criteria 1	Criteria 2	Criteria 3
54	2.39	1	1.02
96	3.75	1	1.03
216	-	1	0.90
360	-	1	1.57
468	-	1	1.05
590	-	1	0.97
720	-	1	0.92
890	-	1	0.95
1080	-	1	0.91
1260	-	1	1.12

Table 3.2: Runtime Comparison Between the Three Criteria

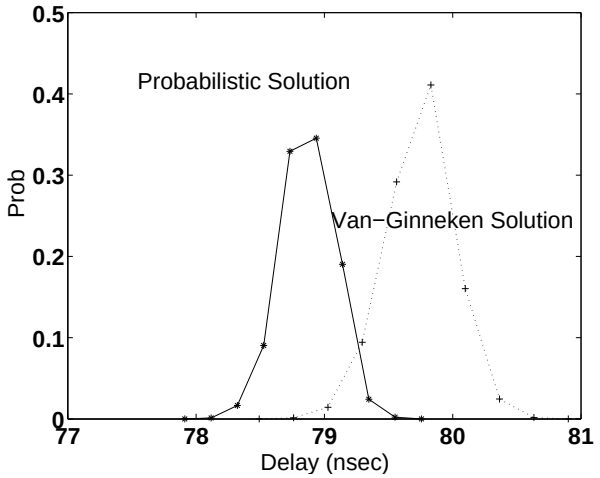


Figure 3.13: Delay Distribution of Buffered Solutions for a 2 Terminal Net with 20 Potential Locations

Bench mark	#Buf. Pos.	V-Gin. #Buf	V-Gin. Delay Range(ns)	Crit-3 #Buf	Crit-3 Delay Range(ns)
Net 1	20	10	[78.5, 80.9]	10	[77.9, 79.7]
Net 2	40	19	[143.2, 145.2]	20	[142.0, 143.2]
Net 3	60	28	[213.3, 215.8]	31	[211.2, 213.1]

Table 3.3: Result for 2 Terminal Nets

We have conducted another set of experiments that demonstrate the difference between our probabilistic scheme and deterministic Van-Ginneken algorithm. We assume that we have long 2 terminal nets with potential buffer locations. The wire-segment between each pair of potential locations is estimated to be Gaussian distributions. We generate a best-delay buffered solution for the net using the Van-Ginneken Algorithm with mean value estimates for each wire-segment and calculate its actual delay distribution using the wire-length distribution. We also apply probabilistic buffer insertion using Criteria 3 and generate a best-delay buffered solution. This experiment has been conducted for three 2 terminal nets with with different lengths having 20 (Net 1), 40 (Net 2) and 60 (Net 3) potential buffer locations respectively. Figure 3.12 shows the results for the buffered net with 20 potential locations. We can see that there is a difference in the final buffered solution and the delay distributions of the two solutions are given in figure 3.13. The probabilistic approach is able to consider the variability in wire-lengths during optimization and makes better choices about potential buffer locations. The results are shown in table 3.3. We can see that the probabilistic buffer insertion results in better delay distributions at almost no buffer penalty.

In order to validate our probabilistic buffer insertion algorithms, we did some experiments on actual nets after placement and compared the delay of the buffered net using post-routing wire-length values. These results show the effectiveness of our proposed algorithms on realistic circuits. We took multi-terminal nets and generated a valid placement for the net using an academic placement tool (Parquet [94]). We generated bounding box estimates for each wire-length segment in the net and used this as the deterministic fixed value estimate. The probabilistic method assumes a net-length distribution based on the proposed model in [9]. We then used an academic routing tool (Labyrinth [85]) to route this net and generated the actual post routing wire-length values for each segment of the net. The delay for the buffered net solutions from the deterministic as well as probabilistic algorithm was computed using these post routing wire-length values for each segment of the net. Table 3.4 illustrates results of the probabilistic buffer insertion for some benchmarks. We compare the actual delay after routing from the traditional deterministic buffer insertion approach to the proposed probabilistic algorithm. It can be seen that actual post-routing delays in the probabilistic case is smaller than the deterministic approach, hence the probabilistic method has higher chance of design closure. We note that the benchmarks on which the data is reported the real post-routing wire-lengths on the critical path was around twice larger than bounding box, which is why the probabilistic approach was better since it used a distribution as the wire-length estimate and captured this wire-length estimation uncertainty. The bounding box of the other hand is a poor estimate and leads to solutions using the deterministic algorithm which have higher delays than that given by probabilistic buffer insertion.

Bench mark	# Sinks in Net	Det. Delay (ns)	Prob. Delay (ns)
Net 1	94	59.9	25.4
Net 2	104	25.2	14.4
Net 3	137	61.7	37.8
Net 4	205	92.3	53.3

Table 3.4: Post Routing Delay Results: Deterministic vs Probabilistic

In general from our experiments we found that bounding box was a pretty bad estimate for 60-70% of the nets, which means that in such situations probabilistic buffer insertion would perform really well.

These results clearly illustrate the effectiveness of probabilistic buffer insertion over the traditional deterministic Van-Ginneken approach under uncertainties in optimization parameters.

3.1.5 Appendix

Directed Maximal Independent Set

We will use the transformation from the Directed-Cover problem to the Directed Maximal Independent Set problem to prove the NP-Completeness of Directed Maximal Independent Set problem. In this proof, we are considering the case of Directed Maximal Independent set for which each node weight is 1. This assumption only means that a Directed Maximal Independent Set solution also implies a solution with maximum node cost. We point out here that this assumption does not

weaken the proof of NP-Completeness that follows. We first need to prove that the Directed-Cover problem is NP-Complete.

Directed Cover

Decision Problem : Given a directed graph G=(V E) and a number K, Is there a vertex cover $V^{'}$ of size K or less such that all edges are covered and there is at least one directed edge from a vertex in $V - V^{'}$ to each of the vertices in $V^{'}$. Note that decision problem does not encompass the cost aspect of our formulation. If this problem is NP-C then the generalized problem which considers cost is also NP-Complete.

Theorem: Directed-Cover is NP-Complete

Proof: We observe that this problem is in NP since given a directed cover solution we can easily verify its feasibility. Next we transform the general instance of UNIPHASE-ONE-IN-3SAT [41] problem to an instance of the Directed-Cover Problem. The UNIPHASE-ONE-IN-3SAT problem is defined as follows:

Set U of variables, collection C of clauses over U such that each clause has exactly 3 variables. Moreover none of the clauses contain a negated literal (uniphase aspect of the problem). Is there a truth assignment to the variables such that all clauses are satisfied and exactly one literal of each clause evaluates to TRUE.

UNIPHASE-ONE-IN-3SAT is NP-Complete[41]

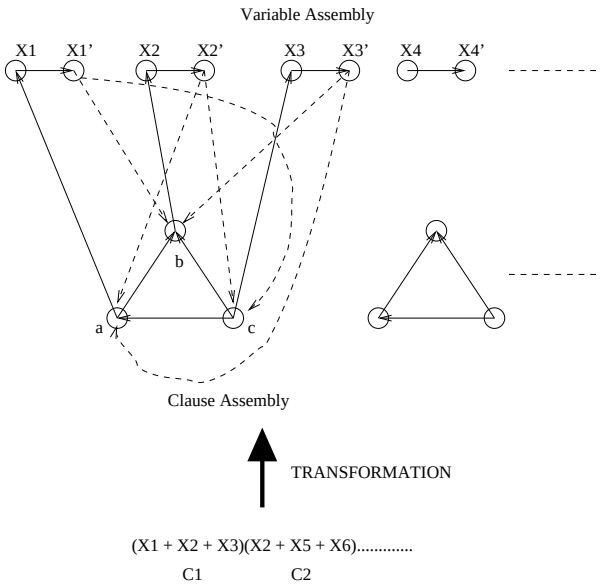


Figure 3.14: Transforming UNIPHASE-ONE-IN-3SAT to Directed-Cover

Figure 3.14 illustrates the transformation. Given a set of C clauses and U variables, we define two nodes for each variable, each corresponding to its positive and negative phase. We assign a directed edge from the positive phase node to the negative phase node. This is called the variable assembly. For each clause we assign three nodes, one corresponding to each literal. The directed edges between these nodes can be arbitrarily assigned. Each node on this triangle corresponds to a specific variable. Now we add a directed edge from this node to the corresponding variable node. Since all variables occur in positive phase in clauses, these edges get connected to the positive phase node. Edges $aX1$, $bX2$, $cX3$ represent such edges. Now for each negative phase node $(X1^{'}, X2^{'})$, we add directed edges from these nodes to those nodes in the clause assembly which do not correspond to this variable. The hashed directed lines $X1^{'}b, X1^{'}c$ represent such edges. This transformation is repeated for all the clauses. Hence we will have 3C+2U nodes in the final graph.

Note that this is a polynomial time transformation. Let us assign K=U+2C. Hence we need to find at most U+2C vertices which cover all the edges and for all covered nodes, there is at-least one incoming edge for a node not covered. Note that in this transformation we must pick exactly one variable node and exactly two clause nodes.

We prove that UNIPHASE-ONE-IN-3SAT problem can be solved iff we can solve this instance of Directed-Cover.

If Part: Given a solution for UNIPHASE-ONE-IN-3SAT, we transform it to a solution of Directed-Cover. For all variables that are assigned a truth value of TRUE, pick the corresponding node in variable assembly that signifies TRUE. If the variable is assigned FALSE pick the other node. We know that for each clause, exactly one variable has true. Hence in the clause triangle we pick two nodes whose corresponding literals evaluate to false (note that each node in the clause triangle corresponds to one of the clause literals). This always generates a vertex set of size K that covers all the edges. This also ensures that each of the selected vertex has at least one incoming edge from a vertex not selected. Hence, this is a valid solution for Directed-Cover.

Only If Part: Given a solution for the Directed Cover Problem, we will generate a valid variable truth assignment. We show that for each clause triangle, exactly one positive variable node is chosen. Let us suppose that this does not happen. In that case the solution has either all variable nodes with negative phase (for a specific

clause), or the clause triangle has more than one variable nodes with positive phase. In the former case, clearly all edges will not be covered since edges like $aX1$ will be left out. If more than one positive phase variable node of a clause is chosen then there will be edges connected to the negative phase nodes that will not be covered. Hence a valid Directed-Cover solution must pick exactly one positive phase variable node for each clause. This would satisfy the UNIPHASE-ONE-IN-3SAT.

Thus we have shown that the Directed-Cover problem is also NP-Hard. **Hence, Directed-Cover is NP-Complete**

Directed Maximal Independent Set

Decision Problem : Given a directed graph G=(V E) and a number P, Is there an independent set of vertices $V^{'}$ of size at least P such that every vertex in $V - V^{'}$ has at least one directed edge coming in from some vertex in $V^{'}$.

Theorem: Directed Maximal Independent Set is NP-Complete

Proof: Given a solution to the Directed Maximal Independent Set, it can easily be verified in polynomial time. Hence, Directed Maximal Independent Set is in NP. Next we prove that Directed Maximal Independent Set problem is also NP-Hard (and hence NP-Complete) We transform a general instance of the Directed-Cover problem (proved NP-Complete earlier) to an instance of the Directed Maximal

Independent Set problem. The Directed Cover problem can be defined as follows:

Given a directed graph G=(V E) and a number K, Is there a vertex cover V' of size K or less such that all edges are covered and there is at least one directed edge from a vertex in $V - V'$ to each of the vertices in V'.

Directed-Cover is NP-Complete (proof given earlier in this section).

The transformation from Directed-Cover to Directed Maximal Independent Set is simple. Given an instance of the Directed-Cover problem with a graph G=(V,E) and a number K such that the size of the Directed-Cover is atmost K, we take the same graph G=(V,E) as an instance of the Directed Maximal Independent Set Problem and define P to be $(|V| - K)$. This is a polynomial time transformation.

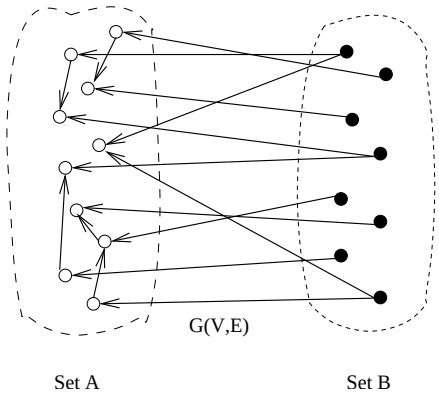


Figure 3.15: Transforming Directed-Cover to Directed Maximal Independent Set

We prove that Directed-Cover problem can be solved iff we can solve this instance of Directed Maximal Independent Set problem.

If Part: Given a solution to the Directed-Cover problem, we will transform it to a solution of Directed Maximal Independent Set problem. A valid solution to the

Directed-Cover problem implies that we are given a Set A of V' vertices as shown in figure 3.15 such that there are n ($n \leq K$) vertices in this set and they form a vertex cover. Additionally, there is atleast one incoming edge to each of these n vertices from some vertex in the remaining set of $V - V'$ vertices represented by Set B in the figure 3.15. We note here that since Set A forms a vertex cover, there are no edges between any two vertices from the Set B. By our transformation the target number P of the Directed Maximal Independent Set problem is given as $P = (|V| - K)$. Now since there are no edges between the vertices in Set B, they are independent by definition and are $P' = |V| - n$ in number. Also, a valid solution of Directed-Cover implies that there is atleast one outgoing edge from some node in Set B to every node in Set A. We note here that since $n \leq K$, $-n > -K$ which implies that $|V| - n > |V| - K$ or $P' > |V| - K$. Hence, we prove that $P' > P$. Therefore set B forms a valid solution of the Directed Maximal Independent Set problem with $P' = (|V| - n)$ vertices in the set.

Only If Part: Given a solution to the Directed Maximal Independent Set problem, we will transform it to a solution of the Directed Cover problem. We are given P' ($P' > P$) independent vertices (Set B) such that there is a directed edge from one of these vertices to every vertex in the set $|V| - P'$ (Set A). Since Set B is an independent set, there are no edges between any two vertices from this set. This means that all the edges of the graph G=(V,E) are either between two vertices in Set A or between one vertex in Set A and one in Set B. This implies that Set A is a valid vertex cover of size $n = (|V| - P')$. Additionally, every vertex in Set A has an incoming edge from some vertex in Set B. Now since $P' > P$,

$n = (|V| - P') < (|V| - P)$. Also, we know that $P = |V| - K$. Combining, we can see that $n < K$. Therefore, Set A is a valid solution for the Directed-Cover problem.

Thus, we have shown that Directed Maximal Independent Set problem is also NP-Hard. **Hence, Directed Maximal Independent Set is NP-Complete**

Complete R-Partite Max Cost Clique

Decision Problem : Given a complete R-Partite graph G=(V,E) with weights on edges w(e) $\in [0, 1]$ and a target weight number B, is there a clique of size R with total edge weight greater than equal to B in G(V,E).

Theorem: Complete R-Partite Max Cost Clique is NP-Complete

Proof: Given a solution to the Max Cost Clique problem, its validity can easily be verified in polynomial time. Hence, Max Cost Clique is in NP. Next we prove that Max Cost Clique problem is also NP-Hard (and hence NP-complete). We transform a general instance of the 3SAT[41] problem to an instance of the Complete R-Partite Max Cost Clique problem. The 3SAT problem can be defined as follow:

Set U of variables, collection C of clauses over U such that each clause has exactly 3 literals. Is there a truth assignment to the variables such that all the clauses are satisfied.

3SAT is NP-Complete[41]

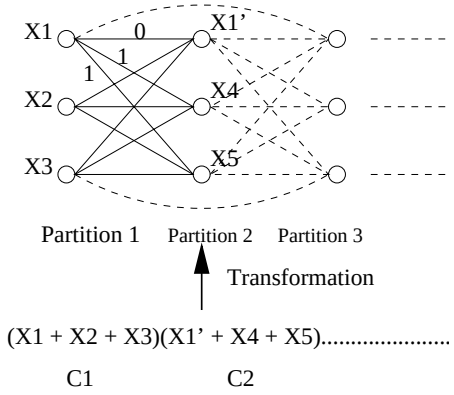


Figure 3.16: Transforming 3SAT to Complete R-Partite Max Cost Clique

The transformation from 3SAT to Complete R-Partite Max Cost Clique is illustrated in figure 3.16. Given an instance of the 3SAT problem on m clauses in the collection C and U variables, we will construct an m partite (R = m) graph G=(V,E) as follows. For each clause in the 3SAT problem, we create a partition in graph G with exactly three vertices each corresponding to one literal in that clause. We put an edge between every pair of vertices from different partitions. We assign an edge weight 0 to an edge if the literals corresponding to those vertices are negations of each other, otherwise we assign an edge weight of 1. For example, if the variables are X1,X2, etc. and u,v are vertices from different partitions, w(u,v) = 0 for (u = X1 and v = $\bar{X}$1), (u = X2 and v = $\bar{X}$2), etc. and w(u,v) = 1 otherwise (refer to figure 3.16). There are no edges between vertices from the same partition. Note that this graph G is a complete m partite graph and the size of the maximum clique is m. Let us assume the target weight number B = m(m-1)/2. Note that this is a polynomial time transformation.

We prove that 3SAT problem can be solved iff we can solve this instance of

Complete R-Partite Max Cost Clique problem.

If Part: Given a solution to the 3SAT problem, we will transform it to a solution of Complete R-Partite Max Cost Clique problem. A satisfying truth assignment to the 3SAT problem implies that there is atleast one true literal in each clause. We pick one vertex from each of the m partitions that corresponds to a true literal in the corresponding clause from the 3SAT solution. Hence, we can construct a clique of size m, whose edges have weight = 1. Summing up the edge weights we get a clique of total cost equal to m(m-1)/2. Therefore, this is a valid solution for the Complete R-Partite Max Cost Clique problem.

Only If Part: Given a solution to the Complete R-Partite Max Cost Clique problem, we will generate a valid satisfying truth assignment to the 3SAT problem. Given a clique of size m with total edge weights equal to m(m-1)/2, we know that each of the edge weights in the clique must be 1. By assigning the value true to the literals corresponding to the vertices of this clique, we can generate a valid truth assignment for each clause since there is exactly one true literal in each clause. There can be no contradictions since for any variable X, both X and $\bar{X}$ cannot be in the clique because the edge (X,$\bar{X}$) has weight 0 and we have selected edges with weights 1 only. Therefore, this is a valid solution for the 3SAT problem.

Thus, we have shown that Complete R-Partite Max Cost Clique problem is also NP-Hard. **Hence, Complete R-Partite Max Cost Clique is NP-Complete**

3.2 Monte-Carlo Driven Stochastic Optimization Framework for Handling Fabrication Variability

Increasing impact of fabrication variability in deep sub-micron technology has become a potent problem in VLSI Design optimization. Variability causes circuit parameters to behave as random variables making traditional deterministic techniques for design optimization sub-optimal. This has inspired a growing interest in investigating statistical techniques for design optimization. Circuit performance (timing, power etc.) has a non-linear dependency on global parameters that are affected by fabrication variability. This makes the problem of handling the variability effects even more difficult.

Several recent works have focused on modeling global circuit parameters (t_{ox}, L_{eff} etc.) as random variables and then introduced techniques to perform analysis and optimization. In order to make the problem computationally efficient, a lot of work assumed that these parameters either have a Gaussian distribution or have a linear dependence on circuit performance or both. While these approximations can be justified to some extent when the magnitude of these parameter variations is small. But with each technology node, the variations are becoming more dominant. Hence we need to develop techniques that do not make such assumptions and have the flexibility to deal with non-linear relationship of circuit parameters with performance under any arbitrary distribution of parameters.

A lot of recent work in fabrication variability aware analysis and optimization has been focused along these directions. Statistical timing analysis (STA) is one such

area that has gained a lot of attention within the research community. The initial works on STA made assumptions that circuit timing was a linear function of Gaussian distributed parameters [26, 42, 10, 118, 60, 39, 79]. More recently, several STA techniques have been proposed that extended this paradigms to consider non-linear and non-Gaussian parameter dependence on timing [127, 30, 43, 119]. Furthermore, a lot of research has been focused on developing statistical optimization techniques for yield improvements [8, 4, 77, 80, 67, 32, 73]. Some of these are based on worst-case assumptions [67], some are sensitivity-driven approaches [4, 32, 73, 80], while others are mathematical approaches [77]. These approaches either do not explicitly converge to the optimal solution, or make assumptions about the nature of the distributions to converge to the optimal solution [67]. The sensitivity based approaches are heuristic in nature and do not converge to the optimal solution. The work presented in [8] presents a stochastic programming based framework for variability-driven gate sizing is an interesting example of stochastic optimization based frameworks.

In the most general case, it is very hard to develop analytical formulations that are able to consider the true non-linear nature of parameter dependence on performance as well an arbitrary distribution of the parameters. As a result either approximations need to be built into the techniques that tend to make results erroneous, or some kind of numerical techniques need to be considered [43, 119]. The only real way to capture variability in analysis and optimization is through Monte-Carlo based techniques. But as is well known, this approach tends to be extremely inefficient making it difficult to be incorporated into classical optimization

frameworks. Hence, there is a very strong motivation to develop Monte-Carlo based statistical frameworks for analysis and optimization that are efficient and would provide accurate results. In this work we propose to develop one such framework. We will look at the classic linear programming framework and show how we can model a deterministic linear programming formulation to be variability-aware and propose a *Stochastic Programming* based framework to solve the problem. The stochastic programming framework intrinsically uses Monte-Carlo based simulations in an intelligent way to make the overall scheme efficient and accurate. Such an approach is important as it provides a mathematical framework for variability driven optimization with convergence to the optimal solution without making any assumption about the nature of variability.

In this work, we investigate the classic linear programming formulation as applied to an important VLSI CAD problem of MTCMOS sizing [116] from a stochastic perspective. We will show how this deterministic linear programming formulation that is incapable of handling the fabrication variability in circuit parameters (like threshold voltage V_t, transistor length L_{eff} etc.) can be extended as a stochastic programming formulation. We present a formal methodology that enables modeling and optimizing linear programming formulations in which the parameters are random variables. These parameters essentially correspond to unpredictable circuit parameters due to fabrication variability. This methodology, formally known as stochastic programming [66, 21, 121, 93, 57, 55], provides a very strong framework for modeling and optimization of linear programming based VLSI-CAD problems under variability. We will investigate/review the theory of stochastic programming

and present some techniques for optimization of the same. Specifically, we present the theory of two stage stochastic programs with recourse and review two techniques for optimizing them: Successive Sample Mean Optimization SSMO and Stochastic Decomposition.

In this work, we will define our optimization objective as *Binning Yield Loss (BYL)*. In high performance designs, process variations result in a spread in the achievable frequency, thereby causing some chips to fail from meeting the nominal target frequency. In [98], the authors have mentioned that as much as 30% frequency variation can be observed in high-performance designs. Chips can be *binned* according to their operating frequency. Those that fail to meet the target frequency can either be sold at a loss or be discarded. In [13], the authors present a hardware design to perform speed binning in microprocessor design. Each speed bin has a corresponding penalty cost that is proportional to its slowdown from the target frequency. Thus, there exists a binning-yield loss with each design depending on the spread in its operating frequency due to process variations. In this work, we use BYL as an optimization objective in our formulation.

The MTCMOS sizing problem for BYL optimization can be formally defined as:

Given a gate-level circuit with fine-grained sleep transistors placed at the gates, the arrival time at each primary input and a required time constraint at each of the primary outputs, optimally size the sleep transistors such that the nominal leakage and BYL is minimized.

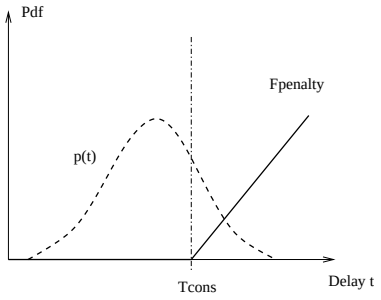


Figure 3.17: Binning Yield Loss with a Linear Penalty Function

Experimental results on the MCNC benchmarks indicate that the deterministic linear programming formulation (that does not consider variability) generates solutions for MTCMOS sizing that violate the timing constraints with a 48% probability. The stochastic decomposition method on the other hand generates solutions that satisfy the timing constraints with a 100% probability (No BYL). The SSMO technique is very slow (reasons analyzed later) and did not converge to a solution for most benchmarks. The experimental results clearly point out the superiority of performing stochastic programming for optimizing VLSI CAD problems in presence of variability as compared with deterministic linear programming. This work introduces the formal optimization framework behind the modeling and optimization of VLSI CAD problems in the stochastic programming perspective.

3.2.1 Binning Yield Loss

In high performance designs, process variations result in a spread in the achievable frequency, thereby causing some chips to fail from meeting the nominal target frequency. In [98], the authors have mentioned that as much as 30% frequency variation can be observed in high-performance designs. Chips can be *binned* according

to their operating frequency. Those that fail to meet the target frequency can either be sold at a loss or be discarded. In [13], the authors present a hardware design to perform speed binning in microprocessor design. The penalty that the chips in a speed bin have to incur is proportional to the slowdown from the target timing constraint (T^{cons}). Let us suppose that the timing delay of the chip is t. We define a BYL penalty function $F_{penalty}(t)$ as follows:

$$F_{penalty}(t) = \begin{cases} t - T^{cons}; & t \geq T^{cons} \\ 0; & otherwise \end{cases} \tag{3.13}$$

Let us suppose the probability density function (pdf) of circuit delay is $p(t)$ as shown in figure 4.1. For this scenario, we can define the BYL for the design as:

$$BYL = \int_{-\infty}^{\infty} F_{penalty}(t)p(t)dt = \int_{T^{cons}}^{\infty}(t - T^{cons})p(t)dt \tag{3.14}$$

In the optimization framework proposed in this work, we will use the above definition for BYL.

Often the optimization objective in variability-driven problem formulations is taken to be the traditional *Timing Yield Loss (YL)*, which is the probability that a design violates the timing constraints.

$$YL = \int_{T^{cons}}^{\infty} p(t)dt \tag{3.15}$$

It is easy to note that even though YL and BYL do not represent the same objective, but there is correlation between the two. As we shall report in our results, even though we optimize for BYL, the YL numbers obtained from our proposed stochastic programming framework are significantly better as compared to the de-

terministic sizing formulations. It is also interesting to note that if there is a solution for which BYL is 0, then this also implies that the YL will be 0 for that design.

3.2.2 Motivational Example: Linear-Programming Based Optimization

In this section, we will consider a general VLSI-CAD optimization formulation that has timing constraints on the design while trying to minimize a certain cost function. From the previous section, we understand that BYL is directly proportional to the timing violation in the design and we will try to demonstrate this through the following formulation.

Let us suppose that we are given a gate-level circuit where the delay of each gate i is denoted by d_i. If we assume a linear gate-delay model, we can define d_i as:

$$d_i(x_i) = p_i + q_i * x_i \quad \forall gate \;\; i \tag{3.16}$$

where p_i and q_i are constants for each gate and x_i is the control variable (say transistor size for example). At this point we can note that if we have a non-linear gate delay model (say convex), we can linearize it using piecewise approximations. The resulting linear-gate delay model can be made sufficiently accurate by controlling the number of piecewise linearizations used.

We can now try to understand a general linear-programming based optimization formulation that using the above linear gate delay model. Let us suppose that we have to assign the control variables ($\vec{x}$) values such that the timing constraints

at the primary output of the circuit are satisfied and an objective cost is minimized. The arrival time at the output of each gate i can be denoted by a variable D_i. At each primary output (PO), we impose a timing constraint T^{cons}. The objective of minimization in general can be a linear function of the control variable $\vec{x}$. We can write this formulation mathematically as:

$$Minimize \ \ (\sum_i c_i * x_i)$$

$$Subject \ \ to: \begin{cases} d_i(x_i) - D_i + D_j \leq 0 & \forall j \in \ Fanin(gate - i) \\ d_i(x_i) = p_i + q_i * x_i \ \ \forall gate \ \ i \\ D_i <= T^{cons} \ \ \forall gate \ \ i \in PO \\ x_i^{min} \leq x_i \leq x_i^{max} \end{cases} \tag{3.17}$$

where c_i are constants and (x_i^{min}, x_i^{max}) denoted the valid range for each control variable x_i.

Effect of Fabrication Variability

Fabrication variability causes several circuit parameters that are assumed to be constants in the previous formulation to have random behavior. Circuit parameters like chip threshold voltage V_t, effective channel length L_{eff} etc. have been shown to have such randomness due to fabrication variability [98]. Other circuit parameters like chip temperature, supply voltage etc. also have randomness due to environmental uncertainties. Randomness in circuit parameters will cause the constants (p_i, q_i) in equation 3.16 to become uncertain, which means that the effective gate delay d_i is now not only a function of the control variable x_i but also the underlying randomness field $\vec{\Omega}$. These random variables will show correlated behavior since

they are inspired by variability in basic circuit parameters. This correlation can be easily considered in the optimization process by appropriately sampling the random field. This motivates us to formulate optimization problems like the previous ones from a stochastic perspective.

The linear gate delay d_i can now be represented as:

$$d_i(x_i, \vec{\Omega}) = p_i(\vec{\Omega}) + q_i(\vec{\Omega}) * x_i \quad \forall gate \ \ i \tag{3.18}$$

Hence the linear-programming based formulation presented earlier in this section can now be written as:

$$Minimize \ \ (\sum_i c_i(\vec{\Omega}) * x_i)$$

$$Subject \ \ to: \begin{cases} d_i(x_i, \vec{\Omega}) - D_i + D_j \leq 0 & \forall j \in \ Fanin(gate - i) \\ d_i(x_i, \vec{\Omega}) = p_i(\vec{\Omega}) + q_i(\vec{\Omega}) * x_i \ \ \forall gate \ \ i \\ D_i <= T^{cons} \ \ \forall gate \ \ i \in PO \\ x_i^{min} \leq x_i \leq x_i^{max} \end{cases} \tag{3.19}$$

where $\vec{\Omega}$ the the randomness field that denotes the randomness in circuit parameters due to fabrication variability. This randomness causes the timing constraints (T^{cons}) at the POs to be violated for certain values $\vec{\omega}$ of $\vec{\Omega}$. Hence, now there will be a likelihood that a given solution for the design has a probability of violating the timing constraints and thus will incur a BYL. In this light, we can redefine the problem objective to minimize the BYL for the design along with the traditional objective function. This can be done by modeling the above linear-programming formulation as a *two-stage stochastic programming formulation* [93].

The first-stage formulation can be written as:

$$Minimize \ \ (\sum_i c_i * x_i + BYL(\vec{x}))$$

$$Subject \ \ to: \begin{cases} d_i(x_i) - D_i + D_j \leq 0 \qquad \forall j \in \ Fanin(gate - i) \\ d_i(x_i) = p_i + q_i * x_i \ \ \forall gate \ \ i \\ D_i <= T^{cons} \ \ \forall gate \ \ i \in PO \\ x_i^{min} \leq x_i \leq x_i^{max} \end{cases} \tag{3.20}$$

which is similar to the formulation presented in 3.17 except that we now additionally minimize the BYL for the design. Formulation 3.20 represents nominal constraints evaluated at the nominal values for all parameters.

From the first stage formulation, we get a solution $\vec{x}$. But in presence of fabrication variability, the delay d_i of each gate w.r.t. x_i also becomes a function of the random field $\vec{\Omega}$. In this scenario, it is possible that the arrival time D_i at the primary outputs may violate the timing constraint T^{cons}. Hence, by the definition of BYL, there will be a penalty imposed on the design depending on the amount of timing violation. Let us define a random variable P that denotes the penalty of violating the timing constraint (T^{cons}) as:

$$P(\vec{x}, \vec{\Omega}) = \begin{cases} (D_i(\vec{x}, \vec{\Omega}) - T^{cons}); \ \ D_i \geq T^{cons} \ i \in PO \\ 0; \quad otherwise \end{cases} \tag{3.21}$$

In equation (4.2), BYL was defined as the expected value of the timing-violation penalty. For a given ($\vec{x}$) and a sample ω of the random field Ω, let $p(\vec{x}, \vec{\omega})$ be the value of the random variable P. By definition, $p(\vec{x}, \vec{\omega})$ denotes the timing-violation penalty for a given ($\vec{x}$) at that variability sample ω. Hence, BYL would be the average timing-violation penalty over all such samples ω which is the expected value of the random variable P for a given ($\vec{x}$). Therefore:

$$BYL(\vec{x}) = E[P(\vec{x}, \vec{\Omega})] \tag{3.22}$$

We can evaluate the timing-violation penalty $p(\vec{x}, \vec{\omega})$ given a fixed $\vec{x}$ and a variability sample $\vec{\omega}$ through another linear formulation that can be written as:

$$p(\vec{x}, \vec{\omega}) = Minimize \quad \sum_{i \in PO} T_i^{viol}$$

Subject to:

$$\begin{cases} d_i(x_i, \omega) - D_i + D_j \leq 0 \qquad \forall j \in \ Fanin(gate - i) \\ d_i(x_i, \omega) = p_i(\omega) + q_i(\omega) * x_i \quad \forall gate \ \ i \\ D_i <= T^{cons} + T_i^{viol} \quad \forall gate \ \ i \in PO \\ T_i^{viol} >= 0 \quad \forall gate \ \ i \in PO \end{cases} \tag{3.23}$$

where T_i^{viol} denoted the extent of timing violation at primary output gate i. For a given value of $\vec{x}$, the optimal objective to this formulation gives us $p(\vec{x}, \vec{\omega})$ which is the desired quantity to compute BYL($\vec{x}$).

The two formulations defined by inequalities (3.20) and (3.23) form a classic Two-Stage Stochastic Programming formulation [93], where the former is called the first-stage problem and the latter second-stage problem.

3.2.3 Stochastic Programming

In the previous section, we presented a motivational example detailing how a timing constrained linear-programming optimization formulation could be modeled as a two-stage stochastic program. We also introduced the idea of BYL as an

optimization objective. In this section, we will generalize this theory and show that it holds for any linear-programming based formulation.

Linear Programming

Linear programming is a very widely used optimization methodology in VLSI design automation. It can be specified as a general linear objective function under a set of linear constraints as follows:

$$Minimize \quad c^{\top}x \tag{3.24}$$

$$subject\ to \quad Ax \leq b \tag{3.25}$$

Several design automation problems have been modeled as linear programs. The polynomial solvability of this formulation makes it a very attractive optimization paradigm.

Optimization Under Uncertainty

Increasing importance of manufacturing variabilities in deep sub-micron technology have made traditional deterministic optimization techniques inaccurate. Variability in problem parameters have made optimization a very difficult task. In a linear programming context, there are three sources of variability: constraint matrix A, vector b and cost vector c, making them non-deterministic random variables. Therefore, the solution to equation 3.24 may be suboptimal due to non-determinism in the cost vector c, or may violate the constraints due to non determinism in A

and b.

Stochastic Programming [66, 21, 121, 93, 57, 55] is a framework that provides a formal methodology for modeling linear programming problems that involve optimization under uncertainty. Stochastic programming is applicable when the variability in vector c, matrix A and vector b can be estimated or modeled as probability distributions.

Variability in Cost Vector c

The uncertainty in cost vector (objective function in equation 3.24) may cause the solution to be sub-optimal. Under such a scenario typically we would like to optimize (minimize) the expected value. The objective function would become $Minimize\ E(c^\top x)$ which is the same as $Minimize\ E(c^\top)x$. Assuming that the non-determinism in problem parameters can be represented as probability distributions, simply replacing c by the $E(c)$ (expected value of c) will ensure the optimality of the expected cost even in the presence of uncertainty. Therefore in subsequent discussions we will assume that variability in c has been addressed by replacing it by $E(c)$. In the next few sections we will discuss the impact of uncertainty due to variability in A and b.

Stochastic Programming with Recourse

Variability in the parameters A and b may cause the violation of several problem constraints after the decision variables x have been fixed. When the real values

of problem parameters becomes known (exact values for matrices A and b), we might find that some of the constraints that are violated, need to be fixed. This is called recourse. Every recourse action is associated with a cost and is typically modeled as the following linear program:

$$h(x,\omega) = \quad Minimize \;\; g(\omega)^{\top} y \tag{3.26}$$

$$subject \;\; to \quad W(\omega)y + T(\omega)x \leq r(\omega) \tag{3.27}$$

$$y \geq 0 \tag{3.28}$$

Essentially $T(\omega)$ is the real manifestation of the constraint matrix A and $r(\omega)$ is the manifestation of b, after the uncertainty has been resolved. Here the parameter "ω" implies the sources of randomness that cause $T(\omega)$ and $r(\omega)$ to be different from A and b. The term $W(\omega)y$ corresponds to the recourse action with "y" denoting the recourse variables and W representing the recourse matrix. Note that if $T(\omega)x \leq r(\omega)$ then the original constraints are satisfied anyway and there is no need for recourse (y = 0). Having a recourse causes an additional cost given by the recourse cost vector $g(\omega)$. This linear programming formulation tries to find the "optimal" recourse such that the recourse cost is minimized. Note that the general theory of stochastic programming assumes that the recourse matrix W and the recourse cost vector "g" are also dependent on randomness. But, in this work we choose to focus on the fixed recourse formulation of stochastic programs where W and g are independent of any randomness. Note that the extent/cost of recourse depends on the value of x (see formulation in equation 3.24) chosen. That is why the cost of the recourse $h(x,w)$ depends on 1) the nature of the randomness ω and 2) the value for

x. For a given value of x, $E(h(x,\omega))$ represents the expected recourse cost for the entire random space of existence of the A and b vectors.

Under the presence of uncertainty, the following formulation considers both the cost of the current solution and the expected recourse cost:

$$Minimize \quad c^{\top}x + E[h(x,\omega)] \tag{3.29}$$

$$subject\ to \qquad Ax \leq b \tag{3.30}$$

Note that variability in the cost vector c can be modeled by simply replacing c by the expected cost vector. Variability in A and b can cause violation of constraints which causes a recourse cost. The objective function tries choose x so as to minimize both the decision cost cx and the expected recourse cost $E[h(x,\omega)]$. The recourse cost is given by the linear programming formulation in equation 3.26. This is a classic two stage stochastic linear programming (SLP) formulation with recourse [57, 55, 66]. The first stage stochastic linear program is given by equation 3.29 and second stage is given by 3.26.

In the first stage, a choice of the decision variable x is made and it does does not consider any variability in parameters. $E[h(x,\omega)]$ is the expected value of the recourse function and ω is a random variable defined on the probability space. The idea is to make a first stage decision on x, after which a random event or variability occurs (ω) changing the outcome of the first stage decision. A recourse action must be taken in the second stage problem to correct any negative impact of the first stage decision after some variability is introduced into the system. In this work we will be interested in only those SLP formulation which have *complete recourse*, i.e.

for any choice of x that is feasible in the first stage problem (equation 3.29), there exists a feasible recourse for possible manifestations of the randomness ω.

3.2.4 SLP and Fabrication Variability

Growing impact of manufacturing variability has inspired a re-investigation of traditional optimization strategies used in VLSI Design automation. Linear programming is a very popular optimization strategy in VLSI CAD problems. Addressing linear programming from a variability perspective naturally inspires us to investigate the stochastic linear programming techniques that address the linear programming problem under uncertainty. Manufacturing variability can cause unpredictability in circuit parameters (matrix A and vector b). Therefore while optimizing/choosing the decision vector we would like to consider this uncertainty such that the overall decision cost and expected recourse cost is minimized. The concept of recourse from a fabrication variability perspective is a bit obscure. The true manifestation of A and b becomes clear only after fabrication. If a certain design constraint is violated at this stage, the chip is wasted.

It is interesting to note that we can make a direct correlation between BYL as presented in section 3.2.1 and recourse. If a chip does not meet its target timing constraint, we can perform operating frequency based binning and sell the chips at a loss that is proportional to how much slower a chip is from its target frequency. This in essence is a recourse that has an associated cost with it. Thus, we can use BYL as the recourse cost in our optimization formulation. In the rest of this work,

we will use BYL to represent the notion of recourse cost.

Let us now try to understand the relationship between recourse cost and BYL in more detail. As defined in subsection 3.2.3, recourse cost $h(x, \omega)$ represents the cost that must be incurred due to violation of constraints when the decision variables have been assigned to value x and the variability sample is ω. In the problem formulation, the objective function given in equation 3.29 considers the expected recourse cost $E[h(x, \omega)]$ as a minimization criteria. In the perspective of timing yield, this expected recourse cost would link with the extent of timing constraint violation that happens due to fabrication variability in parameters. This links closely with the definition of BYL as described in section 3.2.1. Equation 4.2 defines BYL as the expected loss that has to be incurred due to timing violation.

In the previous subsection, if variables y represent the timing constraint violation given a solution x in presence of a variability sample ω, we can define the recourse cost vector $g(\omega)$ to be the BYL penalty function $F_{penalty}(y + T^{cons})$ (where y represents timing violation). In this scenario, the recourse cost $h(x, \omega)$ can be computed from the $BYL(x, \omega)$ as:

$$h(x, \omega) = BYL(x, \omega) \tag{3.31}$$

In order to compute the expected recourse cost $E[h(x, \omega)]$, we need to average $BYL(x, \omega)$ over all samples $\omega \in \Omega$. Hence:

$$E[h(x, \omega)] = BYL(x) \tag{3.32}$$

Hence, we can use BYL to represent the recourse cost in the proposed stochastic programming framework. We note that the discussion of BYL has been presented in the perspective of timing violation. But the notion of BYL can be extended to consider multi-dimensional objectives like timing and leakage BYL. Leakage BYL would represent the loss that has to be incurred when the leakage power constraints of the design are violation. Similar to recourse cost, we can also have a general multi-dimensional BYL.

Theorem: The recourse cost $E[h(x, \omega)]$ is convex in the decision variables x.

Proof: Given a solution x, all the constraints are linear and $BYL(x, \omega)$ has been defined to be a convex function (linear in this work) of the decision variables as well (for a given value of variability sample ω). The total BYL is averaged over all samples $\omega \in \Omega$ which would be a positive sum of convex functions ($BYL(x, \omega)$) which is convex as well. Since recourse cost is the same as $BYL(x)$, we claim that the recourse cost $E[h(x, \omega)]$ is convex.

Several approaches have been proposed to solve the two stage stochastic programming problem. These approaches include successive sample mean optimization (SSMO) based strategies [74, 92] and decomposition based techniques [91, 65, 90]. Another unique approach called stochastic decomposition was also proposed which tries to merge both SSMO and decomposition [57, 55].

3.2.5 Statistical Approximations: Successive Sample Mean Optimization

The objective function of the first stage SLP formulation contains the expected value of the recourse function ($E[h(x,\omega)]$). We can approximate SLP problems with a large number of outcomes by using a sample mean approximation of this function. This technique is similar to Monte Carlo based simulation. The only difference lies in the fact that an optimization problem is solved after each new sample. Essentially we sample the sources of randomness ω iteratively, solving the optimization problem till some stopping criteria is satisfied. For instance, in the kth iteration, we have n_k is the samples from ω. Therefore we have n_k possible values of A and b ($T_1, T_2, T_3....T_k, r_1, r_2.....r_k$). At this step we solve the following linear programming problem:

$$Minimize\ c^{\top}x \quad + \quad \frac{1}{n_k}\sum_{j=1}^{k} H_j \tag{3.33}$$

$$Ax \leq b \tag{3.34}$$

$$Wy_1 + T_1x \leq r_1 \tag{3.35}$$

$$Wy_2 + T_2x \leq r_2 \tag{3.36}$$

$$.. \quad .. \quad .. \tag{3.37}$$

$$Wy_k + T_kx \leq r_k \tag{3.38}$$

$$H_1 = g^{\top}y_1 \tag{3.39}$$

$$H_2 = g^{\top}y_2 \tag{3.40}$$

$$.. \quad .. \quad .. \tag{3.41}$$

$$H_k = g^{\top}y_k \tag{3.42}$$

This problem is solved iteratively (in each iteration we get one new sample) till the optimal solution from iteration k and k+1 are within a user specified range of confidence. It has been shown in [56] that if $n_k \to \infty$ as $k \to \infty$, then the sequence $\{x^k\}$ converges to an optimal solution with probability one. Several SSMO based approximation schemes have been proposed in the literature [92, 74]. Note that as we increase the number of iterations, the total number of samples increase and the size of the formulation shown above increases. Therefore, if the convergence rate is slow, then this technique can become highly cumbersome.

3.2.6 The Cutting Plane Method

Kelley in [53] presented the cutting plane algorithm for convex optimization problems. It was also shown in [91] that

$$E[h(x,\omega)] = \int h(x,\omega)P(d\omega) \tag{3.43}$$

is a convex function. Therefore, the optimal solution to the two stage stochastic linear problem with recourse can be represented as:

$$f(x) = c^\top x + H(x) \tag{3.44}$$

$$X = \{x|Ax \leq b\} \tag{3.45}$$

$$where\ H(x) = E[h(x,\omega)] \tag{3.46}$$

Note that we are only interested in finding x such that only the first stage constraints get satisfied. This is because of the previously assumed *complete recourse* property in which a first stage feasible solution is second stage feasible too. Therefore the optimal solution to the above formulation is definitely optimal solution for the SLP problem. Since $H(x)$ is convex [91], we can use Kelley's cutting plane algorithm for solving the above problem optimally.

Kelley' algorithm provides a technique for us to iteratively generate piecewise linear lower bounding approximations on the expected recourse function ($H(x)$) which is a convex function of x. The kth approximation can be denoted by $v_k(x)$ and objective function in the kth iteration becomes $f_k(x) = c^\top x + v_k(x)$. The technique assumes that X is bounded. As indicated above X is the feasible region of

the linear program. Kelley' algorithm as given in [57] is described below. Due to space limitations several details and proofs have been omitted.

Kelley's Cutting Plane Algorithm:

Step 0 *Initialize*:

Let $\epsilon > 0$ and $x^1 \in X$ be given (one feasible solution obtained by solving the first stage problem without any recourse cost). Let $k \leftarrow 0$ and define $v_0(x) = -\infty$, $u_0 = \infty$, $l_o = -\infty$.

Step 1 *Define/Update the piecewise linear approximation*:

1. k $\leftarrow$ k+1
2. Evaluate α^k, β^k such that H(x) $\geq \alpha^k + \beta^k x$. (Details described later)
3. Add the following to the existing set of constraints $v_k(x) \geq v_{k-1}(x)$, $v_k(x) \geq \alpha^k + (\beta^k)^\top x$. Here v_{k-1} is the lower bounding plane from the previous iteration
4. The new minimization objective is $f_k(x) = c^\top x + v_k(x)$.
5. Let $u_k = Min\{u_{k-1}, f(x^{k-1})\}$

Step 2 *Solve the LP master problem*:

Let $x^k \in argmin\{f_k(x) | x \in X\}$.

Step 3 *Stopping Rule*:

$l_k = f_k(x^k)$. If $u_k - l_k \leq \epsilon$ then stop, otherwise repeat from Step 1.

Essentially in each iteration we generate a new and refined linear lower bound on the convex function H(x) denoted by $\alpha^k + (\beta^k)^\top x$. We add this as a new lower bound constraint to the first stage problem and solve it optimally. Once again this cut generation step is repeated till the convergence criteria of Step 3 is satisfied.

The way these linear lower bounds are generated is briefly described as follows. According to the approach presented by Kelley [53] these linear lower bounds to H(x) can be generated by looking at the Dual of the second stage SLP problem (for brevity we omit the details and the associated theorems). This dual formulation is enumerated below:

$$h(x,\omega) = \quad Maximize \;\; [r(\omega) - T(\omega)x]^\top \pi \tag{3.47}$$

$$subject\ to \quad W^\top \pi \leq g \tag{3.48}$$

It can be seen that the objective function is of the form $\alpha + \beta x$. In the k-th iteration, x is fixed to x^k which is the first stage solution. The dual feasible solution of a primal minimization problem is always a lower bound to the primal optimal solution. This intuition could be used to generate a linear lower bound for H(x). Without going into further details (for brevity), we define the linear lower bounds in the k-th iteration as follows:

$$\alpha^k = \int_\Omega r(\omega)^\top \pi^k(\omega) P(\partial\omega) = E[r(\omega)^\top \pi^k(\omega)] \tag{3.49}$$

$$\beta^k = \int_\Omega -T(\omega)^\top \pi^k(\omega) P(\partial\omega) = E[-T(\omega)^\top \pi^k(\omega)] \tag{3.50}$$

Here π^k corresponds to the solution to the Dual second stage problem in the

k-th iteration with $x = x^k$. We need to understand the computation complexity involved in solving the hyperplane coefficients as given by equation 3.49 and 3.50. This computation inherently involves solving the linear program given by equations 3.47-3.48 for every possible realization of the random variable ω at every iteration k. In most cases, the number of such realization is pretty high and this would correspond to the number of times the linear program would need to be solved for each iteration of Kelley's Method. Therefore, for all practical problems with reasonably large size, this approach becomes rather impractical. Nonetheless, Kelley's algorithm does provide an elegant way in which stochastic programming problems can be solved optimally.

3.2.7 Stochastic Decomposition

As indicated in the previous section, the key problem in Kelley's methodology is the solution to equations 3.49 and 3.50. In general, the analytical solution to these equations is very hard. Therefore, statistical techniques have been proposed to address this issue. Stochastic Decomposition (SD) is a category of algorithms that try to incorporate statistical approximations in decomposition algorithms. Kelley's algorithm is a classic example of decomposition algorithms and has been investigated from a statistical perspective in [57]. Essentially the method proposed in [57] presents a statistical technique for estimating the lower bounding hyperplanes for the expected recourse cost H(x). In this section we will briefly outline the algorithm/methodology presented by [57]. For brevity, we will omit the associated

theorems and proofs.

Once again, just like SSMO based techniques, we iteratively optimize the objective function. In each iteration we generate one new sample from the randomness field ω. Based on the existing set of samples obtained, a lower bounding hyperplane for the expected recourse cost is generated. This optimization is performed iteratively until a pre-specified stopping criteria is satisfied. Without delving into the details we will outline the algorithm. The detailed derivations and proofs can be found in [57]. The basic structure of this algorithm (enumerated below) is similar to Kelley's algorithm.

Basic Stochastic Decomposition Method

Step 0 *Initialize*:

Let $x^1 \in X$ (a feasible solution to the first stage problem) be given and $k \leftarrow 0$

Step 1 *Define/Update the piecewise linear approximation*:

1. $k \leftarrow k + 1$
2. Generate a sample from the random space ω
3. Evaluate a new approximating lower bound hyperplane $\alpha_k^k, +\beta_k^k x$. (Details described later)
4. Update the previous lower bounding approximation to incorporate the new sample in this iteration. (Details described later)
5. Add new constraints $v_k \geq \alpha_k^k + \beta_k^k x$, and $v_k \geq \alpha_t^k + \beta_t^k x$ where $\alpha_t^k + \beta_t^k x$

are the lower bounding hyperplanes from the previous iteration that have been updated/refined to consider the new information provided by the new sample in the current iteration.

6. Set $f_k(x) = c^\top x + v_k$

Step 2 *Solve the LP master problem with f_k as the objective function*:

Let $x^{k+1} \in argmin\{f_k(x)|x \in X\}$.

Step 3 *Repeat from Step 1 if stopping criteria of Kelley's Algo is not satisfied*

Essentially the algorithm first solves the first stage problem assuming the recourse cost is non existent. Then it generates one sample from the random field ω. Based on this new sample it generates a lower bounding hyperplane approximation of the cost. One such lower bounding approximation is generated in each iteration k and is represented as $\alpha_k^k, +\beta_k^k x$. Also, since in each iteration a new sample is generated, more information about the expected recourse function becomes available. Therefore, the lower bounding approximations generated in the previous iteration need to be updated too. These updated lower bounding approximations are denoted as $\alpha_t^k + \beta_t^k x$. Here the subscript "t" indicates the iteration $t \leq k$ where the corresponding approximation was generated. The superscript k corresponds to the current iteration where it has been updated. After generation the new lower bounding hyperplane and updating the existing ones, the following *new* constraints and objective function are added in the first stage problem:

$$Minimize \qquad c^{\top}x + v_k \tag{3.51}$$

$$Ax \leq b \tag{3.52}$$

$$v_k \geq \alpha_t^k + \beta_t^k x \quad \forall t = 1, .., k \tag{3.53}$$

Generating a New Lower Bounding Hyperplane

As indicated earlier, in each new iteration a new sample from the randomness field ω is generated. Let us suppose in iteration k, the linear programming problem in equation 3.51 gives x^k as the optimal solution. We recall from section 3.2.6 that generating the lower bounding hyperplane in the kth iteration in Kelley's Method using equations 3.49 and 3.50 requires the computation of a complicated integral for which there was no known analytical expression. If we replace this integral with sample means using a sample $\{\omega^1, ..., \omega^k\}$ (essentially the samples generated so far), we can get an approximate lower bound. This is performed using the approach presented by [55] which we briefly outline below without delving into details for brevity.

According to the approach in [55], each iteration k solves the second stage problem in equation 3.26 with $x = x^k$ and $\omega = \omega^k$. According to the theory in [55], instead of solving the second stage problem directly, the corresponding Dual problem is solved (equation 3.47). For each of the previous iterations t=1..k, we store all the optimal Dual multipliers π^t. Let Π_k denote the set of all optimal Dual multipliers from iteration 1..k. Without developing the theory in [55] any further we

give the expressions for computing the lower bounding approximate hyperplanes.

In the current iteration k, let π_t^k be defined as follows:

$$\pi_t^k \in argmax\{[r(\omega^t) - T(\omega^t)x^k]^\top \pi | \pi \in \Pi_k\} \tag{3.54}$$

where ω^t is the sample from the iteration t and π_t^k corresponds to that multiplier from the set Π_k which corresponds to the maximum value for $[r(\omega^t) - T(\omega^t)x^k]^\top \pi$. The lower bounding hyperplane coefficients (α_k^k, β_k^k)in the kth iteration are given by:

$$\alpha_k^k = \frac{1}{k}\sum_{t=1}^{k}[r(\omega^t)]^\top \pi_t^k \tag{3.55}$$

$$\beta_k^k = \frac{1}{k}\sum_{t=1}^{k}[-T(\omega^t)]^\top \pi_t^k \tag{3.56}$$

Updating Previously Generated Hyperplane Bounds

In each iteration we define a new lower bounding hyperplane. However, as the iterations progress the sample mean also changes as the sample size keeps increasing. In order to ensure that the previously generated hyperplanes still form a valid lower bound, we need to keep updating their coefficients.

If we are currently in the kth iteration, we would have generated $k-1$ hyperplane coefficients given by $\{(\alpha_t^{k-1}, \beta_t^{k-1})\}_{t=1}^{k-1}$. According to [57], in the kth iteration, we can update the coefficients as follows:

$$\alpha_t^k \leftarrow \frac{k-1}{k}\alpha_t^{k-1} \tag{3.57}$$

$$\beta_t^k \leftarrow \frac{k-1}{k}\beta_t^{k-1} \tag{3.58}$$

$$t = 1, ..., k-1 \tag{3.59}$$

The stopping Criteria

Kelley's algorithm described in the previous section has a stopping criteria which essentially looks at the improvement in the quality of solution from one iteration to the other. If the improvement is less than a user specified limit then we stop. Other stopping criteria which are faster but more complex could also be used here [57].

3.2.8 SLP Applied to VLSI CAD

Let us now try to understand how we can apply the stochastic programming framework to an optimization problem in CAD. Traditional deterministic optimization is not applicable once we consider the effects of variability in parameters. We will consider the problem of sizing MTCMOS transistors for leakage reduction under a delay constraint [116]. The authors have presented a linear programming based approach for optimally sizing sleep transistors under a delay constraint.

Sizing of MTCMOS Sleep Transistors: First Stage Formulation

As presented in [116], a fine-grained MTCMOS sleep transistor scheme places a high threshold sleep transistor at every gate (low threshold) in the circuit. In the standby mode, the leakage current flowing through the low threshold gate sees a high threshold sleep transistor on its way to the ground and hence is reduced considerably.

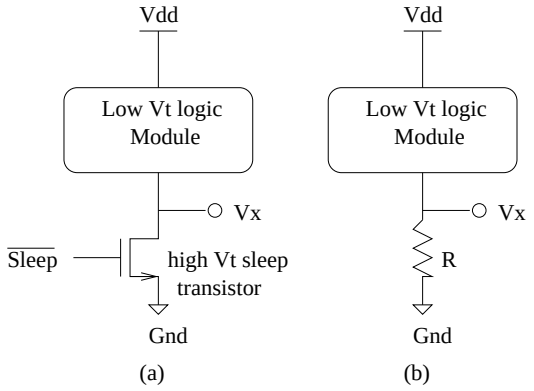


Figure 3.18: Sleep Transistor in MTCMOS Circuits

As shown in figure 3.18(a), low V_t logic modules or gates are connected to the ground rail through high V_t sleep transistors which behave similar to a linear resistor in active mode as shown in figure 3.18(b). The high threshold sleep transistor is controlled using the *Sleep* signal and limits the leakage current to a low value in the standby mode.

The load dependent delay d_i of a gate i in the absence of a sleep transistor can be expressed as:

$$d_i \propto \frac{C_L V_{dd}}{(V_{dd} - V_{tL})^{\alpha}} \tag{3.60}$$

where C_L is the load capacitance at the gate output, V_{tL} is the low voltage threshold

= 400 mV, V_{dd} = 1.8 V and α is the velocity saturation index ($\approx$ 1.3 in 0.18-μm CMOS technology). In the presence of a sleep transistor, the propagation delay of a gate can be expressed as:

$$d_i = \frac{KC_L V_{dd}}{(V_{dd} - V_x - V_{tL})^\alpha} \tag{3.61}$$

where V_x is the potential of the virtual ground rail as shown in figure 3.18 and K is the proportionality constant. Let us suppose $I_{sleep_{ON}}$ is the current flowing in the gate during active mode of operation. During this mode, the sleep transistor is in the linear region of operation. Using the basic device equations for a transistor in linear region, the drain to source current in the sleep transistor (which is the same as $I_{sleep_{ON}}$) is given by:

$$I_{sleep_{ON}} = \mu_n C_{ox}(W/L)_{sleep}((V_{dd} - V_{tH})V_x - \frac{V_x^2}{2}) \tag{3.62}$$

$$\simeq \mu_n C_{ox}(W/L)_{sleep}(V_{dd} - V_{tH})V_x \tag{3.63}$$

The sub-threshold leakage current I_{leak} in the sleep mode will be determined by the sleep transistor and is expressed as:

$$I_{leak} = \mu_n C_{ox}(W/L)_{sleep} e^{1.8} V_T^2 e^{\frac{V_{gs}-V_{th}}{nV_T}}(1 - e^{\frac{-V_{ds}}{V_T}}) \tag{3.64}$$

where μ_n is the N-mobility , C_{ox} is the oxide capacitance, V_{tH} is the high threshold voltage (= 500 mV), V_T is the thermal voltage = 26mV and n is the sub-threshold swing parameter.

We can combine equations 3.61 and 3.63 to get a relationship between gate delay (d_i) and sleep transistor width (W) (assume the length L to be a constant for

all sleep transistors):

$$d_i = \frac{C_1}{C_2 * W - C_3} \tag{3.65}$$

where C_1, C_2 and C_3 are constants.

Let us now try to understand the MTCMOS sleep transistor sizing problem: *Given a circuit with fine-grained sleep transistors placed at the gates, the arrival time at each primary input and a required time constraint at each of the primary outputs, optimally size the sleep transistors for minimal leakage.*

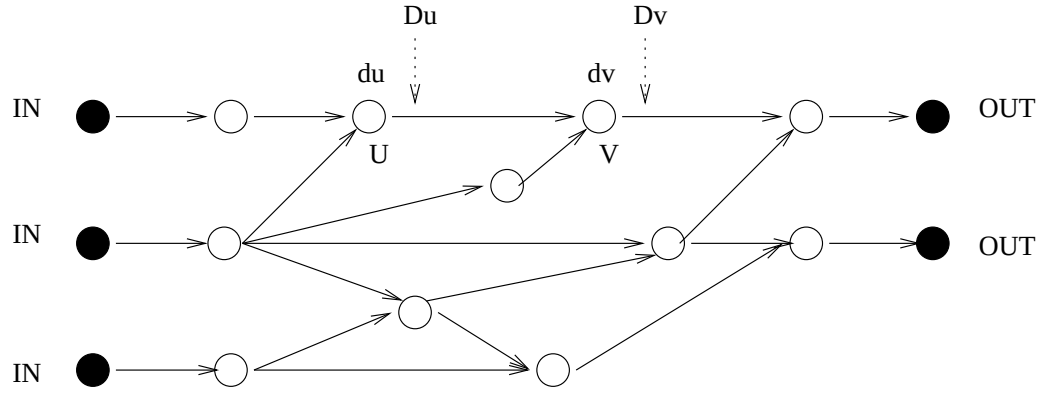


Figure 3.19: DAG representation

This problem can be formulated as a linear program [116]. The circuit can be represented as a DAG, $G(V, E)$ as shown in figure 3.19. Each node in the DAG represents a gate in the circuit. We can add a dummy IN node before each of the primary inputs which are shown as the black nodes marked IN in figure 3.19. We can add a similar dummy node OUT after each of the primary outputs which are shown as the black nodes marked OUT in figure 3.19. For each node u, we associate a variable d_u which represents the delay of that node. We also associate another variable D_u with each node which represents the arrival time at the output of node

u. Now we consider two nodes u and v as shown in figure 3.19. Their corresponding variables have also been shown in the figure. The timing constraints on G(V,E) can be modeled as:

$$d_v - D_v + D_u \leq 0 \qquad \forall\, e(u,v) \in E \tag{3.66}$$

$$D_i = T_i^{arrival} \quad \forall\, vertex\ i \in IN \tag{3.67}$$

$$D_i \leq T_i^{con} \qquad \forall\, vertex\ i \in OUT \tag{3.68}$$

$$d_i = 0 \qquad \forall\, vertex\ i \in IN \tag{3.69}$$

$$d_i = 0 \qquad \forall\, vertex\ i \in OUT \tag{3.70}$$

It has been shown in [116] that gate delay (d_i) has a convex relationship with sleep transistor width (W). Therefore equation 3.65 (which defines this relationship) can be simplified through piecewise linearization. Let us have p piecewise linearizations of the delay/width relationship (equation 3.65). The delay of a node u has the following relationship with these p lines:

$$d_u \geq m_u^1 W_u + c_u^1 \tag{3.71}$$

$$d_u \geq m_u^2 W_u + c_u^2 \tag{3.72}$$

$$\ldots \quad .. \quad \ldots \tag{3.73}$$

$$d_u \geq m_u^p W_u + c_u^p \qquad \forall\, vertex\ u \tag{3.74}$$

Here m_u, c_u are the linearization parameters and W_u is the width of the sleep transistor connected with gate u. Adding these constraints to equations 3.66-3.70 helps assign MTCMOS transistor widths such that the timing constraint of the DAG

is satisfied. Each sleep transistor width has a valid range of existence and must also be added to the constraints:

$$W_u^{min} \leq W_u \leq W_u^{max} \qquad \forall\ vertex\ u \tag{3.75}$$

The leakage current of each gate u can be expressed as a function of its sleep transistor size using equation 3.64:

$$I_{leak}^u = K * W_u \qquad \forall\ vertex\ u \tag{3.76}$$

where K is a constant. These additional variables are also added to the constraints. Now we would like to assign the widths to each sleep transistor such that the timing constraints are satisfied and the overall leakage is minimized. The objective function can be stated as follow:

$$min(\sum_{u \in V} I_{leak}^u) \tag{3.77}$$

This completes the linear programming formulation for the problem of MTCMOS sleep transistor sizing for minimum leakage current under a delay constraint. We note that the above formulation can easily be re-arranged to be in the general linear program form as shown in equations 3.24-3.25. In this formulation, the decision variables x include W_u the width of MTCMOS sleep transistor, D_u the arrival time at the output, d_u the gate delay and I_{leak}^u the leakage current for each gate u in the benchmark. The constraints and objective can be re-arranged to be in the form $Ax \leq b$. From a SLP perspective, the above formulation (equations 3.66-3.77) forms the first stage SLP formulation (with the addition of a recourse function cost $E[h(x, \omega)]$) as given by equations 3.29-3.30.

MTCMOS Sizing Problem Under Variability: Second Stage Formulation

Fabrication variability causes several circuit parameters that are assumed to be constants in the previous formulation to have random behavior. Circuit parameters like chip threshold voltage, effective channel length L_{eff} etc. have been shown to have such randomness due to fabrication variability [98]. Other circuit parameters like chip temperature, supply voltage etc. also have randomness due to environmental uncertainties. This motivates us to formulate optimization problems like the previous ones from a stochastic perspective. Randomness in circuit parameters will cause the constants C_1, C_2, C_3 in equation 3.65 to become uncertain. Therefore the parameters of the approximating linearization m_u^i, c_u^i in equations 3.71-3.74 become random numbers too. Moreover, the constant K in leakage I_{leak}^u vs sleep transistor width W_u relation in equation 3.76 will also become a random variable. These random variables will show correlated behavior since they are inspired by variability in basic circuit parameters. This correlation can be easily considered in the optimization process by appropriately sampling the random field.

As mentioned earlier, the first stage problem assigns values to decision variables which corresponds to W_u the width MTCMOS sleep transistor, D_u arrival time at the output, d_u the gate delay and I_{leak}^u the leakage current for each gate u in the benchmark. Due to fabrication and environmental uncertainty the constraints represented by equations 3.71-3.74 may get violated for the assigned values of d_u and W_u. Therefore each delay variable d_u has a recourse Y_u given by

$$Y_u \geq m(\omega)_u^1 W_u + c(\omega)_u^1 - d_u \tag{3.78}$$

$$Y_u \geq m(\omega)_u^2 W_u + c(\omega)_u^2 - d_u \tag{3.79}$$

$$\ldots \quad .. \quad \ldots \tag{3.80}$$

$$Y_u \geq m(\omega)_u^p W_u + c(\omega)_u^p - d_u \tag{3.81}$$

$$Y_u \geq 0 \tag{3.82}$$

These constraints essentially add an extra offset to the gate delay value d_u that has been fixed by the first stage problem. These offsets ensure that in presence of random manifestations of the circuit parameters which affect the parameters of the piecewise linearizations, the delay d_u vs W_u relationship is not violated. Essentially each gate gets a new delay value $(d_u + Y_u)$ such that the corresponding constraints get satisfied. Now, since the gate delay changes (recourse on gate delay occurs), it has to affect the arrival time values also. Since the arrival time variables D_u have also been fixed by the first stage problem, we add recourse variable Z_u. The relationship between Z_u and Y_u is as follows:

$$(Y_v + d_v) - (Z_v + D_v) + (Z_u + D_u) \leq 0 \ \ \forall \ e(u,v) \in E \tag{3.83}$$

$$Z_u \geq 0 \quad \forall u \in V \tag{3.84}$$

Since the delay of each gate becomes $Y_u + d_u$, we modify the arrival times for each gate such that equations 3.66-3.70 are still satisfied. This essentially implies that we make the arrival time for a gate $Z_u + D_u$ through the recourse variable Z_u.

The variables Y_u and Z_u form our recourse vector y as given in equation 3.27.

From a timing perspective, we loose yield if the circuit timing violates the required timing constraints at the primary outputs. Note that each node arrival time with recourse becomes $D_u + Z_u$. If $Z_u > 0$, then the arrival time of the corresponding gate is higher than that predicted by the first stage problem. Therefore, if the recourse variable Z_u for any of the primary outputs is positive then a valid recourse is needed for the DAG to satisfy the timing constraints at the primary outputs. This implies that the first stage solution violates the timing constraints. In terms of BYL, this would imply that this chip would incur a penalty cost which is proportional to the degree of slowdown from the target timing constraint. Therefore, we add a penalty-cost (which is the BYL) for assigning a positive value for any Z_u associated with the primary outputs. The objective for the second stage problem becomes:

$$h(x, \omega) = Min(g(\omega)^\top y) = Min(\sum_{u \in PO} Cost_1 Z_u) \tag{3.85}$$

This cost function essentially implies that if the randomness ω is such that the primary output timing constraint gets violated then we get a positive recourse cost (recourse is the same as BYL). Note that the value $\sum_{u \in PO} Cost_1 Z_u$ is proportional to Z_u (similar to BYL as described in section 3.2.1). Hence, if a solution has a larger degree of violation of the timing constraint then it will have a higher recourse cost. This completes the description of the second stage problem.

This SLP formulation tries to assign values to the decision variables x such that a minimum degree of timing violation occurs due to fabrication variability. This formulation can be solved using the techniques presented in the previous sections.

Extension to multi-dimensional leakage and delay yield

The formulation that we have proposed in the previous section is general and can be extended to consider multi-dimensional yield objectives. We can also consider leakage current based yield in the optimization objective. Similar to delay based BYL, we can also define BYL for leakage violation and consider it during optimization as follows.

Due to randomness in circuit parameters, the parameter K in equation 3.76 will also become uncertain. Therefore, for the I^u_{leak} assigned by the first stage problem, constraint 3.76 might get violated. The nature of recourse for the leakage current variables is different from that of the delay variables. This is because leakage is an additive quantity and the violation of equation 3.76 is not catastrophic from a yield point of view. We are only interested in modeling situations when the *total* chip leakage becomes more than that predicted by the first stage problem. Therefore, we add a recourse variable Q that corresponds to the total increase in leakage current compared to that predicted by the first stage problem. Therefore, we add the following constraint:

$$Q = \sum_{u \in V} (K(\omega) * W_u - I^u_{leak}) \tag{3.86}$$

$$Q \geq 0 \tag{3.87}$$

The variables Y_u, Z_u and Q form our recourse vector y as given in equation 3.27. In this case, if the total leakage due to variability is less than that predicted by the first stage problem then Q is zero. If Q is positive then the leakage predicted by

Benchmark	Deterministic LP		Stochastic Decomposition		SSMO	
	% Delay Violation	Avg. Leakage	% Delay Violation	Avg. Leakage	% Delay Violation	Avg. Leakage
C432	49.3	1420.0	0	6283.1	DNC	DNC
C499	47.3	3784.1	0	9465.2	DNC	DNC
C880	46.0	2540.3	0	8649.3	DNC	DNC
C1355	48.0	3731.0	0	8757.8	DNC	DNC
i1	44.0	436.9	0	1109.1	0	774.9
i2	58.7	1536.8	0	5039.5	DNC	DNC
i3	50.7	1700.6	0	6119.2	DNC	DNC
i5	45.3	1503.3	0	5668.8	DNC	DNC
x1	43.3	1903.9	0	8036.5	DNC	DNC
x4	47.3	2460.6	0	10487.1	DNC	DNC
Average	48.0		0		DNC = Did Not Converge	

Table 3.5: Result: Delay Constraint Violation and Average Leakage Current

the first stage problem underestimates the real leakage after fabrication. Therefore, we also assign a penalty-cost for having a positive value for Q (if there is a power constraint on the design, then this would be the BYL due to power constraint violation). The objective function for the second stage problem becomes:

$$h(x,\omega) = Min(g(\omega)^{\top} y) = Min(\sum_{u \in PO} Cost_1 Z_u + Cost_2 Q) \tag{3.88}$$

This cost function essentially implies that if the randomness ω is such that the primary output timing constraint gets violated or the leakage current increases then it gives a positive recourse cost (recourse is the same as BYL). This SLP formulation tries to assign values to the decision variables x such that a minimum degree of timing violation occurs due to fabrication variability and there is a minimum increase in leakage current.

3.2.9 Experimental Results and Comparisons

We implemented the MTCMOS sizing formulation for leakage optimization as described in section 3.2.8 using stochastic linear programming in SIS [37]. We assumed that there is variability in threshold voltage due to manufacturing uncertainty. The threshold voltage variations can be spatially correlated and also have an independent randomness component [26]. In order to capture the spatial correlations in our experimental framework, we generated a standard-cell placement of the benchmarks using CAPO. For each gate i, we calculated its physical distance from the corners of the chip (say r_1^i, r_2^i, r_3^i and r_4^i). We assumed that threshold variability in the chip was dependent on four independent random variations $alpha_1$, $alpha_2$, $alpha_3$ and $alpha_4$ located at the four corners of the chip.

For the low threshold voltage (V_{tL}) variations, the random variables were taken to be uniformly distributed with zero mean between a range of (-0.06,0.06)V. For the high threshold voltage (V_{tH}) variations, the random variables were taken to be uniformly distributed with zero mean between a range of (-0.075,0.075)V. The scheme proposed in this work is independent of the nature of the distributions, hence we have assumed the random variables to be uniformly distributed for illustration purposes. Additionally, the threshold voltages also have an independent randomness term denoted by δR. Hence, the low and high threshold voltages at gate can be written as:

$$V_{tL} = V_{tL_0} + f(r_1^i)alpha_1 + f(r_2^i)alpha_2 + f(r_3^i)alpha_3 + f(r_4^i)alpha_4 + \delta R \quad (3.89)$$

$$V_{tH} = V_{tH_0} + f'(r_1^i)alpha_1 + f'(r_2^i)alpha_2 + f'(r_3^i)alpha_3 + f'(r_4^i)alpha_4 + \delta R \tag{3.90}$$

where V_{tL_0} is the mean value of the low threshold voltage (400mV) and V_{tH_0} is the mean for high threshold (500mV) for high threshold). f and f' are functions that can be chosen depending on the variability data available for used technology node. For each gate, the total variations as shown in equations 3.89 and 3.90 were scaled to be uniformly distributed in a 15% range around the corresponding mean values. The independent randomness (δR) comprised a maximum 1% variation around the means. Since the threshold voltage variation at each gate is modeled in terms of the same four independent random variations ($alpha_1 - alpha_4$), the spatial correlations are inherently captured. From equations 3.89 and 3.90, we can see that two gates that are placed in close physical proximity will see very similar threshold voltage variations and those places further apart will not see similar threshold voltage variations. *This is just one possible correlation modeling scheme that we have used and the stochastic programming framework is general enough to use any other modeling scheme.* The delay constraint was set to be 8% higher than the minimum delay for each benchmark. All linear programming formulations were solved using CPLEX. We experimented with stochastic decomposition as well as the SSMO technique and made comparisons with the traditional deterministic linear programming approach. Experiments were done on various benchmarks from the MCNC benchmark suite.

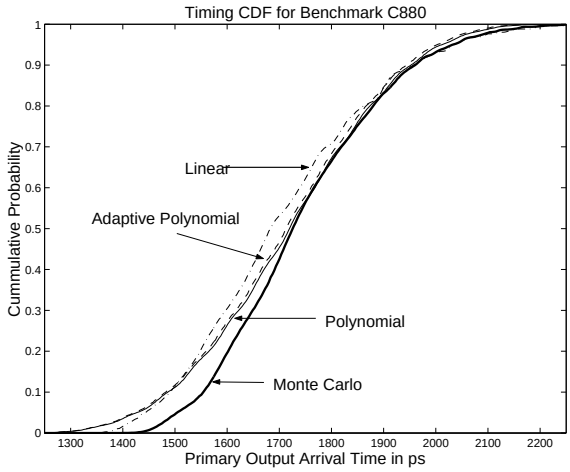


Figure 3.20: Timing Result for C880

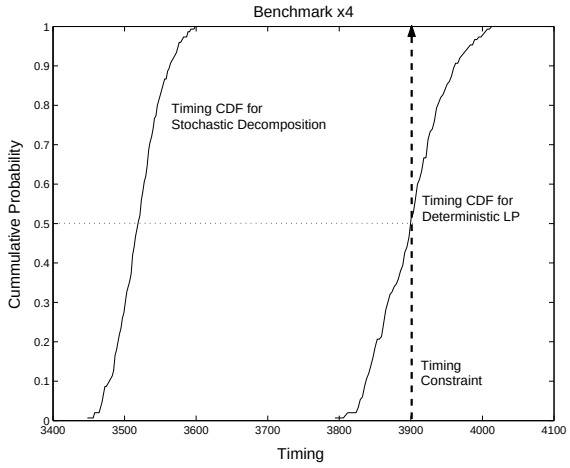


Figure 3.21: Timing Result for x4

Table 3.5 presents the experimental results. We have made a comparison between deterministic linear programming, stochastic decomposition and the SSMO approach as described in the earlier sections. Deterministic linear programming based MTCMOS sizing uses the formulation described in section 3.2.8. It utilizes the available delay slack at each gate to decrease the sleep transistor widths during sizing (thereby increasing the gate delays and reducing leakage current) as long

as the delay constraints are satisfied. Since it tries to leverage maximum delay slack, it performs aggressive sizing (many gates become timing critical). In presence of threshold variability, a solution from such a deterministic approach may easily violate the timing constraints on the critical paths. Hence, such a deterministic approach is vulnerable to variability. We generated a MTCMOS sizing solution from the deterministic approach, the stochastic decomposition approach and the SSMO approach. We performed Monte-Carlo runs (by taking different samples of threshold voltage variability) to generate a timing CDF at each of the primary output gates and the expected (average) value of leakage currents. Given the timing constraint, we evaluated the probability of violating the delay constraints for each benchmark.

Columns 2 and 3 in table 3.5 present the results obtained from deterministic MTCMOS sizing. We can see that on an average there was a 48% violation in delay constraint due to the threshold variability ($YL = 0.48$). On the other hand, columns 4 and 5 present the results obtained from the stochastic decomposition approach. Firstly, we note that there is no delay constraint violation in any of the benchmarks which implies that $BYL = 0$(and $YL = 0$) for all the benchmarks. We would like to point out that the optimization objective in our formulation was BYL and not YL. Our results indicate that there is a strong correlation between the two and improving BYL does improve even the traditional YL to a large extent. This shows the effectiveness of the SD approach in dealing with threshold variability. However, the average leakage currents obtained in the SD approach are much higher than those from the deterministic approach. This happens because the

SD approach performs less aggressive sizing (to account for the timing uncertainty due to threshold variations) and hence keeps the average width of sleep transistors higher than that obtained from deterministic approach. This implies that the leakage current obtained from the SD approach will be higher than that from the deterministic approach. So, at the cost of increased average leakage current, we are able to generate a MTCMOS sizing solution that is immune to variability in the threshold voltage in terms of violating the timing constraints. It is important to note that MTCMOS sleep transistor insertion reduces the leakage by orders of magnitude and hence a linearly scaled difference in leakage between the stochastic and deterministic formulation when compared to the initial leakage value is not significant.

Figures 3.20 and 3.21 presents the timing CDFs obtained for two benchmarks for the deterministic approach as well as the SD approach. We can see from the figures that the deterministic approach has a large probability of violating the timing constraints.

We also ran experiments with the SSMO technique described in section 3.2.5. As shown in columns 6 and 7 in table 3.5, only one small benchmark $i1$ converged to a solution. The rest of the benchmarks did not converge within a reasonable runtime (8 hours). As the sample size increases, the SSMO formulation becomes very large in size and hence takes a very long time to converge to a solution. These results clearly bring out the efficiency obtained from the stochastic decomposition approach in terms of runtime as compared to SSMO based stochastic linear programming techniques.

Benchmark	Runtime for Stochastic Decomposition (in cpu cycles)
C432	126881
C499	253730
C880	211785
C1355	1342305
i1	6685
i2	43490
i3	14535
i5	251825
x1	183815
x4	428435

Table 3.6: Result: Runtime in cpu cycles

Stochastic technique based Monte-Carlo optimization has a higher runtime when compared to deterministic optimization. Depending upon the nature of the variability and the required accuracy, SD runs multiple iterations of the deterministic formulation (first stage SLP) and hence takes more runtime as compared to one run of the deterministic formulation. But the quality of solution that is obtained from stochastic techniques is high and much more robust to fabrication variability. This has been demonstrated by the experiments results in terms of Yield-Loss values. The runtime for SSMO is much higher than that of SD because of the continuous increase in problem size with each iteration rendering it impractical for large problems. Table 3.6 shows the runtime in cpu cycles for stochastic decomposition. It is expected that the runtime would increase as design size increases. Also, depending on the modeling strategy used and the number of samples that are used during optimization, the runtime complexity would change. For our experiments, we used 200 sampling vectors in computing the recourse using the second stage formulation.

In this work we have applied the concept of stochastic linear programming to optimization problems in CAD. In the deep sub-micron technology, increasing manufacturing variabilities have made deterministic optimization less effective. Our work presents a framework under which optimization under variability can be performed within the linear programming paradigm. There are several problems in CAD that have been solved using linear programming and it will be interesting to extend this stochastic decomposition concept to those problems. The major bottleneck is stochastic programming is in estimating the coefficients of the lower bounding hyperplanes as given by equations 3.49 and 3.50. Estimating these coefficients

efficiently and accurately is an interesting direction for future work.

Chapter 4

Variability-Aware Design Optimization: Post-Silicon Tunability

The technique of Post-Silicon tunability has tremendous potential to counter the uncertainty introduced in design performance due to fabrication variability. Post-silicon tunability entails building tuning knobs into the design that can be tweaked after the chip has been manufactured to selectively change the performance of certain parts of the chip in order to counter the impact of variability and improve timing yields. This would allow the manufacturer to tune each chip individually to try and meet the required performance constraints. Recently, post-silicon tunable (PST) clock-tree synthesis [72, 114, 82, 36] has been proposed as one such approach that can be applied to high performance designs to correct timing violations. Such post-fabrication yield improvement techniques can be very powerful to counter variability since each chip can be tuned independently. In [114], the authors present how the PST technique has been applied to Intel's Dual Core Itanium processor to improve timing yields. It can be noted that having PST in the design incurs a cost overhead both in terms of hardware (area) and power. Techniques like adaptive body-biasing and supply voltage scaling can also be used to provide tuning knobs in the design that can be used to change the performance of the design after fabrication.

In this work [117], we have proposed a design management philosophy to balance the effort between pre-silicon and post-silicon tunability in order to get maximum yield gains at minimal design overhead.

4.1 Variability-Driven Formulation for Simultaneous Gate Sizing and Post-Silicon Tunability Allocation

Process variations cause design performance to become unpredictable in deep sub-micron technologies. Several statistical techniques (timing analysis, gate-sizing, buffer insertion) have been proposed to counter these variations during the optimization phase of the design flow to get better timing yields. Another interesting approach to improve timing yield is post-silicon tunable (PST) clock-tree. However, gate sizing and PST clock tree management have not been integrated together into a single framework for better optimization. In this work, we propose such an integrated framework that performs simultaneous statistical gate-sizing in presence of PST clock-tree buffers for minimizing binning-yield loss (BYL) and tunability costs by determining the ranges of tuning to be provided at each buffer. The simultaneous gate-sizing and PST buffer range determination problem is proved to be a convex stochastic programming formulation under longest path delay constraints and hence solved optimally. We further extend the formulation into a heuristic to additionally consider shortest path delay constraints. We make experimental comparisons using nominal gate sizing followed by PST buffer management using [72] as a base-case. We take the solution obtained from this approach and perform 1) Sensitivity-based

statistical gate-sizing while retaining the PST clock tree 2) Simultaneous gate sizing and PST buffer range determination as proposed in this work. On an average, the BYL obtained from our approach is 98% lower than the base-case ([72]) and 95% lower than the sensitivity-based algorithm. On an average the base-case approach ([72] gave 22% timing yield loss (YL), the sensitivity approach gave 19% YL, where as our proposed algorithm gave only 3% YL. The total PST tuning buffer range that is allocated through the proposed algorithm is comparable to that obtained from [72]. The proposed algorithm had a $2.2x$ runtime speedup compared to the sensitivity-based algorithm.

4.1.1 Introduction

Process variations are posing a major challenge to IC designers in the nanometer regime. They cause a significant spread in the performance distribution of designs, making traditional design and analysis techniques to become inaccurate. There has been a distinct shift in VLSI design paradigm to try and develop variability aware methodologies.

In high performance designs, process variations result in a spread in the achievable frequency, thereby causing some chips to fail from meeting the nominal target frequency. In [98], the authors have mentioned that as much as 30% frequency variation can be observed in high-performance designs. Chips can be *binned* according to their operating frequency. Those that fail to meet the target frequency can either be sold at a loss or be discarded. In [13], the authors present a hardware design to

perform speed binning in microprocessor design. Each speed bin has a corresponding penalty cost that is proportional to its slowdown from the target frequency. Thus, there exists a binning-yield loss with each design depending on the spread in its operating frequency due to process variations. In this work, we use binning yield loss (BYL) as an optimization objective in our formulation.

A lot of recent work has focused on statistical techniques for considering process variability during analysis and optimization. One such direction of research has been timing analysis in presence of variability. Statistical Timing Analysis has emerged as a powerful tool to predict the timing distribution of designs [42, 127, 119, 43]. Other recent approaches have tried to utilize this available statistical information about the design to perform statistical optimizations like gate sizing [77, 97, 4, 32, 67, 80]. Essentially, these are analysis and optimization techniques that can be used to counter variability at design time.

Post-silicon tunability is another technique to improve timing yield in circuits. This would allow the manufacturer to tune each chip individually to try and meet the required performance constraints. Recently, post-silicon tunable (PST) clock-tree synthesis [72, 114, 82, 36] has been proposed as one such approach that can be applied to high performance designs to correct timing violations. Such post-fabrication yield improvement techniques can be very powerful to counter variability since each chip can be tuned independently. It can be noted that having PST in the design incurs a cost overhead both in terms of hardware (area) and power. This can be interpreted as the cost of tunability in the design.

There is no existing work that tries to integrate both post-silicon and pre-

silicon optimization paradigms into one flow. While performing design time optimization (say gate sizing) one can leverage the information about the available post-silicon tunability and vice-versa. The work in [72] determines the locations of the PST buffers and also their ranges. In this work, we do not decide the location of the PST buffers. The PST clock tree structure as determined by [72] is taken as an input to our algorithm. We retain the PST buffer locations and clock tree structure but perform simultaneous gate sizing and PST buffer range determination for improved BYL. Hence, the proposed approach in this work can be used alongwith [72] for more robust design solutions. Additionally, our formulation optimally solves the simultaneous gate sizing and PST buffer range determination problem under longest path delay constraints. Our technique can handle any distribution of variations with any arbitrary correlation model.

The problem that we address in this work can be formally stated as:

Given a sequential design with a synthesized PST clock-tree (with known tunable buffer locations), we perform simultaneous gate sizing of the combinational logic gates and tuning range determination of each PST buffer, such that the Binning Yield Loss and Tunability Cost is minimized.

We formulate this problem as a two-step stochastic program [93]. We will first develop a formulation considering only longest path constraints. We will prove that it is a convex formulation and hence can be solved optimally. We extend this formulation further into a heuristic considering shortest path constraints (which are

inherently non-convex). We use the Kelley's Cutting Plane Method [93] to solve the formulations.

We make experimental comparisons using nominal gate sizing followed by PST buffer management using [72] as a base-case. We take the solution obtained from this approach and perform 1) Sensitivity-based statistical gate-sizing (similar to [32, 4]) while retaining the PST buffer locations and ranges as determined in the base-case [72] in an effort to re-optimize the design. 2) Simultaneous gate sizing and PST buffer range determination as proposed in this work. On an average, the BYL obtained from our approach is 98% lower than the base-case ([72]) and 95% lower than the sensitivity-based algorithm. On an average the base-case approach ([72] gave 22% timing yield loss (YL), the sensitivity approach gave 19% YL, where as our proposed algorithm gave only 3% YL. The total PST tuning buffer range that is allocated through the proposed algorithm is comparable to that obtained from [72]. The proposed algorithm had a $2.2x$ runtime speedup compared to the sensitivity-based algorithm.

4.1.2 Background and Definitions

In this subsection, we will discuss the relevant background information that is needed to understand this work.

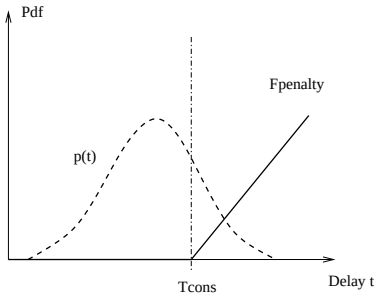


Figure 4.1: Binning Yield Loss with a Convex Penalty Function

Binning-Yield Loss

In high performance designs, process variations result in a spread in the achievable frequency, thereby causing some chips to fail from meeting the nominal target frequency. In [98], the authors have mentioned that as much as 30% frequency variation can be observed in high-performance designs. Chips can be *binned* according to their operating frequency. Those that fail to meet the target frequency can either be sold at a loss or be discarded. In [13], the authors present a hardware design to perform speed binning in microprocessor design. The penalty that the chips in a speed bin have to incur is proportional to the slowdown from the target timing constraint (T_{cons}). Let us suppose that the timing delay of the chip is t. We define a BYL penalty function $F_{penalty}(t)$ as follows:

$$F_{penalty}(t) = \begin{cases} q(t - T_{cons}); & t \geq T_{cons} \\ 0; & otherwise \end{cases} \tag{4.1}$$

where $q(t - T_{cons})$ is assumed to be a convex function. Let us suppose the probability density function (pdf) of circuit delay is $p(t)$ as shown in figure 4.1. Hence for longest path constraints, we can define the BYL for the design as:

$$BYL = \int_{-\infty}^{\infty} F_{penalty}(t)p(t)dt = \int_{T_{cons}}^{\infty} q(t - T_{cons})p(t)dt \tag{4.2}$$

BYL for shortest path constraints can also be defined similarly. In the optimization framework proposed in this work, we will use the above definition for BYL.

Traditional Gate Sizing

The traditional gate sizing problem tries to minimize the cumulative sum of gate sizes while assigning a size to each gate in the circuit such that the timing constraint T_{cons} at the primary outputs are met. Let x_i denote the size of gate i. The delay of the gate d_i is a function of its size and the sizes of all its fanout gates and hence is denoted as $d_i(\vec{x})$. In general, we perform sizing by varying the channel widths of each transistor in the gate (hence gate size x_i is proportional to the channel width), while the channel lengths are kept constant. If we denote the arrival time at gate i as t_i. The traditional gate sizing problem can be written as:

$$\begin{aligned} & Minimize \quad \textstyle\sum_{\forall gate\ i} c_i \times x_i \\ Subject\ \ to: & \begin{cases} t_j + d_i(\vec{x}) \leq t_i \quad \forall j \in fanin(i); \forall gate\ i \\ t_i \leq T_{cons} \qquad \forall i \in PO \\ x^i_{min} \leq x_i \leq x^i_{max} \quad \forall gate\ \ i \end{cases} \end{aligned} \tag{4.3}$$

where c_i is a positive weighting constant for each gate. In this simple formulation, we propose to optimize the total area of the gates which is the most common optimization objective [67, 80]. Additionally, one gate can perform gate sizing to minimize the power [77, 97] or yield-loss [4, 32].

Convex Gate Delay Modeling

As shown in [54, 107], the elmore delay of a gate can be modeled as a posynomial function of the transistor sizes $\vec{x}$. We can model each transistor as an equivalent resistor and capacitor whose magnitudes are proportional to the channel width w of each transistor. Elmore delay of gate i can be written as a posynomial functions of these resistors and capacitors of gate i and the capacitors of its fanout gates. As shown in [107], gate delay can be written as a function of its size x_i (since it is proportional to the channel width w). Hence, the posynomial gate delay can be expression as:

$$d_i(\vec{x}) = a_{0i} + a_{1i}\frac{\sum_{\forall j} x_j}{x_i} \quad j \in fanout(i) \tag{4.4}$$

where a_{0i} and a_{1i} are positive constants that depend on circuit parameters such as threshold voltage, effective channel length, supply voltage and oxide thickness. This posynomial gate delay representation can be changed into a convex form but making a change of variables $x_i = e^{y_i}$. Each arrival time variable t_i in the gate sizing formulation can be represented as $t_i = e^{z_i}$. Hence, the gate sizing formulation can be presented as:

$$\begin{aligned} & Minimize \quad \sum_{\forall gate\ i} c_i \times e^{y_i} \\ Subject\ \ to: & \begin{cases} t_j(z_j) + d_i(\vec{y}) \leq t_i(z_i) \ \ \forall j \in fanin(i) \\ t_i(z_i) \leq T_{cons} \ \ \forall i \in PO \\ x^i_{min} \leq e^{y_i} \leq x^i_{max} \quad \forall gate\ \ i \end{cases} \end{aligned} \tag{4.5}$$

All variables have an exponential representation which makes the above gate sizing formulation convex in $\vec{y}$ [99].

Post-Silicon Tunable Clock Tree

Several recent work [72, 114, 82, 36] have proposed that PST clock tree can improve the timing yield for designs in presence of process variations. The central idea is to insert post-silicon tunable buffers into the clock tree that can be used to introduce extra slack into the critical paths in order to correct the timing violations by adjusting the clock skews. In [72], the authors have proposed an approach for PST clock-tree synthesis that tries to minimize the total number of candidate PST clock buffer locations and also reduce the hardware cost of each PST buffer by computing its required tuning range. It is important to note here that inserting redundant PST buffers into the clock tree may results in significant overhead in chip area. Moreover, since the clock buffer also have some capacitance, they also increase the power consumption of the clock tree.

There is no existing technique that tries to optimize the design for delay while determining the PST buffer ranges. In this work, we perform simultaneous gate sizing and PST range determination for better design optimization. Such an optimization framework allows gate sizing to leverage the presence of PST buffers in the design to potentially prevent aggressive oversizing of the design. Additionally, the optimization of delay paths results in more effective tunable range allocation without increasing the tunability cost (silicon area and power). The PST clock tree

with its buffer locations as obtained from [72] is used as an input to our algorithm.

Let us try to understand how a PST clock tree can help improve timing yield. Given a sequential design, we can represent it as a graph $G = (V, E)$, where V is a set of flip-flops (FFs) and E is a set of edges representing timing arcs between the FFs. An edge e_{ij} would represent a combinational logic path between flop i and j. Let us suppose that T_i and T_j are the clock arrival times at flops i and j respectively (they may not be the same due to clock skew). In this work, we look to satisfy the longest path constraint in sequential design for BYL optimization. Let the maximum delay between all combinational logic paths between FFs i and j be D_{ij}. Let the setup time for flip-flop (FF) j be T_{set}^{j} and T_{clk} be the nominal clock period. In order to meet the longest path timing constraint, the circuit needs to satisfy the following inequality:

$$T_i + D_{ij} \leq T_{clk} + T_j - T_{set}^{j} \tag{4.6}$$

Now, as shown in figure 4.2 let us suppose that we have a PST clock tree with tunable buffers $B1 - B7$ as shown. Each of these tunable buffers k has a tuning delay T_k^{Buf} that can be in the range of 0 to R_k^{max} which has been decided during the design stage (pre-fabrication):

$$0 \leq T_k^{Buf} \leq R_k^{max} \tag{4.7}$$

Now, as is evident from figure 4.2, each FF i can have its clock arrival time T_i adjusted by tuning appropriate buffers that lie on the path between the clock

tree source and itself. For example, FF 1 can be affected by PST buffers $B1$, $B2$ and $B4$. Hence, if a path starting at FF 1 violates the timing constraint (equation (4.6)) post-fabrication due to process variability, we can adjust the tuning of the corresponding buffers to try to bring the path back into feasibility region. Each FF i is affected by a subset C_i of PST tunable buffers and hence this technique can be used to redistribute timing slack between critical and non-critical paths such that maximum timing violations can be mitigated. Also, it is easy to note that since many FFs share the same PST buffer, this tuning needs to be done carefully to ensure that no other path violates its timing constraint. In essence, we can re-write equation (4.6) considering PST tunability as:

$$(T_i + \sum_{k \in C_i} T_k^{Buf}) + D_{ij} \leq T_{clk} + (T_j + \sum_{k \in C_j} T_k^{Buf}) - T_{set}^j \tag{4.8}$$

Let us consider an example from figure 4.2 to better understand this technique. Let us suppose that there is a combinational logic path ($path - 1$) between FFs 1 and 5 that violates the longest path timing constraint (equation (4.6)). We can tune the clock buffers $B3$ and $B6$ to assign more clock skew to this path. Assuming there is sufficient range available at each of these buffers, to bring the path back into the feasibility region, we would have:

$$(T_1 + T_2^{Buf} + T_4^{Buf}) + D_{15} \leq T + (T_5 + T_3^{Buf} + T_6^{Buf}) - T_{set}^j \tag{4.9}$$

where the exact tuning delay of each buffer (T_i^{Buf}) needs to be adjusted to satisfy the above constraint. However, one needs to understand that adding this

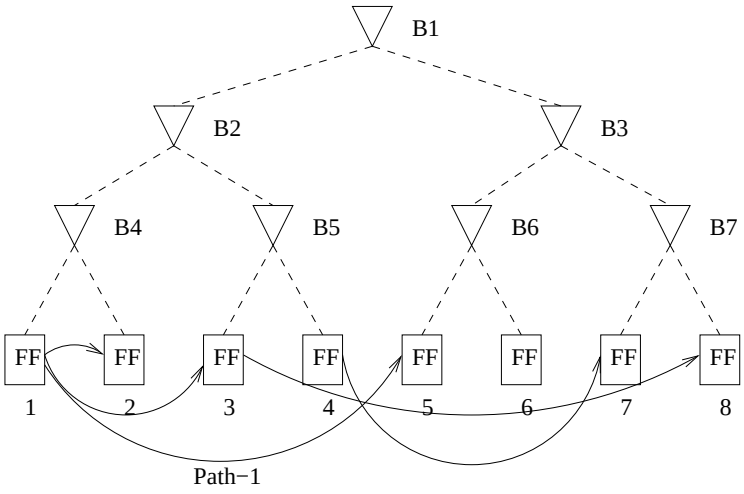


Figure 4.2: Sequential Design with a PST Clock Tree [72]

tunability might make the slack on some other path to become critical. So we need to ensure that all paths satisfy the timing constraints while using these tuning buffers to fix path timing violations. Also, it can be noted here that by adding any tuning to buffer $B1$ would not help fix this timing violation, since both FFs are equally effected by tuning $B1$. This example illustrates the mechanism by which PST buffers can be used to redistribute path slack between critical and non-critical paths to ensure a higher timing yield.

In [114, 71], the authors have proposed a design for PST buffers using passive loads and inverters. The final tuning can be done by connecting the required number of passive banks through a programmable pass bit at each bank. This design provides tuning proportional to its RC delay which in turn corresponds to its bank size and silicon area. There is a hardware and power overhead that is associated with implementing a PST clock tree. The hardware cost is reflective of the silicon area overhead which is proportional to both the number of tunable buffers and their

respective tuning ranges (which decides the passive load bank and inverters that are used). There is also a cost associated with the actual tuning delay used at each buffer (indicative of the clock tree power overhead). Thus it is important to compute the tuning range at each buffer such that the maximum timing yield improvements can be achieved without having wasted passive load banks at the PST buffers. We define tunability cost (TC) as a metric of the overhead of having these passive load banks and inverters in the PST tree. As explained, this overhead is in terms of both silicon area and power and is proportional to the range of the tuning buffers. In this work, TC is also an optimization objective.

$$TC = \sum_{k \in PST-buffers} R_k^{max} \tag{4.10}$$

where R_k^{max} is the tuning range allocated to PST buffer k.

4.1.3 Simultaneous Gate Sizing and PST Buffer Range Determination for Minimizing BYL and TC

In this work, we address the following problem:

Given a sequential design with a synthesized PST clock-tree (with known tunable buffer locations), we perform simultaneous gate sizing of the combinational logic gates and tuning range determination of each PST buffer, such that a combined objective function of the binning yield loss and tunability cost is minimized.

In this section, we first develop the formulation considering only longest path

constraints and prove it to be optimally solvable. Later, we will extend the formulation to consider shortest path constraints as well.

Motivation

The existing work in literature does not consider performing simultaneous gate sizing and PST buffer range determination in an integrated framework. The work proposed in [72] assumes that the longest delay path (with delay D_{ij}) between every pair of FFs (i, j) is given. These paths are prone to path delay variations and hence can cause timing violations. They present algorithms that use statistical timing analysis (STA) on these combinational paths to determine the location of PST buffers and their appropriate tuning ranges. They do not try to optimize the combinational paths themselves for better timing yield. In this work, we present an optimization framework that not only determines the PST buffer ranges (given the PST clock tree with tuning buffers placed), but also tries to size the combinational gates such that BYL and TC is minimized.

Moreover, we prove that the simultaneous gate sizing and PST buffer range determination problem under longest path delay constraints is convex and can be solved optimally. We extend that formulation to a heuristic considering shortest path constraints as well. This is the first work to propose such a unified framework. We make experimental comparisons using nominal gate sizing followed by PST buffer management using [72] as a base-case. We take the solution obtained from this approach and perform 1) Sensitivity-based statistical gate-sizing while retaining the

PST clock tree 2) Simultaneous gate sizing and PST buffer range determination as proposed in this work.

Let us first try to understand the intuition behind the potential benefits that can be derived from our scheme as opposed to performing a variability-aware gate sizing followed by PST clock-tree synthesis. In a variability-aware gate sizing framework, the objective is to minimize the likelihood of timing violation at each primary output. In order to meet the timing constraints as much as possible, the optimization engine might be forced to aggressively size the most critical paths, such that the area and power overhead incurred can be potentially very high. At this stage, the optimization engine does not leverage the information that PST buffers could have been used to meet the timing constraints along these paths and the cost of aggressive sizing could be saved. Also, the gate-sizing algorithm would try to reduce the sensitivity to process variations on each path (in order to maximize the chances of meeting the constraints). However, depending on the PST clock-tree structure, it might be cheaper (in terms of silicon area and total power) to maintain some paths to be more sensitive than others (and let PST buffers satisfy the timing constraints there). On the PST clock tree synthesis side of things, simultaneous gate sizing and PST buffer range determination allows for more accurate computation of the required buffer ranges. As mentioned before, each passive load bank amounts to an area and power overhead. We propose an integrated algorithm in this work is best able to leverage the trade-off between overall area and power while attempting to minimize the BYL and TC.

Effect of Variability on Gate Sizing

Process variations cause significant spread in circuit parameters like L_{eff}, t_{ox} and V_{th}. These in turn make the gate delays unpredictable. Typically, the circuit parameters that are affected by variations can be treated as random variables, making gate delay a function of these random variables. Let us denote the random vector denoting all the variable circuit parameters be $\vec{\Omega}$ where each parameter can have its own density function and these can be correlated in arbitrary ways. Thus, the coefficients in the gate delay model presented in equation (4.4) would now become a function of the underlying random field. In this light, the delay of gate i in the convex gate sizing formulation (as presented in equation (4.5)) also becomes a random variable and can be denoted as:

$$d_i(\vec{y}, \vec{\Omega}) = a_{0i}(\vec{\Omega}) + a_{1i}(\vec{\Omega})\frac{\sum_{\forall j} e^{y_j}}{e^{y_i}} \quad j \in fanout(i) \tag{4.11}$$

In presence of process variability, we can therefore redefine the objective of gate sizing to be BYL minimization. Let us suppose that $\vec{\omega}$ represents the nominal values of each of the varying parameters. We attempt to perform gate sizing at these nominal parameter values such that the BYL is minimized. This problem can be formulated as:

$$\begin{gathered} Minimize \;\; BYL(\vec{y}) \\ Subject \;\; to: \begin{cases} t_j(z_j) + d_i(\vec{y}) \leq t_i(z_i) \;\; \forall j \in fanin(i) \\ t_i(z_i) \leq T_{cons} \;\; \forall i \in PO \\ x^i_{min} \leq e^{y_i} \leq x^i_{max} \quad \forall gate \;\; i \end{cases} \end{gathered} \tag{4.12}$$

We try to meet the timing constraint T_{cons} at these nominal parameter values. Additionally, in order to control the total sizing area, we could add a constraint $\sum c_i \times e^{y_i} \leq Area_{max}$ to the formulation above. The gate sizing formulation used in this work is similar to that proposed in [8].

PST Clock Tree Structure and Assumptions

In this work, we assume that we are given a synthesized PST clock tree as well as the location of the tunable clock buffers. We do not make any assumption about the structure of the clock tree (it can be balanced or unbalanced), clock skews or the location of the tunable buffers. We use the PST clock tree alongwith the buffer locations obtained from [72] as an input to our algorithm.

Problem Formulation

The simultaneous gate sizing and PST buffer range determination problem can be formulated as a Two-Stage *Stochastic Program* [93]. The sequential design can be viewed as a set of FFs and logic gates. Each pair of FFs can share a combinational logic path between them. Each such path needs to meet the timing constraint in order to make the design feasible.

For every pair of FFs i, j that are connected through combinational logic, we define a variable D_{ij} that represents the delay of the longest path between them. We can compute D_{ij} using the inequalities similar to that in the gate sizing formulation on the combinational logic between these two FFs:

$$
\begin{aligned}
t_p(z_p) + d_q(\vec{y}) &\leq t_q(z_q) \quad \forall p \in fanin(q) \\
t_q(z_q) &\leq D_{ij} \quad q\ is\ fanin\ of\ FF:j \\
x^q_{min} \leq e^{y_q} &\leq x^q_{max} \quad \forall gate\ \ q
\end{aligned}
\tag{4.13}
$$

For each pair of FFs i,j, we can write the constraints mentioned above through inequalities (4.13) and compute the longest path delay D_{ij}.

1. **Variables of Interest**

   There are three sets of variables in the problem formulation. The first set are the gate-size variables represented by $\vec{y}$, where the size of gate i is given by e^{y_i}. The second set of variables represented by $\vec{r}$, where the tuning buffer ranges for each PST buffer i is given by e^{r_i}. The third set of variables are represented by $\vec{z_i}$, where the arrival time at each gate i is given by e^{z_i}.

2. **Objective of Interest**

   A general objective function can be to minimize a combination of BYL, TC (which is representative of the area and power overhead incurred in PST clock tree) and also the total gate-size (similar to traditional gate-sizing problem). Since the tuning range at each PST buffer is proportional to the area and power overhead, TC can be represented by the sum of the total range of all PST tuning buffers. Hence, a general objective function of interest could be written as:

$$
Minimize \quad (BYL(\vec{y},\vec{r})) + TC(\vec{r}) + \sum_i Gate - Sizes) \tag{4.14}
$$

$$Minimize \quad (BYL(\vec{y},\vec{r})) + \sum_k \alpha_k e^{r_k} + \sum_i \beta_i e^{y_i}) \tag{4.15}$$

BYL is a function of both $(\vec{y},\vec{r})$ as explained later. This objective function allows to explore the trade-off between BYL, TC and the total gate-size area by appropriately scaling the constants $\vec{\alpha}$ and $\vec{\beta}$.

Two-Stage Stochastic Program

The first stage of the problem formulation can be written in general form as:

$$Minimize \;\; (BYL(\vec{y},\vec{r}) + \textstyle\sum_k \alpha_k e^{r_k} + \sum_i \beta_i e^{y_i})$$

Subject to:

$$\begin{cases} \left.\begin{array}{l} T_i + D_{ij} \le T_{clk} + T_j - T^j_{set} \;\; \forall FFs(i,j) \\ t_p(z_p) + d_q(\vec{y}) \le t_q(z_q) \;\; \forall p \in fanin(q) \\ t_q(z_q) \le D_{ij} \qquad q \; is \; fanin \; of \; FF:j \end{array}\right\} \forall FFs(i,j) \\ x^q_{min} \le e^{y_q} \le x^q_{max} \qquad \forall gate \;\; q \\ \textstyle\sum_k e^{y_k} \le X_{max} \qquad \forall gate \;\; k \\ 0 \le e^{r_m} \le R^{max}_m \qquad \forall m \in PST \; Buffer \\ \textstyle\sum_m e^{r_m} \le Range^{max} \qquad \forall m \in PST \; Buffer \end{cases} \tag{4.16}$$

Let us try to understand the constraints in the above formulation. The first constraint in inequalities of (4.16) represents the longest-path constraint (equation (4.6)) between each pair of FFs that share a path between them. Here, T_i, T_j, T_{clk} and T^j_{set} are known constants that correspond to clock arrival times. The longest

path delay D_{ij} can be determined from the next three inequalities that represent the gate sizing formulation for the logic paths between FFs i and j. We note that a gate can show on multiple paths, hence there would be several such sizing constraints on each gate. But since the first stage problem considers all these constraints together, there is no discrepancy that can come in. The total sum of gate sizes for the design can be bounded to be less than a constant X_{max} using inequality 5 above. Each PST buffer m can be bound to have a maximum allowed tuning range R_m^{max}. In order to limit the total tunability cost, we can also have a bound on the total cumulative tuning range given by $Range^{max}$. These are represented by the last two inequalities. This is the most general form of the first stage problem.

In presence of process variability, the delay between each pair of FFs i-j that have a combinational logic path between them, becomes a random variable that can be represented as $D_{ij}(\vec{y}, \vec{r}, \vec{\Omega})$ that depends on the gate-sizes $\vec{y}$, the tuning buffer range $\vec{r}$ and the random field due to process variations $\vec{\Omega}$ (that may have some correlation between its components). Let us define a random variable P that denotes the penalty of violating the timing constraint (T_{clk}) as:

$$P(\vec{y}, \vec{r}, \vec{\Omega}) = \begin{cases} q(D_{ij}(\vec{y}, \vec{r}, \vec{\Omega}) - T_{cons}); & D_{ij} \geq T_{cons} \\ 0; & otherwise \end{cases} \tag{4.17}$$

where $q(.)$ is the convex penalty function that was defined in equation (4.1).

In equation (4.2), BYL was defined as the expected value of the timing-violation penalty. For a given $(\vec{y}, \vec{r})$ and a sample ω of the random field Ω, let $p(\vec{y}, \vec{r}, \vec{\omega})$ be the value of the random variable P. By definition, $p(\vec{y}, \vec{r}, \vec{\omega})$ denotes

the timing-violation penalty for a given $(\vec{y}, \vec{r})$ at that variability sample ω. Hence, BYL would be the average timing-violation penalty over all such samples ω which is the expected value of the random variable P for a given $(\vec{y}, \vec{r})$. Therefore:

$$BYL(\vec{y}, \vec{r}) = E[P(\vec{y}, \vec{r}, \vec{\Omega})] \tag{4.18}$$

We can evaluate the timing-violation penalty $p(\vec{y}, \vec{r}, \vec{\omega})$ given a fixed $\vec{y}$, $\vec{r}$ and a variability sample $\vec{\omega}$ through another convex formulation that can be written as:

$$p(\vec{y}, \vec{r}, \vec{\omega}) = Minimize \quad \sum_{FF(i,j)} q(T_{ij}^{viol_s})$$

Subject to:

$$\begin{cases} (T_i + \sum_{k \in C_i} T_k^{Buf}) + D_{ij}(\vec{y}, \vec{r}, \vec{\omega}) \leq T_{clk} + \\ (T_j + \sum_{k \in C_j} T_k^{Buf}) - T_{set}^j + T_{ij}^{viol_s} \quad \forall FFs(i,j) \\ \left.\begin{array}{l} t_p + d_q(\vec{y}, \vec{\omega}) \leq t_q \quad \forall p \in fanin(q) \\ t_q \leq D_{ij}(\vec{y}, \vec{r}, \vec{\omega}) \qquad q \; is \; fanin \; of \; FF : j \end{array}\right\} \forall FFs(i,j) \\ T_{ij}^{viol_s} \geq 0 \; \forall FFs(i,j) \\ 0 \leq T_k^{Buf} \leq e^{r_k} \; \forall k \in PST \; Buffer \end{cases} \tag{4.19}$$

Let us try to understand this formulation. Given a value of $\vec{y}$, $\vec{r}$ and a variability sample $\vec{\omega}$ implies that the delay of each gate i ($d_i(\vec{y}, \vec{\omega})$) is known. Also, since $\vec{r}$ is given, the range of each tuning buffer k is will be e^{r_k}. As mentioned before in subsection 4.1.2, T_i and T_j are the clock arrival times at FFs i and j respectively and are known values. For each FF i, we know the set of tuning buffers C_i that can affect the clock arrival time at this FF. In the above formulation, the problem variables are T_k^{Buf} which is the actual tuning at PST buffer k that is used to reduce the

timing violation. The longest path delay D_{ij} for each pair of FFs (i, j) is a variable and the arrival time t_i at each gate i is a variable. Additionally, we define a variable T_{ij}^{viol} for each pair of FFs (i, j) that represents the timing violation along the longest path between those FFs. The timing-violation penalty at each FF pair (i, j) can be computed as $q(T_{ij}^{viol_s})$. The objective of this problem is to minimize the sum of timing-violation penalty across all pairs of FFs (i, j) by appropriately assigning delay tuning to each PST buffer within the range given by the variables $\vec{r}$. Essentially, this formulation tries to determine the best combination of tuning set ($T^{\vec{Buf}}$) that should be applied at the PST buffers such that the total timing-violation penalty for the design is minimized.

For a given value of $\vec{y}$, $\vec{r}$, the optimal objective to this formulation gives us $p(\vec{y}, \vec{r}, \vec{\omega})$ which is the desired quantity to compute BYL$(\vec{y}, \vec{r})$.

The two formulations defined by inequalities (4.16) and (4.19) form a classic Two-Stage Stochastic Programming formulation [93], where the former is called the first-stage problem and the latter second-stage problem. We would like to point out that even though the proposed formulation considers clock arrival times (T_i, T_j) to be constant, our formulation can be extended to consider uncertainty in clock tree as well. In that case, the second stage formulation would consider the clock arrival times $(T_i(\vec{\omega}), T_j(\vec{\omega}))$ to be dependent on the randomness (Ω).

The problem formulation is convex in ($\vec{y}$,$\vec{r}$)

Theorem: The proposed two-stage stochastic programming formulation is convex.

Proof: In order to show that the two stage stochastic programming is convex, we need to show that the objective and constraints of both the stages are convex in the decisions variables ($\vec{y}$,$\vec{r}$). Let us first look at the first stage formulation from equations 4.16. The first three constraints as shown below represent the timing constraints on the design.

$$\begin{aligned} & T_i + D_{ij} \leq T_{clk} + T_j - T_{set}^j \;\; \forall FFs(i,j) \\ & \left.\begin{aligned} t_p(z_p) + d_q(\vec{y}) \leq t_q(z_q) \;\; \forall p \in fanin(q) \\ t_q(z_q) \leq D_{ij} \qquad q \; is \; fanin \; of \; FF:j \end{aligned}\right\} \forall FFs(i,j) \end{aligned} \tag{4.20}$$

The gate delay variable ($d_q(\vec{y})$) is defined to be convex in the decision variables $\vec{y}$, thereby making these constraints convex. The remaining constraints are also convex in decision variables ($\vec{y}, \vec{r}$). The objective function has three sets of terms, the first one BYL($\vec{y}, \vec{r}$) is the expected value of a random variable $P(\vec{y}, \vec{r}, \vec{\Omega})$ that depends only on variables ($\vec{y}, \vec{r}, \vec{\omega}$). The second term is $\sum_k \alpha_k e^{r_k}$ which is convex and the third term is $\sum_i \beta_i e^{y_i}$ which is convex as well (constants $\vec{\alpha}$ and $\vec{\beta}$ are positive). If we can show that $BYL(\vec{y}, \vec{r})$ is convex in ($\vec{y}, \vec{r}$), the first stage problem will be a convex formulation.

We have defined $BYL(\vec{y}, \vec{r}) = E[P(\vec{y}, \vec{r}, \vec{\Omega})]$. Hence, BYL can be interpreted to be a weighted sum of timing-violation penalty $p(\vec{y}, \vec{r}, \vec{\omega})$ at each sample ω of the random field Ω, where the weights are positive. Hence, if we can prove that each

of $p(\vec{y}, \vec{r}, \vec{\omega})$ is convex in $(\vec{y}, \vec{r})$, then BYL would also be convex since it is a sum of positively weighted convex functions which is convex by definition. From the second-stage formulation as given by inequalities in (4.19), $p(\vec{y}, \vec{r}, \vec{\omega})$ is equal to the objective $\sum_{FF(i,j)} q(T_{ij}^{viol_s})$. Each of $q(T_{ij}^{viol_s})$ is positive and convex in $T_{ij}^{viol_s}$, hence it is sufficient to show that $T_{ij}^{viol_s}$ is a convex function of $(\vec{y}, \vec{r})$. From the constraints in the second stage formulation, we can see that the dependence of $T_{ij}^{viol_s}$ on the decision variables is remains convex as gate-delay $d_i(\vec{y}, \vec{\omega})$ and tuning buffer ranges e^{r_k} are convex in $(\vec{y}, \vec{r})$. Hence, $T_{ij}^{viol_s}$ is convex in $(\vec{y}, \vec{r})$. This implies that $BYL(\vec{y}, \vec{r})$ is convex and hence the proposed two-stage stochastic programming formulation is convex.

We would like to point out that the proposed two-stage stochastic programming formulation and its proof convexity does not make any assumption about the distributions of the randomness (Ω) and the correlations between its different components.

4.1.4 Shortest Path Delay Constraints

The formulation discussed in the earlier sections presents a provably optimal technique considering only longest path (setup time) constraints. However, for a pair of FFs i and j, we also need to satisfy the shortest path (hold time) constraints. Given the shortest path delay D_{ij}^{short} between the two FFs, we can write the shortest path delay constraint as:

$$T_i + D_{ij}^{short} \geq T_j + T_{hold}^{j} \quad \forall FFs(i,j) \tag{4.21}$$

where T_{hold}^{j} is a constant denoting the hold-time for FF j, T_i and T_j are clock arrival times. As can be seen, this is a non-convex constraint considering the convex gate delay models given by equation 4.11. Hence, considering shortest path constraints in the formulation proposed in the earlier section would break the convex nature of the problem. We will now present an efficient heuristic to consider the shortest path constraints in our formulation while preserving its convexity.

Let us suppose that we are given p paths which are candidates for shortest path delay violation (can be determined from static timing analysis). The cumulative delay of the gates on each of these paths would give us the delay of the path. We will make a linear approximation on the gate delay model for these gates wrt the gate sizing variable. Given a gate m (with size e^{y_m}) and its fanout gate n (with size e^{y_n}), we can approximate its gate delay as a linear function of the sizing variables (y). This model is constructed such that it is a lower bound to the convex gate delay model given by equation 4.4. Therefore, the shortest path delay is under-predicted by our linear gate delay model approximation and any valid solution will always satisfy the shortest path delay constraint. Let us suppose that the path delay of the pth shortest path is denoted by $D_{ij}^{short_p}$, we can compute the linear gate delay and the shortest path delay as:

$$D_{ij}^{short_p} = \sum_{m} d_m^{lin} \quad \forall gates\ m\ on\ path\ p \tag{4.22}$$

$$d_m^{lin} = a_{0m} + a_{1m} y_m + \sum_{\forall fanout-n} b_n y_n \tag{4.23}$$

where a_0, a_1 and b_n are constants. Under these assumptions, it can be seen that the shortest path constraint as given by equation 4.21 is now convex and can be added to our proposed formulation without breaking the convex nature of the problem.

Let us now understand, how we can extend the two-stage stochastic programming formulation to also consider shortest path delay constraints. Given the p paths which are candidates for shortest path delay violation, the first stage formulation as given by equations 4.16 can be modified to additionally consider the constraint:

$$T_i + D_{ij}^{short_p} \geq T_j + T_{hold}^j \quad \forall paths\ p \quad \forall FFs(i,j) \tag{4.24}$$

where $D_{ij}^{short_p}$ is defined using equations 4.22 and 4.23.

The BYL will now consists of both longest path delay violation and shortest path delay violation. The second stage problem given by equations 4.19 can be modified to consider the BYL due to shortest path delay violation. The timing violation penalty can now be computed as:

$$p(\vec{y}, \vec{r}, \vec{\omega}) = Minimize \sum_{FF(i,j)} q(T_{ij}^{viol_s}, T_{ij}^{viol_h}) \tag{4.25}$$

where $T_{ij}^{viol_s}$ represents the timing violation in the longest path constraints and $T_{ij}^{viol_h}$ represents the timing violation in the shortest path constraints. The second stage formulation can consider additional constraints for shortest path delay violation as given by:

$$(T_i + \sum_{k \in C_i} T_k^{Buf}) + D_{ij}^{short_p}(\vec{y}, \vec{r}, \vec{\omega}) + T_{ij}^{viol_h} \geq (T_j + \sum_{k \in C_j} T_k^{Buf}) + T_{hold}^{j} \;\; \forall paths\ p \;\; \forall FFs(i,j)$$

$$T_{ij}^{viol_h} \geq 0 \;\; \forall FFs(i,j) \tag{4.26}$$

where T_k^{Buf} is the tunable delay introduced due to PST buffer k. This constraint gives us the timing violation in the shortest path constraint $T_{ij}^{viol_h}$ for path p.

This completes the extension of the two-stage stochastic programming formulation to consider the shortest path constraints in addition to the longest path constraints. Although, we preserve the convex nature of the formulation, the error introduced due to the lower bounding linear approximation on the gate delay models for shortest path constraints makes this a heuristic technique.

4.1.5 Solving the Two-Stage Stochastic Program

In this work, we have used Kelley's Cutting Plane Method [99] to solve the two-stage stochastic programming formulation. *We would like to point out that this is just one technique that can be applied to solve this convex formulation. Any other convex optimization scheme can be used as well.* We will now briefly describe how we can use this technique to solve the convex formulation discussed in the previous section.

Algorithm 3 Kelley's Cutting Plane Algorithm

Step 1: *Initialize*

Let $\epsilon > 0$, $k \leftarrow 0$ and define $l_0(\vec{x}) = -\infty$, $u_0(\vec{x}) = \infty$.

Step 2: Compute a feasible solution, $\vec{x}_k$, satisfying the constraints.

Step 3: *Set* $k \leftarrow k + 1$

Step 4: *Define the Lower Bound at* $\vec{x}_k$

Evaluate α_k and $\vec{\beta}_k$ such that $l_k \geq \alpha_k + < \vec{\beta}_k, \vec{x} >$:

$$\alpha_k = BYL(\vec{x}_k) - \vec{\beta}_k \vec{x}_k \qquad \vec{\beta}_k = \frac{\partial BYL(\vec{x})}{\partial \vec{x}}|_{\vec{x}_{k-1}}$$

Step 5: *Update the Optimization Set*

Add the following to the existing set of constraints:

$$l_k \geq l_{k-1} \qquad l_k \geq \alpha_k + < \vec{\beta}_k, \vec{x} >$$

Update the objective function to $Minimize\ l_k$.

Step 6: *Solve the Optimization to get* $\vec{x}_k$ *and Update the Bounds*

Let upper bound $u_k = Min\{u_{k-1}, BYL(\vec{x}_k)\}$ and lower bound l_k.

Step 7: *Stopping Rule*

Stop if $u_k - l_k \leq \epsilon$, otherwise go to Step 2.

Kelley's Cutting Plane Algorithm

Kelley's Algorithm is an iterative approach that solves the first-stage formulation and then uses the solution $(\vec{y}, \vec{r})$ obtained to generate a lower bound to the BYL$(\vec{y}, \vec{r})$ from the second-stage formulation. In the next iteration, this lower bound is used to guide the first-stage problem to a new solution to get a better BYL$(\vec{y}, \vec{r})$ estimate. Hence, in each iteration we add a linear lower bound to BYL$(\vec{y}, \vec{r})$ and subsequently converges to the optimal value.

The overall algorithm for the Cutting Plane Method can be summarized in Algorithm-1. Let us suppose that the variables in the problem are defined as $\vec{x}$ which in our case is a vector constituting the gate-size variables $\vec{y}$ and the PST buffer range variables $\vec{r}$. At each iteration k, the lower bound found from the previous iteration is used to find a new solution $\vec{x_k}$. The lower bound is actually the sub-gradient of the objective (BYL) at the previous solution $\vec{x_{k-1}}$. We find the lower bounding co-efficients (α_k,$\vec{\beta_k}$) such that BYL$(\vec{x_{k-1}}) = \alpha_k + \vec{\beta_k}\vec{x_{k-1}}$. This sub-gradient forms a lower bound to the objective BYL function. We use this new lower bound value l_k as shown in the algorithm to represent the current value of the BYL function. In the next iteration, we get a new solution $\vec{x_k}$. At each iteration, we also get an upper bound to the BYL function that is BYL$(\vec{x_k})$. In each iteration, the upper and lower bound come closer and final converge to the optimal solution. Since this is a convex formulation, this technique gives us the optimal solution [99].

Computing the lower bound to BYL

This step is the most critical step in Kelley's algorithm, since the lower bound generated is used to drive the algorithm towards the optimal solution. Given a solution $\vec{x_{k-1}}$, (note that $\vec{x}$ represents both the sizing variables $\vec{y}$ and PST buffer range variables $\vec{r}$ from our formulation) we need to compute the sub-gradient to the BYL($\vec{x}$) function at this solution point $\vec{x_{k-1}}$. The lower bound is expressed as $\alpha_k + < \vec{\beta_k}, \vec{x} >$. Let us understand how we can compute $\vec{\beta_k}$. We use the method of finite differences in our work to compute the sub-gradient. Any other sub-gradient estimation technique can be used as well. Each component β_i is defined as $\frac{\partial BYL(\vec{x})}{\partial \vec{x_i}}|_{\vec{x}_{k-1}}$ and represent the sensitivity of the BYL function to the variable x_i at the current solution x_{k-1}. This can be computed by incrementing the variable x_i by a small quantity Δx_i and then computing the change in BYL per unit change in x_i. Mathematically, this can be represented as:

$$\beta_i = \frac{BYL(\{x_1; ...x_i + \Delta x_i; ...; x_n\}) - BYL(\{x_1; ...x_i; ...; x_n\})}{\Delta x_i}|_{\vec{x}_{k-1}} \tag{4.27}$$

Once $\vec{\beta}$ have been computed, it is fairly easy to compute α through the relation BYL($\vec{x_{k-1}}$) = $\alpha_k + \vec{\beta_k}\vec{x_{k-1}}$.

Let us try to understand how we can compute BYL($\vec{x}$), since this is a very important step in generating the lower bound. At a given solution of the first stage problem, i.e. ($\vec{y}$,$\vec{r}$), computing the BYL($\vec{y}$,$\vec{r}$) amounts to estimating the expected value of the timing-violation penalty $P(\vec{y}, \vec{r}, \vec{\Omega})$. In the scenario when there are no PST clock buffers in the design, the problem of computing the timing-violation

penalty would amount to computing the timing pdf that can be done using STA technique ([42, 127, 119, 43]). But in our case, we also have PST clock buffers, where the amount of tuning required at each buffer for best timing yield would vary depending on each variability sample ω. To our best knowledge, there are no current STA techniques that can handle timing analysis in presence of PST clock buffers.

Consequently, in this work we resort to using a Monte-Carlo based STA technique were for each sample ω of the random field, we formulate the second-stage problem as proposed using inequalities (4.19) and compute the actual timing-violation penalty $p(\vec{y}, \vec{r}, \vec{\omega})$. This is repeated for every variability sample ω such that the expected value of timing-violation penalty which equals BYL($\vec{y}$,$\vec{r}$) is eventually computed. We note here that since we need to generate each β_i once at a time, this STA process is repeated for every variable $\vec{y}$ and $\vec{r}$. It is easy to note that this step becomes a major bottleneck in the performance of our algorithm and makes the entire computation slow.

However, the proposed algorithm is free to use any efficient STA technique that can predict timing pdf in presence of PST clock buffers. In the future, when such a STA technique has been developed, it can be plugged into the proposed algorithm. In our results section, we will show that almost all the computational time for our algorithm goes into this Monte-Carlo based STA computation.

4.1.6 Experimental Results

The overall formulation considering shortest and longest path constraints was implemented in SIS [37]. We performed experiments on the ISCAS benchmark suite. We generated a valid placement for each benchmark using $CAPO$. The correlation information between gates was generated using the model proposed in [42]. We assumed that process variability caused threshold voltage to have a Gaussian distribution with a mean value of $0.2V$ and a standard deviation of 15% from the mean. We used 90nm technology parameters (from [125]) to compute the coefficients of the convex gate delay expression (as a function of its size) as given by equation (4.4). The PST clock tree structure used in our experiments is obtained using the algorithm proposed in [72]. Each PST buffer was allowed to have a maximum tuning delay of 5 psec.

In order to solve the first-stage convex formulation, we integrated MOSEK [49] with SIS. The formulation proposed in section 4.1.5 was also implemented in SIS. As mentioned in that section, we implemented a Monte-Carlo based STA scheme to compute the BYL during each iteration of the cutting plane algorithm. In figure 4.3, we can see that the upper bound (objective) representing the BYL at the current solution improves in each iteration and quickly converges to the lower bound.

There is no scheme in the literature that does simultaneous gate sizing and PST buffer management. We have run three set of experiments to evaluate our algorithm:

1. A nominal gate sizing scheme followed by PST buffer management as proposed

in [72]: We first run gate sizing assuming nominal process parameter values. On this solution, we perform PST buffer management (location and tuning range determination of each PST buffer) using the algorithm proposed in [72].

2. Taking the solution from experiment 1 ([72]), we retain the PST clock buffer structure (location and ranges) but try to re-optimize the design using a sensitivity-based statistical gate-sizing approach similar in spirit to that proposed in [32, 4]: This approach is an iterative scheme where at each step, we evaluate the BYL improvements that can be achieved per unit size increase for each gate. The most sensitive gate is chosen as the next gate to be upsized.

3. Taking the solution from experiment 1 ([72]), we retain only the locations of the PST clock buffers and run our simultaneous gate sizing and PST buffer range determination algorithm: The PST clock tree obtained in experiment 1 is taken as an input, though we reallocate the range of each of the PST buffers while performing gate sizing as proposed in this work.

The aim of these experiments is to show that our proposed algorithm can provide significant improvements over the design obtained from [72]. Furthermore, comparison with experiment 2 shows that the simultaneous gate sizing and PST buffer range determination algorithm proposed in this work is significantly more effective than performing a statistical resizing of the design.

In order to compute the BYL for each experiment, we impose the process parameter variations (Ω) on the final design solution through monte-carlo simulation and compute the minimal timing violation considering tunability for each sample

bench	T_{cons}	[72]			[72] + Sensitivity			[72] + Convex Stochastic		
name	$(psec)$	BYL	$Area$	$Buf.Range$	BYL	$Area$	$Buf.Range$	BYL	$Area$	$Buf.Range$
s27	450	4165	402	9	3293	403	9	4	418	16
s298	700	30854	4135	4	28477	4187	4	414	4146	5
s344	1000	38850	3822	3	14289	4006	3	377	3838	3
s382	700	54916	5073	6	95364	5273	6	116	5162	7
s400	850	71823	5370	3	61400	5441	3	1638	5418	8
s499	1350	232523	6614	15	260749	6706	15	8766	6714	17
s526	900	58750	8091	8	2210	8152	8	568	8133	10
s635	2500	253551	7730	3	111368	7853	3	1308	7784	1

Table 4.1: Comparison of Binning Yield-Loss, Area and Total PST Buffer Range in (psec)

(ω). The average BYL over all ω was taken as the BYL for the design.

Table 4.1 compares the three approaches in terms of the BYL, the total area after gate sizing and the tuning buffer range. We can see that our proposed convex-stochastic approach resulted in significantly lower BYL compared to the other two cases. Since the nominal gate sizing is not variability-aware, experiment 1 resulted in the highest BYL. On an average, the BYL obtained from our approach is 98% lower than the solution from experiment 1 ([72]) and 95% lower than experiment 2, the sensitivity-based algorithm.

The final gate-size area obtained for our approach is on an average 1.25% lower than that obtained from experiment 1 (nominal gate sizing followed by [72]) and 0.62% higher than experiment 2, the sensitivity approach. Hence, the convex-stochastic algorithm gives better BYL for similar total gate-size area. From figure 4.4, we can see that our approach gives much lower BYL for the same total gate-size area as compared to the sensitivity-based algorithm.

The total PST tuning buffer range that is allocated through the proposed

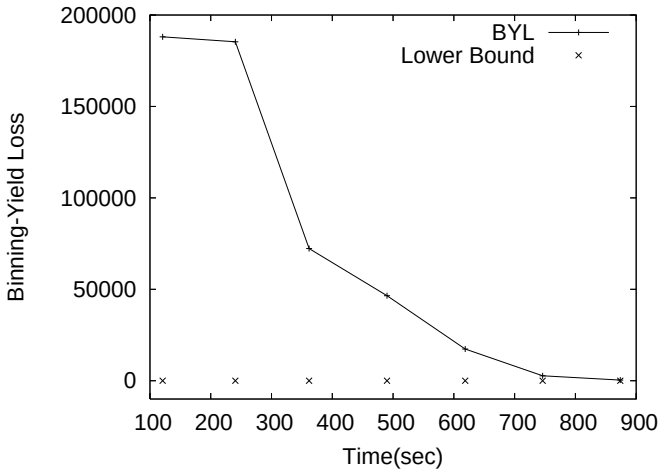


Figure 4.3: Convergence of BYL to its lower bound with time for s344

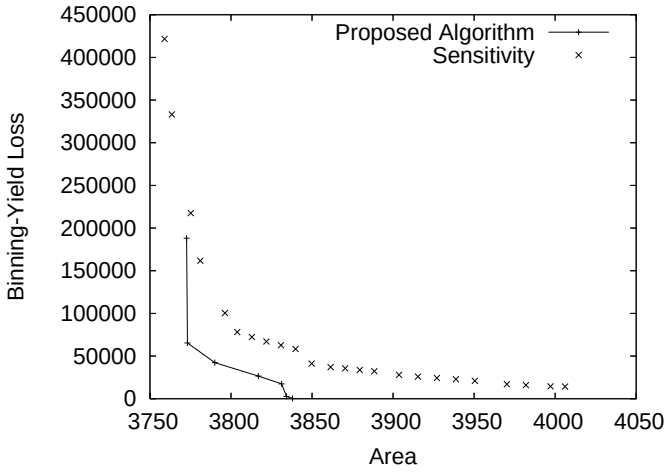


Figure 4.4: BYL vs. Area Generated at Different Iterations of Kelley's and Sensitivity-Based Algorithms

bench	T_{cons}	[72]	[72] + Sensitivity	[72] + Convex Stochastic
s27	450	0.23	0.18	0.03
s298	700	0.16	0.11	0.02
s344	1000	0.24	0.15	0.03
s382	700	0.16	0.11	0.02
s400	850	0.26	0.18	0.05
s499	1350	0.24	0.26	0.03
s526	900	0.26	0.09	0.02
s635	2500	0.17	0.12	0.05
Average		0.22	0.19	0.03

Table 4.2: Comparison of Yield-Loss

algorithm is comparable to that obtained from [72]. Hence, our algorithm is able to identify PST buffer ranges that result in BYL reduction without putting any additional overhead in terms of PST buffer cost while performing simultaneous gate sizing.

Table 4.2 reports the traditional timing YL that were obtained for the solutions from all three approaches. It can be seen that on an average the nominal-sizing followed by [72] gave 22% yield-loss, while the sensitivity approach gave 19% yield-loss whereas our proposed algorithm gave only 3% yield-loss. These results show that even though we do not directly optimize for timing yield loss (we optimize BYL), we get better and more robust design solutions.

From figure 4.5, it is evident that the convex stochastic algorithm has a much faster rate of convergence than the sensitivity-based algorithm. The runtimes for

bench	T_{cons}	Sensitivity		Convex Stochastic		Speedup
		#itera.	time	#itera.	time	
s27	450	14	0.5	10	0.9	0.6
s298	700	16	13.7	9	11.6	1.2
s344	1000	24	24.3	7	14.6	1.7
s382	700	40	53.9	19	41.3	1.3
s400	850	18	28.3	13	19.5	1.5
s499	1350	35	87.1	19	72.2	1.2
s526	900	15	52.0	14	40.1	1.3
s635	2500	109	378.0	7	43.3	8.7
Average						2.2x

Table 4.3: Comparison of Total Run-Time (min) and Number of Iterations

each benchmark are reported in table 4.3 alongwith the number of iterations. It can be observed that our approach converges to a better solution in fewer iterations and on an average is $2.2x$ faster than the sensitivity-based algorithm.

As pointed out earlier in this work, the maximum runtime in our approach is taken in computing the BYL using Monte-Carlo based STA. This is due to the fact that none of the current STA techniques are able to perform timing analysis considering tunability. Our proposed algorithm is independent of the STA algorithm used and can be used in combination with an efficient PST aware STA scheme developed in future. From table 4.4 it can be seen that almost 93% of the computational runtime goes into the STA process.

bench	Avg. Iter. Time	Avg. STA time / Iter	%
s27	5	4	80.0
s298	77	74	96.1
s344	125	117	93.6
s382	130	127	97.7
s400	90	85	94.4
s499	228	225	98.7
s526	172	165	95.9
s635	371	351	94.6
Average			93.8

Table 4.4: Contribution of Monte-Carlo Based STA time to Iteration Time (sec)

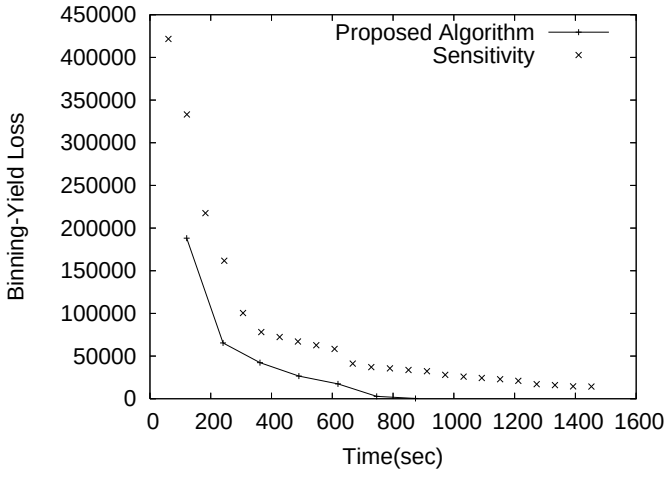


Figure 4.5: BYL vs. Time Generated at Different Iterations of Kelley's and Sensitivity-Based Algorithm

Chapter 5

Variability-Aware Design Optimization: Runtime Techniques

In this chapter, we will look at a design optimization technique that is a runtime dynamic power optimization scheme. Such schemes allow a designer to ensure that the design can adapt at runtime to reduce power overheads in the design. Runtime management of dynamic power through supply voltage scaling is one such technique that has been proposed in literature by several researchers [117, 30]. This technique utilizes the concept of having multiple supply voltages available on chip, and certain modules can be made to operate at lower or higher supply voltages without paying a penalty on the overall latency of the system. There is an overhead of adding extra hardware cost to provide the design with such a functionality, but for high-performance lower power designs these techniques provide a useful methodology for runtime optimization of performance.

In this work, we look at the impact of fabrication and environmental variability on such a runtime power optimization technique of dual-supply voltage allocation.

5.1 Simultaneous Resource Binding and Dual-Vdd Allocation for Power Optimization with Probabilistic Reliability Guarantee

In the nanometer regime, fabrication and environmental variations pose a potent problem to IC designers. The uncertainty introduced due to these variations cause a spread in design performance, rendering traditional design techniques inaccurate. A lot of recent research has been focused on statistical modeling of variations to consider this performance uncertainty during analysis and optimization [111, 112, 43, 119]. The reliability and robustness of designs is compromised due to these variations. Most of these research efforts have been focused on physical synthesis and very little work exists in high-level synthesis that tries to adapt to these uncertainties.

Low power resource binding is a very extensively researched problem [30, 29, 63, 64]. More recently [87, 86] have looked at the resource binding problem from a peak temperature control perspective. The authors in [104, 103, 102] have looked at the problem of power minimization during datapath scheduling and synthesis. The work in [108, 109] addresses the problem of reliability due to soft errors during high-level synthesis. In [47, 46], the authors addressed the problem of delay variability (from a worst-case bound perspective) during the resource binding problem. It this work, we focus on the problem of simultaneous resource binding and dual-Vdd allocation from a reliability perspective in presence of fabrication and environmental variability.

In presence of fabrication and environmental variability, the delay of func-

tional modules and the latency of clocks become non-deterministic random quantities. These uncertainties can potentially cause violation of the implicit deadlines imposed on a DFG that has been scheduled and bound (on resources) causing incorrect and unreliable operation. Traditionally, these issues could be handled using a worst-case analysis. It is known that the probability distributions of variability can be correlated in arbitrary ways, causing a worst-case approach to yield extremely pessimistic solutions [43, 119]. A probabilistic approach that captures these correlations is therefore imperative.

We address the following problem in this work:

Given a scheduled data flow graph (DFG) with resource and latency constraints and two supply voltage values Vdd_h and Vdd_l, the problem is to assign supply voltage values to each operation and perform resource binding of operations such that power is optimized while maintaining a reliability guarantee on the design. This work presents a novel probabilistic framework to model variability into the problem in order to provide a reliability guarantee (which is defined as an acceptable probability of failure) on the solution. To our best knowledge, this is the first work that attempts to incorporate fabrication and environmental variability explicitly during high-level synthesis in a probabilistic framework to provide reliability guarantees on the design. Our proposed formulation can be extended to consider leakage power optimization and soft error based reliability issues as well.

The main contributions of this work are:

1. We propose probabilistic modeling of fabrication and environmental variability in scheduled DFGs.

2. We propose the concept of *Reliability Guarantee* as a quantitative metric and present a polynomial time optimal algorithm to compute the same. Reliability guarantee implies a specified acceptable probability of failure. We note here that our technique can handle any distribution of variability which can be correlated in arbitrary ways.

3. We present a polynomial time optimal algorithm to solve the simultaneous resource binding and dual-Vdd allocation problem under reliability guarantee.

4. We present a polynomial time optimal algorithm to determine the *optimal* value of Vdd_l for the above problem.

5. We present an efficient rescheduling heuristic that performs local perturbations on the input schedule to provide additional power savings while maintaining the reliability guarantee.

We performed experiments to show that existing techniques that ignore variability [30] provide solutions that are extremely prone to failure. Design solutions obtained from our approach are reliable under variability. The power obtained through our scheme is on an average 3% higher than that obtained from [30]. Thus, the modeling framework and algorithm proposed in this work are able to provide designs that robust under variability without any significant overhead in power.

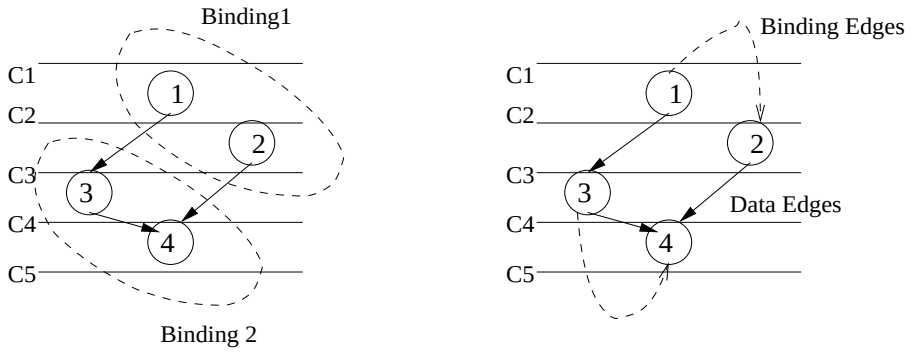


Figure 5.1: Reliability in a Scheduled and Bound DFG

5.1.1 Fabrication and Environmental Variability: Impact and Modeling

Consider the DFG in figure 5.1 which has been scheduled in 4 clock steps and bound on two resources. Implicitly, there are several deadlines imposed on the operations in this DFG. Operation 1 must finish its computation before clock edge C2 because it is bound on a functional module that needs to start processing operation 2 immediately afterwards. Operation 3 must finish processing before clock edge C4 because its data is needed by operation 4. As indicated on the figure these deadlines are indicated by the data and binding edges. Given a scheduled DFG that has been bound on functional resources, these implicit deadlines can be inferred. For example, operation 1 has two deadline constraints:

$$D_1 \leq C2 - C1 \tag{5.1}$$

$$D_1 \leq C3 - C1 \tag{5.2}$$

In presence of fabrication and environmental variability, the delay of functional

modules and the latency of clocks become non-deterministic random quantities. Environmental variability like thermal hotspots, noisy cross-coupling, supply voltage fluctuation etc. can cause the delay of operations and the skew between clock edges to become random. Fabrication variability makes design time estimation of the delay/power characteristics of operations and functional modules a very hard problem. Lately, a significant amount of research has been focused towards modeling and understanding the impact of fabrication and environmental uncertainty [111, 112, 43, 119].

These uncertainties can potentially cause violation of the implicit deadlines imposed on a DFG that has been scheduled and bound (on resources) causing incorrect and unreliable operation. Typically, existing techniques ignore these uncertainties. Operation delays and clock edges are assumed to be fixed and deterministic. In presence of fabrication and environmental uncertainty, let us suppose that the delay D_i of an operation i has an associated randomness δD_i. Let us also suppose that each clock edge C_i (as indicated in figure 5.1) also has an associated random skew δC_i. This randomness can occur due to environmental fluctuations and/or fabrication variability. Each of these random variables also have an associated probability density function (PDF). The nature of these PDFs and their associated correlations depend on the kind of fabrication and environmental randomness the system needs to tolerate.

In this paper, we assume that there exists a modeling engine that generates these PDFs and also the associated correlations by analyzing the fabrication/environmental randomness. Extensive work is being done to model environ-

mental and fabrication variability [111, 112, 43, 119] and such a modeling engine is feasible. For example, PDFs for module delay can be generated using the recent statistical timing techniques [43, 119] and similar analysis can also be applied to model clock skews. The optimality properties of our algorithm for simultaneous resource binding and dual supply voltage allocation with reliability guarantee for power optimization are independent of the nature of the PDFs (and correlations).

5.1.2 Reliability Guarantee : Definition and Understanding

What is Reliability Guarantee

Let E_{ij} be an edge (data or binding) between operations i and j, with C_m being the clock at which operation i begins and C_n being the clock edge at which operation j begins. The deadline imposed by this edge is as follows:

$$D_i + \delta D_i \leq C_n - C_m + \delta C_n - \delta C_m \tag{5.3}$$

Here D_i is the delay of operation i and $\delta D_i, \delta C_n, \delta C_m$ are the random variables associated with operation delay and clock skews respectively. Under these timing constraints, the environmental and fabrication randomness can impose a finite probability of failure of this edge E_{ij}. The probability of failure can be defined as:

$$P_f^{ij} = Prob(D_i + \delta D_i > C_n - C_m + \delta C_n - \delta C_m) \tag{5.4}$$

This probability depends on D_i, C_n, C_m and the nature of random variables

$\delta D_i, \delta C_n, \delta C_m$ (which could have arbitrary PDFs and correlations). Let us suppose that we are given an *acceptable probability of failure* denoted by α. An edge E_{ij} is considered to have *reliability guarantee* if:

$$P_f^{ij} \leq \alpha \tag{5.5}$$

Lemma 1: Given an edge E_{ij} (data or binding edge), the associated probability of failure P_f^{ij} is monotonically non-decreasing with operation delay D_i.

Proof: We have defined the proability of failure of an edge E_{ij} as

$$P_f^{ij} = Prob(D_i + \delta D_i > C_n - C_m + \delta C_n - \delta C_m) \tag{5.6}$$

where D_i, C_n, C_m are known determistic values and $\delta D_i, \delta C_n, \delta C_m$ are random variables with some probability density functions. Let us now rewrite the definition as:

$$P_f^{ij} = Prob(\delta D_i - \delta C_n + \delta C_m > C_n - C_m - D_i) \tag{5.7}$$

Let δR denote a new random variable defined as:

$$\delta R = \delta D_i - \delta C_n + \delta C_m \tag{5.8}$$

Also, let us define $C = C_n - C_m$. We can rewrite the probability of failure at a given gate delay D_i as:

$$P_f^{ij}(D_i) = Prob(\delta R > C - D_i) \tag{5.9}$$

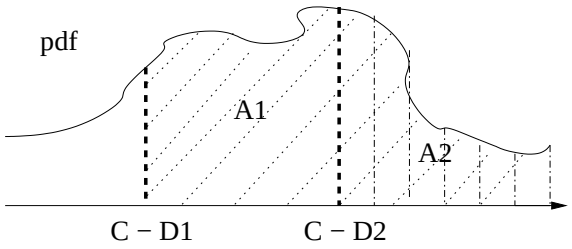


Figure 5.2: Example: Computing P_f^{ij}

where δR has some known pdf.

Without loss of generality, let us suppose that we have two gate delay values D_1 and D_2, where $D_1 > D_2$. In order to show that the probability of failure P_f^{ij} is a monotonically non-decreasing with operation delay D_i, we need to show that $P_f^{ij}(D_1) \geq P_f^{ij}(D_2)$.

$$P_f^{ij}(D_1) = Prob(\delta R > C - D_1) \tag{5.10}$$

$$P_f^{ij}(D_2) = Prob(\delta R > C - D_2) \tag{5.11}$$

Since the pdf of the random variable δR is always positive, and D_1 is greater than D_2, by definition (from equations 5.10 and 5.11) $P_f^{ij}(D_1) \geq P_f^{ij}(D_2)$ will always hold true. This is graphically shown in figure 5.2, where it can be seen that area $A1$ is always greater than or equal to area $A2$.

Hence, we have proved that the probability of failure is a monotonically non-decreasing function of operation delay D_i.

Lemma 1 implies that for a given edge E_{ij}, there exists a value of operation

delay D_i at which the probability of failure is α. Any increase in operation delay beyond this value makes the probability of failure greater than α and any reduction of delay below this value maintains the probability of failure to be atmost α. This value of delay called as *critical delay*, is denoted by D_{crit}^{ij}. It is noteworthy that the critical delay for operation i depends on the edge E_{ij}.

The concept of critical delay of an operation is important since this is the highest value of operation delay that ensures reliability guarantee of edge E_{ij}. Typically higher operation delay means lower operation power. Any further reduction in operation delay will increase operation power without any benefit in reliability guarantee. We note here that a worst-case approximation of the randomness would yield a pessimistic (lower) estimate of critical delay since the PDFs can be correlated in arbitrary ways and result in higher operation power. Hence, a probabilistic definition of critical delay is a truely optimal way to capture reliability.

Computation of Critical Delay

Given an deadline edge E_{ij}, critical delay D_{crit}^{ij} can be computed by solving the following mathematical formulation:

$$D_{crit}^{ij} = Maximize \;\; D_i$$
$$Subject \;\; to: \begin{cases} D_i \leq C_n - C_m \\ P_f^{ij} \leq \alpha \\ D_i^{min} \leq D_i \leq D_i^{max} \end{cases} \tag{5.12}$$

This is a non-linear non-convex optimization problem due to the nature of the probability constraints and the PDFs of the random variables. We will now present a polynomial time optimal algorithm to solve this formulation. Let us suppose that we are given a positive ϵ close to zero. We want to compute the critical delay of an operation D_{crit}^{ij} to be within an ϵ bound.

From Lemma 1, we know that the probability of failure P_f^{ij} is a monotonically non-decreasing function of operation delay D_i. Figure 5.3 denotes this result graphically. We can use a binary search based algorithm to compute the critical delay D_{crit}^{ij} for operation i corresponding to the deadline edge E_{ij}. We start by setting the bounds of the binary search to be at the limits of the entire operation delay range $LowerBound = D_{min}^{i}, UpperBound = D_{max}^{i}$. In each iteration, we compute the probability of failure P_f^{ij} at the mid-point of the current delay range $(LowerBound + UpperBound)/2$. Depending on the magnitude of P_f^{ij}, we update either the upper or the lower bound of the binary search range appropriately. We iterate the binary search until we reach within an ϵ interval range on the operation delay. This ensures that the final computed value of $D_{crit}^{ij_{compute}}$ is within an ϵ bound of the optimal value D_{crit}^{ij} as denoted by equation 5.13.

$$\left| D_{crit}^{ij_{compute}} - D_{crit}^{ij} \right| \leq \epsilon \tag{5.13}$$

It is important to note here that for a given value of operation delay D_i, we need to be able to compute the probability of failure P_f^{ij} of the deadline edge. This above mentioned binary search algorithm assumes that we have such a prediction

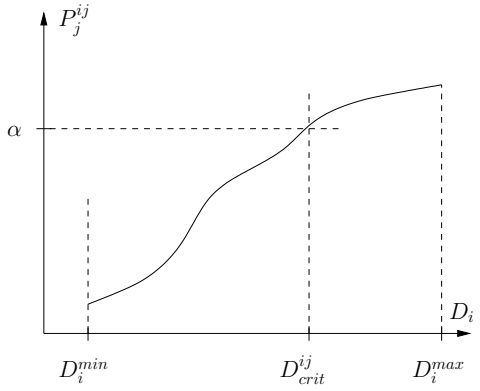


Figure 5.3: Example: Computing D_{crit}^{ij}

engine available to us. In this work we use a monte-carlo based engine to compute the probability of failure for a given value of operation delay D_i. Consider equation 5.4, in order to compute the probability of failure, we need to generate probabilistic samples of the random variables. Let us suppose we have a sample from the space of randomness of the random variables $\delta D_i, \delta C_n, \delta C_m$ denoted by $\Delta D_i', \Delta C_n', \Delta C_m'$. This sample should be generated from the *correlated* density functions associated with these random variables. Now using a monte-carlo based strategy, we can compute the probability of failure of the deadline edge assuming the corresponding operation delay to be D_i under the given correlated variability distributions representing the randomness space.

Using the above mentioned technique, we can compute the critical operation delay value D_{crit}^{ij} corresponding to every deadline edge E_{ij}. It can be noted that the binary search based algorithm presented in this section takes polynomial time to compute the critical operation delay for each deadline edge within an ϵ bound of the optimal value.

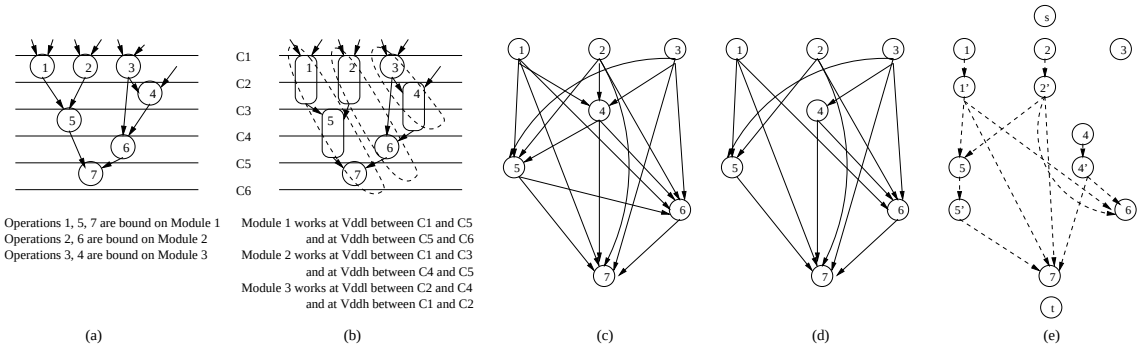


Figure 5.4: Example: (a) DFG (b) Extended Operations (c) Comparability Graph for DFG in a (d) Comparability Graph for DFG in b (e) Network Graph

5.1.3 Simultaneous Resource Binding and Dual-Vdd Allocation With Reliability Guarantees

Problem Definition and Understanding

The input to the problem is a scheduled DFG with resource, and clock latency constraints. We are also given high (Vdd_h) and low (Vdd_l) supply voltage values that can be assigned to a particular operation.

Our objective is to bind operations on modules and assign supply voltage to each operation such that 1) There is no violation of data, resource and timing constraints 2) Maximum number of operations are assigned Vdd_l and among all solutions that have this maximal assignment, the resource binding with minimal switching activity is chosen 3) All associated data and binding edges E_{ij} have a reliability guarantee ($P_f^{ij} \leq \alpha$). Essentially, we try to minimize the power dissipation while maintaining a reliability guarantee on all data and binding edges.

Let us consider a scheduled and bound DFG shown as an example in figure

5.4(b). In this example, module 1 has operations $\{1, 5, 7\}$ bound on it. Between clock edges C1 and C3, this module works at Vdd_l since operation 1 is assigned Vdd_l. It can be seen that we assume that the power supply of functional modules can be changed dynamically (between two pre-decided voltage levels Vdd_h and Vdd_l). The overhead to provide this functionality and maintain correct interaction between modules working at different supply voltage is assumed to be minimal [30]. The primary focus of this paper is to develop the concept of reliability guarantees and a methodology for achieving the same.

Comparability Graphs

For each operation type in the scheduled DFG, the simultaneous resource binding and dual-Vdd allocation problem can be solved independently as long as we ensure that all data dependency constraints (from a reliability perspective) are satisfied. Optimally solving the problem separately for different operation types will give the global optimal solution [30], [63], [64].

For a particular operation type we initialize a *comparability graphs* $G_c(V_c, E_c)$. The comparability graphs of different operation types do not interact and can be solved independently. In the rest of the discussion we assume that we are looking at the comparability graph of only one operation type. For the moment, let us suppose that all operations in the DFG are assigned to Vdd_h (fastest operation delay). Between two operations i and j (i scheduled before j), that have non-overlapping execution times, there is a directed edge e_{ij}. This edge e_{ij} denotes

that i and j are *comparable* operations and can be bound on the same functional resource [30]. Every such edge e_{ij} has a weight w_{ij} associated with it. This weight corresponds to the switching activity between these two operations when bound on the same functional resource with operation j executing after operation i. Therefore, comparability graph edges indicate potential binding edges. For the scheduled DFG shown in figure 5.4(a), let us assume that all operations are of the same type. The comparability graph of the DFG is illustrated in figure 5.4(c) when all operations are assigned Vdd_h.

Let us now try to understand how we can model the dual-Vdd problem into the comparability graph as well. When an operation i works at Vdd_l, its delay D_i increases. We define an operation i to be extendible if its operation at Vdd_l does not 1) Violate any data dependency constraints 2) Violate the schedule latency constraints. The concept of extending an operation is shown in figure 5.4(b) for the scheduled DFG in figure 5.4(a). It can be seen that operations $\{1, 2, 4, 5\}$ are extendible operations as shown in figure 5.4(b), since no data or timing constraints are violated.

Typically, extending operation delays using supply voltage assignment helps in reducing the dynamic power dissipation. Therefore, maximization of Vdd_l assignments to operations while maintaining timing, resource and data dependency constraints is desirable. We can see that even though it may be possible to extend a large number of operations, the resource constraint puts a limit to the number of operations that may finally get extended. For the DFG in figure 5.4(a), one possible set of extendible operations are shown in 5.4(b). The comparability graphs for

the DFGs in figures 5.4(a) and (b) are given by figures 5.4(c) and (d) respectively. As expected the comparability graph also depends upon the operations which have been assigned Vdd_l. For example, operation 1 and 4 which were comparable earlier are no longer comparable when operation 1 is assigned Vdd_l.

Incorporating Reliability Guarantees

Assignment of a particular supply voltage to an operation implies the assignment of a delay value D_i. As indicated earlier each edge in comparability graph represents a potential binding edge. Each potential binding and data edge (given by the schedule) has an associated critical delay D_{crit}^{ij}. If the given assignment of operation supply voltage is such that the operation delay D_i is less than D_{crit}^{ij} for the edge, the corresponding data or binding edge is guaranteed to be reliable (Lemma 1). On the other hand if D_i is greater than D_{crit}^{ij}, then the corresponding edge violates reliability guarantee. For a given D_i, if an associated data edge becomes unreliable, then operation i cannot be assigned delay D_i. On the other hand if a potential binding edge becomes unreliable, then it must be removed from the associated comparability graph. The operation might still work at delay D_i but may be bound differently. Hence, given a supply voltage assignment and its corresponding operation delay, we can easily infer if the supply voltage is valid from a data edge reliability perspective. Also, from the associated comparability graph we can remove all those potential binding edges that do not have a reliability guarantee for the specific delay assignment. Let us call the residual comparability graph as a

reliable comparability graph since all existing potential binding edges are reliable.

This way of judging the reliability of an operation supply voltage assignment is based on the theory of reliability guarantee developed in this work. If Lemma 1 does not hold then this simplistic way of incorporating reliability would not have been possible. Also, the crux of this methodology is based on fast computation of critical delay for each data and potential binding edge using our optimal polynomial time algorithm (section 5.1.2).

Simultaneous Supply Voltage and Resource Binding with Reliability Guarantee

In the existing literature, several works have proposed the use of network flows to reduce power during resource binding [30, 63, 64, 29]. The network flow formulation for solving the dual-Vdd assignment problem during resource binding was proposed in [30]. In this work, we will present a network flow based formulation to solve the same problem under the concept of reliability guarantee in presence of fabrication and environmental variability.

We construct a network $N_g = (s, t, Vn, En, C, K)$ as follows: Let us assume that all operations are assigned Vdd_h. We also assume that all data and binding edges have a reliability guarantee at this supply voltage value. Let $G_c(V_c, E_c)$ be the corresponding comparability graph where every operation i has a vertex v_i in V_c. Note that all potential binding edges in this graph will have a reliability guarantee. We introduce two vertices source s and sink t. Every vertex in V_c has an incoming

edge from the source vertex s and an outgoing edge to the sink t. For a given value of Vdd_l, an operation i is extendible if 1) No data edges become unreliable 2) Timing/latency constraints are not violated. For each extendible operation i, a new vertex v_i' is initialized. We add an edge connecting v_i to v_i' and another edge connecting v_i' to the sink t. This new vertex v_i' which corresponds to operation i being assigned Vdd_l, will also be comparable with other vertices in V_c.

Consider a potential comparability (or binding) edge between vertices (v_i', v_j) (v_j scheduled after the completion of v_i'). If this edge has a reliability guarantee at the given value of Vdd_l (and therefore operation delay D_i), we add this edge in the graph. For each edge in N_g, there is a cost defined by C and capacity defined by K (as explained later).

As an example, let us consider the DFG given in figure 5.4(a). At Vdd_h each of operations $\{1, 2, 3\}$ are comparable with operations $\{4, 5, 6, 7\}$, operation 4 is comparable with operations $\{5, 6, 7\}$, operation 5 is comparable with operations $\{6, 7\}$ and operation 6 is comparable with operation $\{7\}$. This is represented as a comparability graph in figure 5.4(c). Further, we see that operations $\{1, 2, 4, 5\}$ are extendible (assuming that at Vdd_l the corresponding data edges have a reliability guarantee). Hence, the network graph N_g of this design would look like figure 5.4(e). All the edges that exist in the original graph in figure 5.4(c) would also exist in N_g but have not been drawn for clarity. The extendible operations get new vertices namely $\{1', 2', 4', 5'\}$ and their corresponding comparability edges have been shown in figure 5.4(e). Note that these edges will exist only if they have a reliability guarantee. There will also exist edges from s to $\{1, 2, 3, 4, 5, 6, 7\}$ and

from $\{1, 1', 2, 2', 3, 4, 4', 5, 5', 6, 7\}$ to t.

There are four kind of vertices in N_g, namely (s, t, v_i, v_i') and six types of edges, namely those between vertex pairs: (s, v_i), (v_i, v_j), (v_i, v_i'), (v_i', v_j), (v_i, t), (v_i', t). The cost and capacity of edges of Network N_g can be set as follows:

$$
\begin{aligned}
&C(s, v_i) = 0 |\ \forall v_i \in N_g \\
&C(v_i, t) = 0 |\ \forall v_i \in N_g \\
&C(v_i', t) = 0 |\ \forall v_i' \in N_g \\
&C(v_i, v_j) = s_{ij}) |\ \forall edges : (v_i, v_j) \in N_g \\
&C(v_i', v_j) = s_{ij}) |\ \forall edges : (v_i', v_j) \in N_g \\
&C(v_i, v_i') = -T |\ \forall v_i \in N_g \\
&Capacity = K(e) = 1 |\ \forall edges : e \in N_g
\end{aligned}
\tag{5.14}
$$

Here C denotes the cost assigned to an edge, K denotes edge capacity, s_{ij} denotes the switching activity. The cost $C(v_i, v_i')$ is set to $-T$, where T is greater than the total switching activity of all the potential binding edges in N_g. This completes the description of the network. For reasons described later, we impose a node capacity of one unit and a node cost of -2T for each node of type v_i.

It can be seen that the network N_g captures all possible configurations of comparability graph that can be obtained by extending all possible combination of operations.

Another point to note is that the generated N_g has only those potential binding edges that have reliability guarantee. Also, only those extendible nodes v_i' exist whose extension (or assignment to Vdd_l) main-

tains the reliability guarantee of the data edges. Therefore, choosing a resource binding and supply voltage assignment solution from N_g ensures that we always have reliability guarantee.

The simultaneous resource binding and dual-Vdd allocation problem with reliability guarantee can be solved by sending M units of minimum cost flow on N_g from s to t. Here M is the number of available resources for the given operation type.

Lemma 2: A unit flow f in the network N_g corresponds to a resource binding solution where every edge (v_i, v_j) (or (v_i', v_j))on the path of the flow implies that operations i and j are bound on the same functional resource. Also, an edge (v_i, v_i') indicates that operation i is assigned to Vdd_l during its execution.

Proof: Details are given in [31].

The network N_g has cost $C(v_i, v_i')$ set to $-T$, where T is greater than the total switching activity in all the potential binding edges in N_g. This ensures that for a pair of comparable operations $\{i, j\}$, $C(v_i, v_j) > C(v_i, v_i') + C(v_i', v_j)$. Hence, the flow solution always extends an operation (if still reliable). Hence, the final flow solution has maximum number of operations assigned to Vdd_l. Since each vertex v_i has a capacity of 1, it is present on at most on flow path (and therefore bound on only one resource). Also the cost of a node v_i is -2T. Therefore the final flow solutions must include all operations, guaranteeing a valid binding and supply voltage assignment for all operations. The mincost objective ensures that among all valid resource binding solutions with maximum number of Vdd_l assignments we pick the

one with minimum total switching activity. Note that, by construction of N_g, the resulting binding solution will always have reliability guarantee.

Theorem 1: The min-cost flow f, with $|f| = M$ (M being the resource constraint) on N_g gives the largest number of extended operations in the design with the minimum total switching activity on M functional units while maintaining the reliability guarantee.

Proof: Details are given in [31]. It should be noted that by constructing any binding solution on N_g maintains the reliability guarantee.

As has been discussed in [30], the optimality results hold true when we ignore inter-frame considerations regarding the switching activity during cyclic execution of the DFG. The problem in that case formulates as a multi-commodity flow problem. The focus of this work is to introduce the concept of reliability guarantee as a means of handling fabrication and environmental variability into the resource binding and dual-Vdd allocation problem. Similar to [30], we ignore inter-frame considerations in this work.

5.1.4 Architectural Issues

In the proposed technique, each functional module has to have the ability to operate at both Vdd_h and Vdd_l. As shown in figure 5.5, we adopt the architectural scheme similar to that proposed in [30, 38] where each functional module's oper-

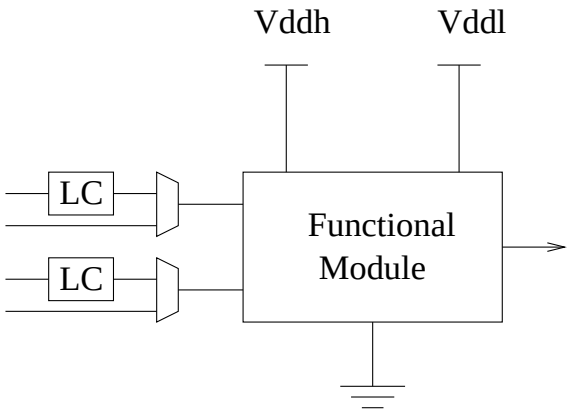


Figure 5.5: Architectural Considerations for Dual-Vdd Scheme

ating voltage can be changed dynamically at run time. Hence, a large number of operations can now operate at Vdd_l, leading to significant power reduction. Since this opens the possibility of having a Vdd_l signal trying to drive a Vdd_h module, we need to add the overhead of using level converters (LC) and appropriate MUXs at the inputs of the functional modules. This extra logic adds a delay penalty to the functional module delay as well as a small power overhead. It has been shown in previous works that the overhead of dual-Vdd power rails and level converters is acceptable compared to the amount of power savings that can be achieved [30, 38].

In order to capture the extra delay overhead due to the level converters and MUXs, the critical operation delay D_{crit}^{ij} can be modified to include the effect of this extra delay penalty ($D_{overhead}$). So the effective critical operation delay would be

$$D_{crit}^{ij_{eff}} = D_{crit}^{ij} - D_{overhead} \tag{5.15}$$

Therefore, the proposed technique is able to consider the overheads while maintaining reliability guarantee in the design.

5.1.5 Determination of the Optimal Vdd_l

The minimal power solution for the simultaneous resource binding and dual-Vdd allocation problem is dependent on Vdd_l. If the value of Vdd_l is very low, very few operations may get extended due to data edge reliability and resource constraint violations. Hence, the solution will have fairly high power. On the other hand, if the value of Vdd_l is high, a large number of operations are likely to get extended but the individual gains in power would be small. Therefore, it is important to determine the *optimal* value of Vdd_l for the simultaneous resource binding and dual-Vdd allocation problem.

Let us assume that we are given a fixed $Vdd_h = VDD$ and we are allowed to choose Vdd_l between $[VDD/2, VDD]$. Our objective is to determine the *optimal* Vdd_l such that the power obtained from the simultaneous resource binding and dual-Vdd allocation solution is minimized. We will now present a polynomial time optimal solution to determine this Vdd_l.

Let us suppose we are given a data or a potential binding edge. As we increase the value of Vdd_l from VDD/2 to VDD, there is a point at which the edge becomes completely reliable and stays reliable. Also consider operations 5 and 6 in figure 5.4(b). It is possible that for Vdd_l very close to VDD, operations 5 and 6 become comparable (even though 5 is delayed). Any increase in Vdd_l will ensure that these operations remain comparable. Given a DFG with n nodes, there can be at most $O(n^2)$ edges (binding and data) in it. Let us suppose we now increase Vdd_l from VDD/2 to VDD. The following lemmas hold true:

Lemma 3: Network N_g can change its edge set E at most $O(n^2)$ times as Vdd_l is increased from VDD/2 to VDD.

Proof: It can be noted that as Vdd_l is moved to a particular value, any edge (data or binding) that becomes valid under reliability guarantee in N_g, will always remain valid as Vdd_l is increased further. Since there are at most $O(n^2)$ edges, a change in the edge set of N_g can happen at most $O(n^2)$ times.

Lemma 4: There are $O(n^2)$ candidate values for *optimal* Vdd_l.

Proof: Given an interval of Vdd_l, $[VDD_i, VDD_j]$ with $VDD_i < VDD_j$ where N_g does not change, we can use the formulation described earlier to optimally solve the resource binding and dual-Vdd allocation problem. In this range, the best value of Vdd_l is obviously VDD_i (since N_g does not change). Because of Lemma 3 there are $O(n^2)$ points between $[VDD/2, VDD]$ at which N_g changes. Hence, there will be $O(n^2)$ consecutive intervals $[VDD_i, VDD_j]$ with $VDD_i < VDD_j$ between $[VDD/2, VDD]$ where N_g is fixed. Therefore, there will be $O(n^2)$ candidate values for optimal Vdd_l (one for each interval).

Algorithm 4 *Algorithm for Determining Optimal Vdd_l*

Step-1: Increase Vdd_l from VDD/2 to VDD

Step-2: Every time N_g changes, solve the Mincost Flow formulation.

(According to Lemma 3 we will have only $O(n^2)$ such events)

Step-3: The lowest power solution among these $O(n^2)$ candidates gives us the *optimal* Vdd_l.

We can determine the *optimal* Vdd_l in polynomial time as given by algorithm 1 above.

5.1.6 Rescheduling the DFG through local perturbation

The input schedule of the DFG to this problem is critical in deciding the power of the design. Proposing a variability-aware scheduling algorithm is beyond the scope of this work. However, we do investigate the possibility of performing small local perturbation based rescheduling on the input schedule. We propose to analyze the solution obtained from our algorithm (under the input schedule) to get insights about which operations to reschedule.

Algorithm 5 *Heuristic for Local Perturbation of Schedule*

INPUT: Scheduled DFG, Binding Solution, Vdd_l

Compute slack for all data edges

Create an ordering of target operations based on slack

For each target operation i {

Try to move operation i one clock step up

Or try to move sink operation j one clock step down

}

Given the current flow solution, we know the operations that were extended, we know the resource bindings of all operations and we know the value of Vdd_l. For each data edge E_{ij} in the design, we compute the slack at the edge using equation 5.16.

$$Slack_{ij} = D^{ij}_{crit} - D_i(Vdd_l) \tag{5.16}$$

We sort the date edges based on their slack in increasing order. The corresponding operations (i for edge E_{ij}) become our target operations in the same order. For a given target operation i (corresponding to edge E_{ij}, we can either move it one clock step up or try to to move the corresponding sink operation j one clock step down in the schedule. In order to move any operation (up/down) it is important to verify that: 1) No data edge gets violated (or becomes tighter in slack than the current data edge) 2) Resource is available for the moved operation in the new clock step.

We iterate over all target operations and try to reschedule them. The overall scheme is shown in algorithm 2.

5.1.7 Consideration of Leakage Power and Soft Errors

Several recent works in high-level synthesis [108, 109] have focused on reliability issues due to soft errors. If we know the value of supply voltage below which a transistor becomes prone to soft errors, we can capture the corresponding maximum allowed module delay through our concept of critical delay D_{crit}^{ij}. Hence, our reliability guarantee based formulation can be extended to consider soft errors as well.

With technology scaling, leakage power has become a critical optimization objective. Techniques like body-biasing [17] allow the effective threshold voltage of a module to be changed at runtime, thereby allowing its leakage change as well. Using the trade-off between module leakage and delay, we can optimize the total power (leakage + dynamic) of the design using a dual-Vdd, dual-Vth scenario. Each operation could work at either Vdd_h/Vdd_l and Vth_h/Vth_l, giving us four possible configurations. Our reliability guarantee based formulation can be extended to consider this scenario as well.

5.1.8 Experimental Results

We experimented with the Mediabench suite [23]. The DFGs were extracted using SUIF/Machine-SUIF and scheduled using [101] for maximum delay slack spreading. Edge switching activity and power/delay information was generated using random input vector simulations followed by VSS simulator and Synopsys DC respectively. We assumed Vdd_h to be $1.8V$ and Vdd_l to exist in the range

Benchmark	Power (nw)	Opt. Vdd_l (V)	#D-Edge Failure	#B-Edge Failure
fft1	161.2	0.9	26	25
fft2	58.3	0.9	12	6
jctrans1	26.7	0.9	4	6
jctrans2	20.3	0.9	0	6
jdmerge1	46.2	0.9	4	14
jdmerge2	90.7	0.9	8	31
jdmerge3	58.3	0.9	5	15
jdmerge4	60.7	0.9	10	13
motion2	69.7	0.9	13	15
motion3	64.8	0.9	13	12
noise_est2	98.0	0.9	3	4

Table 5.1: Power and Reliability Results Obtained From [30]

Benchmark	Power (nw)	Power After Rescheduling (nw)	Improv. %	Opt. Vdd_l (V)	#D/B-Edge Failure	Power [30] (nw)	% Increase compared to [30]
fft1	192.6	181.3	5.8	0.995	0	161.2	12.5
fft2	71.2	71.2	0.0	0.995	0	58.3	22.1
jctrans1	29.8	23.0	29.0	0.995	0	26.7	-13.9
jctrans2	21.9	15.1	21.2	0.995	0	20.3	-25.6
jdmerge1	52.1	52.1	0.0	0.995	0	46.2	12.7
jdmerge2	100.1	75.3	24.8	0.995	0	90.7	-16.9
jdmerge3	64.8	51.3	20.8	0.995	0	58.3	-12.0
jdmerge4	72.1	72.1	0.0	0.995	0	60.7	18.7
motion2	82.9	80.7	2.7	0.995	0	69.7	15.7
motion3	78.4	76.2	2.8	0.995	0	64.8	17.5
noise_est2	103.2	100.9	2.2	0.995	0	98.0	2.9

Table 5.2: Experimental Results: Power, Optimal Vdd_l and Reliability

$[0.9V, 1.8V]$. Clock skew variability was assumed to have a Gaussian distribution with 0 mean and a 3σ interval of $0.05 * Clock - Period$. Module delay variability was assumed to have Gaussian distribution with 0 mean and a 3σ interval of $0.1 * Clock - Period$. All random variables were assumed to be uncorrelated, though our reliability guarantee computation can handle any distribution of variability under any arbitrary correlation model. We implemented the proposed algorithm in C and used MOSEK [49] to solve the convex formulation to get the probability guarantees for each edge in the DFG. All experiments were run on a SunBlade 150 machine with 512mb of RAM. Dynamic Power of each module was computed using $Power_{dyn} = 0.5sCVdd^2f$, where s is the switching activity, C the capacitive loading and f is the operating frequency. Delay of a module was scaled using the alpha model ($Delay = KCVdd/(Vdd - Vth)^{\alpha}$), where K is a constant and Vth the threshold voltage. For our experiments, we chose the value of α to be 0.01, which implies that any deadline edge had a reliability guarantee if

$$P_f^{ij} \leq 0.01 \tag{5.17}$$

We ran two sets of experiments:

1. We performed simultaneous resource binding and dual-Vdd allocation using the scheme proposed in [30]. We extended this approach to consider the proposed optimal Vdd_l selection paradigm. Using Monte-Carlo simulations, we did a reliability analysis on all data and binding edges on the solution thus obtained.

2. We performed simultaneous resource binding and dual-Vdd allocation using the reliability guarantee driven scheme proposed in section 5.1.3. We considered the optimal Vdd_l selection paradigm. Further, we did our heuristic rescheduling followed by another run of our algorithm to get the final solution. We did a reliability analysis on the solution thus obtained.

Table 5.1 shows the results from the first experiment [30]. Column 2 reports the total power obtained from the simultaneous resource binding and dual-Vdd allocation solution and column 3 reports the corresponding optimal Vdd_l. It is interesting to note that the best solution was obtained at $Vdd_l = VDD/2 = 0.9V$ for all benchmarks. We ran Monte-Carlo simulations using the variability distribution mentioned above to compute the probability of failure at each data and binding edge. The total number of edges with a probability of failure greater than α (= 0.01) have been reported in columns 4 (data edges) and 5 (binding edges). As can be seen, a large number of edges are prone to failure and modeling of variability during optimization is critical in getting robust and reliable designs.

Table 5.2 shows the results from the proposed algorithm simultaneous resource binding and dual-Vdd allocation with reliability guarantee. Column 2 reports the total power of the solution obtained from the algorithm using the given input schedule. Using this solution as a guide, we performed rescheduling as described in section 5.1.6 and ran our algorithm again. The final power obtained after rescheduling is reported in column 3. As can be seen from column 4, on an average we get 9.9% additional power savings through rescheduling. The optimal value of Vdd_l was found

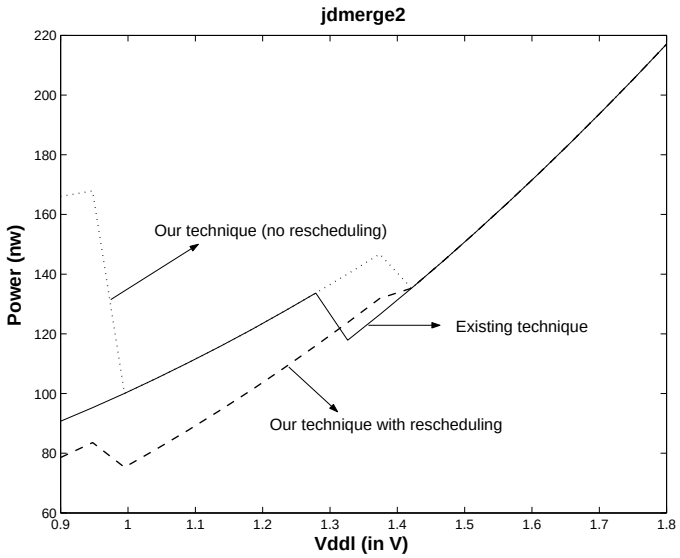


Figure 5.6: Power Versus Vdd_l Trade-Off For *jdmerge2*

to be $0.995V$ for all benchmarks as shown in column 5, which means that our algorithm tends to decrease the delay of each extended operation (thereby increasing its power) in order to ensure reliability guarantee. We ran a reliability analysis on all the data and binding edges in the solution and found them to meet the desired reliability guarantee as indicated in column 6. This result validates our claim that our algorithm provide robust solutions that have a reliability guarantee even in presence of fabrication and environmental variations.

Column 7 shows the power obtained from [30]. Intuitively, we would expect that the power obtained from our algorithm should be higher as we increase the value of Vdd_l to ensure reliability guarantee on all edges. Column 8 shows that on an average the power obtained in our solution is about 3% higher than that from [30]. The more interesting observation is that for some of the designs, our solution gives better power and this can be attributed to the effectiveness of our rescheduling heuristic.

We can see the trade-off between the power versus the selected value of Vdd_l of the solution obtained for benchmark $jdmerge2$ in figure 5.6. Traditional techniques that ignore variability [30], give the minimum power solution at the lowest value of Vdd_l. On the other hand, our proposed technique tends to give the best power solution at a higher value of Vdd_l. This occurs because at low values of Vdd_l, a lot of data and binding edges tend to become unreliable and hence the power obtained through our approach (which guarantees reliability) is high at lower value of Vdd_l. It can also be seen that the rescheduling heuristic followed by our technique is able to significantly lower power at lower values of Vdd_l while maintaining the reliability guarantee.

These results support our claim that it is important to model variability during optimization in order to get robust and reliable designs. The algorithm proposed in this work gives solutions which have a reliability guarantee with minimal increase in total power (3%).

The input schedule of the DFG to this problem is critical in deciding the power of the design. We experimented with a large number of schedules to get an idea about the nature of a *desirable* schedule. As can be expected, our conclusion was that schedules with maximum delay slack spreading (similar to [101]) were in general the best candidates. However, the delay slack spreading could be made more precise by considering the existence of variability during scheduling for maximum power improvements.

Chapter 6

Conclusion and Future Work

In this dissertation, we have tried to address the variability challenge facing nanoscale VLSI design automation. We have addressed all design analysis as well as design optimization aspects of the problem. In chapter 1, we have discussed the existing paradigms that are being used to handle variability due to both fabrication and environmental randomness. We motivate the need for a variability-aware design methodology and present some key advantages and challenges of the same.

In the subsequent chapters of this dissertation, we outline our research contributions in four broad areas namely, design analysis, design time optimization, post-silicon design tunability and runtime optimization. In chapter 2, we outline the main concept behind statistical timing analysis and why it has become an important design analysis tool for improving yield. We presented the existing work in the area of statistical timing analysis and have discussed our research contributions in detail. We propose a novel error budgeting concept in this dissertation that can be generalized to work with any statistical timing analysis scheme allowing the designer to control the trade-off between error induced and runtime, thereby providing significant runtime speedups in statistical timing analysis. Furthermore, we present our general framework for statistical timing analysis considering correlations. This is one of the first works that can handle non-linear delay models and non-Gaussian

variability distributions as well.

In chapter 3, we present two research contributions that propose statistical/probabilistic design optimization philosophies. The first work present a novel probabilistic buffer insertion framework considering wirelength uncertainty. This work presents probabilistic algorithms to optimize the design considering distributions for each wire segment instead of fixed deterministic estimates (as done in traditional buffer insertion). Several probabilistic pruning criteria have been proposed in this work to limit the potential buffer insertion solutions in the design, each with their own benefits and drawbacks. The second work presents a general stochastic linear programming based framework that builds on Monte-Carlo scheme to handle any problem in VLSI-CAD that can be modeled using linear programming. This framework allows us to use any variability distribution, under any underlying correlation model. We apply this framework to the problem of sleep transistor sizing for leakage optimization considering fabrication variability.

In chapter 4, we present our work on variability-driven simultaneous gate sizing and post-silicon tunability allocation. This work brings out an interesting philosophy of post-silicon tunability that can be a powerful means to counter fabrication variability after it has been manifested during fabrication. Design tunability in itself incurs a penalty cost, hence in this work we look to balance the overhead between design time optimization and post-silicon tunability to get maximum yield gains without a severe overhead penalty. We prove that the problem is convex under the longest path constraints and can be solved optimally in polynomial time.

In chapter 5, we present our work on variability-aware runtime optimization

techniques for dynamic power optimization using supply voltage scaling. This work presents a new scheme for modeling fabrication and environmental uncertainty during system-level design while considering runtime power optimization through dual-Vdd functionality. We propose the concept of probabilistic reliability guarantee as an optimization metric. We propose a methodology to model uncertainty in embedded dataflow graphs which can then be optimized probabilistically for ensuring reliability in the design. We specifically look at the problem of simultaneous resource binding and dual-Vdd allocation problem and apply this technique to the problem under a reliability perspective.

6.1 Future Work

Several important directions of future work exist in variability-driven design methodologies. In today's world, variability has opened up three key issues that need to be addressed:

1. Performance (reliability)
2. Yield
3. Power

In this dissertation, we have talked about the microscopic view of the design of a single integrated or embedded system. We have developed analysis and optimization techniques that handle variability for such an isolated system. We have looked at the problems of optimizing the design for performance, yield as well as

power.

There is increasing emphasis on distributed integrated and embedded systems, which could be both off-chip (distributed network of embedded sensors), or on-chip (multi-core designs). These distributed systems are prone to not only fabrication and environmental variations, but also other routing/topology based uncertainty. It is important to extend our paradigms to consider the macroscopic view of distributed integrated and embedded systems as well.

6.1.1 Microscopic View: Single Integrated/Embedded System

In this dissertation, we have developed several analysis and optimization techniques that are targeted towards the nanoscale issues arising due to aggressive technology scaling. In future, we intend to build a complete design automation framework that includes integrated modeling of fabrication and environmental uncertainty in design parameters for analysis and optimization. It is necessary to develop framework that allow for integrated management of reliability, performance and power in designs.

The development of accurate and compact models to represent the fabrication and environmental randomness in design parameters is crucial for effective analysis and optimization. Such models that can be integrated into design analysis and optimization. This requires an in-depth understanding of the causes of randomness in the nanoscale fabrication process. Furthermore, it is also important to understand the degree of randomness and correlation patterns that exist between various sources

of variability. Techniques like statistical timing analysis are crucial in predicting the probabilistic performance spread of the design, thereby driving optimization algorithms to improve design yields. In the future, one needs to develop fast and accurate techniques for statistical analysis of timing, power and noise in nanoscale systems.

Given the statistical analysis information, we need to optimize the design to maximize the probability of meeting the constraints. There are several design methodologies that can be used to integrate statistical analysis information in design optimization.

1. Developing design-time optimization schemes that consider fabrication and environmental randomness. Design yields can be improved by developing accurate and fast algorithms for both probabilistic and deterministic optimization. Chapter 3 in this dissertation presents some work that has been done along these lines.

2. Develop algorithms for allocation of post fabrication tuning of designs that provide a technique to correct the chip after it has been fabricated. In particular, clock skew and body-bias based tuning techniques provide promising directions for future work. This paradigm of design tunability is a very promising solution to the problem of variability. We need to develop techniques that can efficiently integrate this philosophy into mainstream design flows without having a significant overhead in design, fabrication and testing costs.

3. Develop techniques for self-correcting circuits that enable the chip to adjust its

parameters (say threshold voltage through body-biasing) at runtime once the performance shift of the chip due to variability has been computed, thereby improving design yield. Furthermore, we need to investigate exotic circuit techniques that are inherently more tolerant towards fabrication and environmental randomness. Increasing use of asynchronous logic and well as reconfigurable logic (redundant paths etc.) are important steps in this direction.

It is important to develop design management techniques that provide an integrated framework to simultaneously consider these different optimization philosophies for maximum yield improvements.

6.1.2 Macroscopic View: Distributed Integrated and Embedded Systems

Power and delay have been the central optimization objectives in design of integrated and embedded systems. In presence of manufacturing and environmental variations, the randomness caused in power and delay accompanied with other phenomena like temperature hotspots, soft errors have posed severe reliability issues in nanoscale embedded and SoC designs. In order to get robust design solutions, these issues need to be addressed at higher levels of design abstraction where a lot of flexibility in design is available. It has been a challenge to model these low-level issues at higher levels of design flow. In this dissertation, we have proposed a framework for modeling and optimization of such reliability issues in embedded and SoC designs which optimizing the design for dynamic power.

Reliability, yield and power management issues need to be considered not just at the scale of an isolated embedded system, but for a network of distributed integrated and embedded systems. These systems can be off-chip network of systems or an on-chip system (such as a multi-core design). Depending on the routing topology, communication protocols and power management techniques, each of these systems are presented with their own unique challenges. In such situations not only data communication, but also data computation becomes unreliable. Ensuring reliable performance from a network of distributed embedded components potentially working in an unreliable environment is an important problem that needs further investigation. Applications like a distributed system to monitor the carbon monoxide or sulfur levels in a hazardous industrial plant or monitor radioactivity levels in a nuclear power generation plant using a network of embedded sensors are typical examples of unreliable environments where temperature variations can be large between different embedded sensor nodes. On the other hand, even at the scale of an individual embedded or SoC system, different units on the same chip may interact (e.g. a RF receiver may cause power grid noise resulting in the slowdown of the analog to digital converter) causing various design issues. Extending the techniques developed to ensure reliable performance, yield and power from isolated integrated systems need to be broadened to become applicable to the design of distributed systems.

BIBLIOGRAPHY

[1] A. Agarwal, D. Blaauw and V. Zolotov. "Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations". In *Procs of ICCAD*, 2003.

[2] A. Agarwal et al. "Computation and Refinement of Statistical Bounds on Circuit Delay". In *Procs of DAC*, 2003.

[3] A. Agarwal, V. Zolotov and D. Blaauw. "Statistical Timing Analysis Using Bounds and Selective Enumeration". In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.22, Sept. 2003.

[4] A. Agrawal, K. Chopra, D. Blaauw, and V. Zolotov. "Circuit Optimization Using Statistical Static Timing Analysis". In *DAC*, pages 338–342, 2005.

[5] A. Caldwell et al. "Can Recursive Bisection Alone Produce Routable Placements?". In *Proc. of DAC*, 2000.

[6] A. Davoodi and A. Srivastava. "Voltage scheduling under unpredictabilities: A risk-management paradigm". In *Proc. of the IEEE International Symposium on Low Power Electronics and Design*, pages 25–27, August 2003.

[7] A. Davoodi and A. Srivastava. "Variability-Driven Buffer Insertion Considering Correlations". In *Procs of ICCD*, 2005.

[8] A. Davoodi and A. Srivastava. "Variability-Driven Gate Sizing for Binning Yield Optimization". In *Procs of DAC*, 2006.

[9] A. Davoodi, V. Khandelwal and A. Srivastava. "Empirical models for net-length probability distribution and applications". In *Proc. of the IEEE Transactions on Very Large Scale Integration Systems*, October 2004.

[10] A. Devgan and C. Kashyap. "Block-based Static Timing Analysis with Uncertainty". In *Procs of ICCAD*, 2003.

[11] A. Gattiker, S. Nassif, R. Dinakar and C. Long. "Timing yield estimation from static timing analysis". In *Proc. of IEEE International Symposiom on Quality Electronic Design*, pages 26–28, March 2001.

[12] A. Kahng, X. Xu. "Accurate pseudo-constructive wirelength and congestion estimation". In *Proc. of the International Workshop on System-level Interconnect Prediction*, pages 61–68, 2003.

[13] A. Raychowdhury, S. Ghosh, and K. Roy. "A Novel On-chip Delay Measurement Hardware for Efficient Speed-Binning". In *IOLTS*, July 2005.

[14] A. Srivastava and M. Sarrafzadeh. "Predictability: Definition analysis and optimization". In *IEEE Transactions on Computer-Aided Design of ICs and Systems*, pages 118–121, 2002.

[15] A. Srivastava and Majid Sarrafzadeh. "Exact Algorithm for Modifying Buffer Trees Using Buffer Duplication in a Delay Optimization Perspective". In *International Workshop on Logic Synthesis*, 2001.

[16] A. Srivastava S. O. Memik B. Choi and M. Sarrafzadeh. "Achieving design closure through delay relaxation parameter". In *IEEE Transactions on Computer-Aided Design of ICs and Systems*, pages 54–57, 2003.

[17] Ali Keshavarzi et al. "Forward body Bias for Microprocessors in 130nm Technology Generation and Beyond". In *VLSI Circuits Symp.*, pages 312–315, 2002.

[18] C. Ababei and K. Bazargan. "Placement method targeting predictability robustness and performance". In *IEEE Transactions on Computer-Aided Design of ICs and Systems*, pages 81–85, 2003.

[19] C. Alpert, A. Devgan and S.T. Quay. "Buffer Insertion for Noise and Delay Optimization". In *Procs of Design Automation Conference*, 1998.

[20] C. Alpert, A. Devgan and S.T. Quay. "Buffer Insertion with Accurate Gate and Interconnect Delay Computation". In *Procs of Design Automation Conference*, 1999.

[21] C. Bouza. "Stochastic Programming: the state of the art". In *Revista Investigacion Operacional*, page 14(2), 1993.

[22] C. E. Clark. "The Greates of a Finite Set of Random Variables". In *Operations Research*, pages 145–162, 1961.

[23] C. Lee, M. Potkonjak and W.H. Mangione-Smith. "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems". In *International Symposium on Microarchitecture*, 1997.

[24] C. P. Chen, C. N. Chu and D. F. Wong. "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation". In *Proc. of IEEE Transactions on Computer-Aided Design of ICs and Systems*, July 1999.

[25] C. Visweswariah. "Death, taxes and failing chips". In *Proc. of Design Automation Conference*, pages 343–347, 2003.

[26] C. Visweswariah et al. "First-Order Incremental Block-Based Statistical Timing Analysis". In *Procs of DAC*, 2004.

[27] C. Visweswariah et al. "First-Order Parameterized Block-Based Statistical Timing Analysis". In *Procs of TAU*, 2004.

[28] C.C.N. Chu and D.F. Wong. "A New Approach to Simultaneous Buffer Insertion and Wire Sizing". In *IEEE/ACM Intl. Conference on Computer Aided Design*, 1997.

[29] D. Chen and J. Cong. "Register Binding and Port Assignment for Multiplexer Optimization". In *Procs of ASP-DAC*, 2004.

[30] D. Chen, J. Cong and J. Xu. "Optimal Module and Voltage Assignment for Low-Power". In *Procs of ASP-DAC*, 2005.

[31] D. Chen, J. Cong and J. Xu. "Optimal Simultaneous Module and Multivoltage Assignment for Low-Power". In *ACM Transactions on Design Automation of Electronic Systems*, pages 362–386, Vol. 11, No. 2 2006.

[32] D. Sinha, N. V. Shenoy, and H. Zhou. "Statistical Gate Sizing for Timing Yield Optimization". In *ICCAD*, Nov. 2005.

[33] D. Stroobandt. "A priori system-level interconnect prediction: Rent's rule and wire length distribution models". In *Proc. of the International Workshop on System-level Interconnect Prediction*, pages 3–21, 2001.

[34] D. Stroobandt. "Multi-terminal nets do change conventional wire length distribution models". In *Proc. of the International Workshop on System-level Interconnect Prediction*, pages 41–48, 2001.

[35] D. Sylvester and Kurt Keutzer. "Getting to the bottom of deep submicron". In *IEEE Transactions on Computer-Aided Design of ICs and Systems*, pages 203–211, 1998.

[36] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi . "A post-silicon clock timing adjustment using genetic algorithms ". In *Digest of technical papers of the 2003 symposium on VLSI circuits*, 2003.

[37] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, A.L. Sangiovanni-Vincentelli. *SIS: A System for Sequential Circuit Synthesis*. Memorandum No. UCB/ERL M92/41, Department of EECS. UC Berkeley, May 1992.

[38] F. Li et. al. "FPGA Power Reduction Using Configurable Dual-Vdd". In *Procs. of DAC*, June 2004.

[39] F. Najm et al. "Statistical Timing Analysis Based on a Timing Yield Model". In *Procs of DAC*, 2004.

[40] G. Tellez and Majid Sarrafzadeh. "Clock Period Constrained Minimal Buffer Insertion in Clock Trees". In *Proceedings of the International Conference on Computer-Aided Design*, 1994.

[41] M.R. Garey and D.S. Johnson. *Computers and Intractability, A guide to the theory of NP Completeness*. W.H. Freeman and Company, New York, 1979.

[42] H. Chang and S. Sapatnekar. "Statistical Timing Analysis Considering Spatial Correlations Using a Single Pert-Like Traversal". In *Procs of ICCAD*, 2003.

[43] H. Chang, V. Zolotov, C. Visweswariah and S. Narayan. "Parameterized Block-Based Statistical Timing Analysis With non-Gaussian and non-Linear Parameters". In *Procs of DAC*, 2005.

[44] H. J. Bungartz. "Higher Order Finite Elements on Sparse Grids". In *Technical Report SFB-Bericht Nr. 342/01/95 A*, Institut fur Informatik, TU Munich 1995.

[45] H. J. Bungartz and T. Dornseifer. "Sparse Grids: Recent Developments for Elliptic Partial Differential Equations". In *Technical Report TUM-19702, SFB-Bericht Nr. 342/02/97 A*, Institut fur Informatik, TU Munich 1997.

[46] H. Tomiyama et. al. "Module Selection Using Manufacturing Information". In *Procs of ASPDAC*, 1998.

[47] H. Tomiyama et. al. "Statistical Performance-Driven Module Binding in High-Level Synthesis". In *Procs of ISSS*, 1998.

[48] Hai Zhou, D.F. Wong, I-Min Liu and Adnan Aziz. "Simultaneous Routing and Buffer Insertion with Restrictions of Buffer Locations". In *IEEE Trans. on CAD on Integrated Circuits and Systems*, Jul. 2000.

[49] http://www.mosek.com. .

[50] J. A. Davis, V. K. De and J. D. Meindl. "A stochastic wire-length distribution for gigascale integration (GSI): Part II: applications to clock frequency, power dissipation, and chip size estimation,". In *Proc. of IEEE Transaction on Electron Devices*, pages 590–597, March 1998.

[51] J. A. G. Jess, K. Kalafala, S. R. Naidu, R. H. J. M. Otten and C. Visweswariah. "Statistical timing for parametric yield prediction of digital integrated circuits". In *Proc. of Design Automation Conference*, pages 932–937, 2003.

[52] J. Dambre, P. Verplaetse, D. Stroobandt and J. van Campenhout. "Getting more out of Donath's hierarchical model for interconnect prediction". In *Proc. of the international workshop on System-level interconnect prediction*, pages 9–16, 2002.

[53] J. E. Kelley. "The Cutting Plane Method for Convex Programs". In *Journal of SIAM*, pages 703–712, 8(1960).

[54] J. Fishburn and A. Dunlop. "TILOS: A Posynomial Programming Approach to Transistor Sizing". In *ICCAD*, pages 326–328, 1985.

[55] J. Higle and S. Sen. "Stochastic Decomposition: An Algorithm for Two-Stage Linear Programs with Recourse". In *Mathematics of Operations Research*, Vol. 16, No. 3, August 1991.

[56] J. Higle and S. Sen. "Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming". In *Kluwer Academic*, 1996.

[57] J. Higle and S. Sen. "Statistical Approximations for Stochastic Linear Programming Problems". In *Annals of Operations Research*, pages 173–192, 85(1999).

[58] J. Hu, Sachin Sapatnekar. "Simultaneous Buffer Insertion and Non-Hanan Optimization for VLSI Interconnect under a Higher Order AWE Model". In *Proceedings of the ACM International Symposium on Physical Design*, 1999.

[59] J. Jess et al. "Statistical Timing for Parametric Yield Prediction of Digital Integrated Circuits". In *Procs of DAC*, 2003.

[60] J. Le, X. Li and L. Pileggi. "STAC: Statistical Timing Analysis with Correlation". In *Procs of DAC*, 2004.

[61] J. Lillis, C.K. Cheng and T.T. Lin. "Optimal and Efficient Buffer Insertion and Wire Sizing". In *Proc. Custom Integrated Circuits Conf.*, 1995.

[62] J. Lillis, C.K. Cheng and T.T.Y. Lin. "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model". In *IEEE J. Solid-State Circuits*, 1996.

[63] J. M. Chang and M. Pedram. "Register Allocation and Binding for Low Power". In *Procs of DAC*, 1995.

[64] J. M. Chang and M. Pedram. "Module Assignment for Low Power". In *Procs of Euro-DAC*, 1996.

[65] J. R. Birge. "Decomposition and Partitioning Methods for Multistage Linear Programs". In *Operations Research*, pages 989–1007, 33(1985).

[66] J. R. Birge and F. Louveaux. "Introduction to Stochastic Programming". In *Springer-Verlag*, 1997.

[67] J. Singh, V. Nookala, Z. Luo, and S. Sapatnekar. "Robust Gate Sizing by Geometric Programming". In *DAC*, pages 315–320, July 2005.

[68] J. Wong, A. Davoodi, V. Khandelwal and A. Srivastava. "Wire-length prediction using statistical techniques". In *Proc. of the International Conference on Computer-Aided Design of ICs and Systems*, November 2004.

[69] J. Xiong and L. He. "Fast Buffer Insertion Considering Process Variations". In *Proc. of ISPD*, 2006.

[70] J.D. Cho and Majid Sarrafzadeh. "A Buffer Redistribution Algorithm for High-Speed Clock Routing". In *Proceedings of 1993 IEEE Design Automation Conference*, 1993.

[71] Jeng-Liang Tsai, DongHyun Baik, Charlie Chung-Ping Chen, and Kewal K. Saluja. "A yield improvement methodology using pre- and post-silicon statistical clock scheduling". In *Procs. of ICCAD*, 2004.

[72] Jeng-Liang Tsai, Lizheng Zhang and Charlie Chung-Ping Chen. "Statistical Timing Analysis Driven Post-Silicon-Tunable Clock-Tree Synthesis". In *Procs. of ICCAD*, 2005.

[73] K. Chopra, S. Shah, A, Srivastava, David Blaauw, and D. Sylvester. "Parameteric Yield Maximization using Gate Sizing based on Efficient Statistical Power and Delay Gradient Computation". In *ICCAD*, Nov. 2004.

[74] K. Healy. "Optimizing Stochastic Systems: A Retrospective/deterministic Approach". In *Ph.D. Dissertation, Cornell University, Ithaca, NY*, 1992.

[75] A. Srivastava E. Kursun and M. Sarrafzadeh. "Predictability Driven Binding: Methodologies and Tradeoffs". In *Journal of Circuits, Systems and Computers, Special Issue on Low Power IC Designs*, 2002.

[76] L.P.P.P. van Ginneken. "Buffer placement in distributed RC-tree networks for minimal Elmore delay". In *Proc. of International Symposium on Circuits and Systems*, pages 865–868, December 1990.

[77] M. Mani, A. Devgan, and M. Orshansky. "An Efficient Algorithm for Statistical Minimization of Total Power under Timing Yield Constraints". In *DAC*, pages 309–314, July 2005.

[78] M. Orshansky and K. Keutzer. "A general probabilistic framework for worst case timing analysis". In *Proc. of Design Automation Conference*, pages 556–561, June 2002.

[79] M. Orshansky et al. "Fast Statistical Timing Analysis Handling Arbitrary Delay Correlations". In *Procs of DAC*, 2004.

[80] M. R. Guthaus, N. Venkateswaran, C. Visweswariah, and V. Zolotov. "Gate Sizing Using Incremental Parameterized Statistical Timing Analysis". In *ICCAD*, Nov. 2005.

[81] N. Sherwani. "Algorithms for VLSI physical design automation". In *Kluwer*, 1995.

[82] Patrick Mahoney, Eric Fetzer, Bruce Doyle, and Sam Naffziger. "Clock distribution on a dual-core multi-threaded Itanium-family processor ". In *Digest of technical papers of the 2005 international solid-state circuits conference*, 2005.

[83] R. Ahmadi and F. Najm. "Timing Analysis in Presence of Power Supply and Ground Voltage Variations". In *Procs of ICCAD*, 2003.

[84] R. B. Hitchcock, G. L. Smith and D. D. Cheng. "Timing Analysis of Computer Hardware". In *IBM Journal of Research and Development*, pages 100–105, 1982.

[85] R. Kastner, E. Bozorgzadeh and M.Sarrafzadeh. "Predictable routing". In *Proc. of IEEE International Conference on Computer-Aided Design of ICs and Systems*, Nov. 2000.

[86] R. Mukherjee et. al. "Peak Temperature Control and Leakage Reduction During Binding in High Level Synthesis". In *Procs of ISLPED*, 2005.

[87] R. Mukherjee et. al. "Temperature Aware Resource Allocation and Binding in High-Level Synthesis". In *Procs of DAC*, 2005.

[88] R. P. Abato, A. D. Drumm, D. J. Hathaway and L. P. P. P. Van Ginneken. "Incremental Timing Analysis". In *US Patent 5,508,937*, April 1993.

[89] R. Rao, A. Devgan, D. Blaauw and D. Sylvester. "Parametric yield estimation consdering leakage variability". In *Proc. of Design Automation Conference*, pages 442–447, 2004.

[90] R. T. Rochafellar and R. Wets. "Scenarios and Policy Agggregation in Optimization Under Uncertainty". In *Mathematics of Operations Research*, pages 119–147, 16(1991).

[91] R. V. Slyke and R. Wets. "L-shaped Linear Programs With Application to Optimal Control and Stochastic Programming". In *SIAM Journal on App. Math.*, pages 638–663, 17(1969).

[92] R. Y. Rubinstein and A. Shapiro. "Discrete Event Systems: Sensivity Analysis and Stochastic Optimization by the Score Function Method". In *Wiley, NY*, 1993.

[93] Roger Wets. "Stochastic Programs With Fixed Recourse: The Equivalent Deterministic Program". In *SIAM Review*, pages 309–339, 16(1974).

[94] S. Adya and I. Markov. "Fixed-outline floorplanning through better local search". In *Proc. of IEEE International Conference on Computer-Aided Design of ICs and Systems*, pages 321–334, 2001.

[95] S. Arora. "The Approximability of NP-Hard Problems". In *ACM STOC*, 1998.

[96] S. Bhardwaj et al. "TAU: Timing Analysis Under Uncertainty". In *Procs of ICCAD*, 2003.

[97] S. Bhardwaj, S. B.. K. Vrdhula. "Leakage Minimization of Nano-scale Circuits in the Presence of Systematic and Random Variations". In *ICCAD*, Nov. 2005.

[98] S. Borkar et al. "Parameter Variations and Impact on Circuits and Microarchitecture". In *Proc. Design Automation Conference*, June 2003.

[99] S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge 2004.

[100] S. Devadas, A. Ghosh and K. Keuter. "Logic synthesis". In *McGraw-Hill*, 1994.

[101] S. Ogrenci Memik, A. Srivastava, E. Kursun and M. Sarrafzadeh. "Algorithmic Aspects of Uncertainty Driven Scheduling". In *Procs of ISCAS*, 2002.

[102] S. P. Mohanty et. al. "Energy Efficient Datapath Scheduling Using Multiple Voltages and Dynamic Clocking". In *ACM Transactions on Design Automation of Electronic Systems*, pages 330–353, April 2005.

[103] S. P. Mohanty et. al. "Simultaneous Peak and Average Power Minimization During Datapath Scheduling". In *IEEE Transactions on Circuits and Systems Part I*, pages 1157–1165, June 2005.

[104] S. P. Mohanty et. al. "ILP Models for Simultaneous Energy and Transient Power Minimization During Behavioral Synthesis". In *ACM Transactions on Design Automation of Electronic Systems*, pages 186–212, Jan 2006.

[105] S. Raj et al. "A Methodology to Improve Timing Yield in the Presence of Process Variations". In *Procs of DAC*, 2004.

[106] S. Raj, S. B. K. Vrudhula and J. Wang. "A methodology to improve timing yield in the presence of process variations". In *Proc. of Design Automation Conference*, pages 448–453, June 2004.

[107] S. Sapatnekar, V. B. Rao, P.M. Vaidya, and S. M. Kang. "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization". In *IEEE Transactions on CAD*, pages 1621–1634, Nov. 1993.

[108] S. Tosun et. al. "An ILP Formulation for Reliability-Oriented High-Level Synthesis". In *Proc. of ISQED*, 2005.

[109] S. Tosun et. al. "Reliability-Centric High-Level Synthesis". In *Proc. of DATE*, 2005.

[110] S. Wallace. "Decision making under uncertainty: Is sensitivity analysis of any use?". In *Operations Research*, pages 20–25, January 2000.

[111] Sani R. Nassif. "Design for variability in DSM technologies". In *Proc. First International Symposium on Quality of Electronic Design, p.451, March 20-22, 2000*, pages 451–454, March 2000.

[112] Sani R. Nassif. "Modeling and forecasting of manufacturing variations". In *Proc. Asia South Pacific design automation*, pages 145–150, 2001.

[113] Semiconductor Industry Association. "National Technology Roadmap for Semiconductor". 1997.

[114] Simon Tam, Stefan Rusu, Utpal Nagarji Desai, Robert Kim, Ji Zhang, and Ian Young. "Clock generation and distribution for the first IA-64 microprocessor". In *IEEE Journal of Solid-State Circuits*, pages 35(11):1545–1552, Nov 2000.

[115] V. Khandelwal, A. Davoodi, A. Nanvati and A. Srivastava. "A probabilistic approach to buffer insertion". In *Proc. of the International Conference on Computer-Aided Design of ICs and Systems*, November 2003.

[116] V. Khandelwal and A. Srivastava. "Leakage Control Through Fine-Grained Placement and Sizing of Sleep Transistors". In *Proc. of ICCAD*, pages 533–536, 2004.

[117] V. Khandelwal and A. Srivastava. "Variability Driven Formulation for Simultaneous Gate Sizing and Post-Silicon Tunability Allocation". In *Procs of ISPD*, April 2007.

[118] V. Khandelwal et al. "Efficient Statistical Timing Analysis through Error Budgeting". In *Procs of ICCAD*, 2004.

[119] V. Khandelwal et al. "A General Framework for Accurate Statistical Timing Analysis Considering Correlations". In *Procs of DAC*, 2005.

[120] V. Mehrotra, S. L. Sam, D. Boning, et al. "A methodology for modeling the effects of systematic within-die interconnect and device variation on circuit performance". In *Proc. of Design Automation Conference*, pages 172–175, 2000.

[121] W. K. K. Haneveld and M. H. Vander Vlerk. "Stochastic Integer Programming: general models and algorithms". In *Annals of Operations Research*, pages 39–57, 85 1990.

[122] W.C. Elmore. "The transient analysis of damped linear networks with particular regard to wideband amplifiers". In *Journal of Applied Physics, vol. 19(1)*, 1948.

[123] Weiping Shi and Zhuo Li. "An O(nlogn) Time Algorithm for Optimal Buffer Insertion". In *Procs of Design Automation Conference*, Jun. 2003.

[124] X. Bai, C. Visweswariah, p. N. Strenski and D. J. Hathaway. "Uncertainty-aware circuit optimization". In *Proc. of Design Automation Conference*, pages 58–63, 2002.

[125] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu. "New paradigm of predictive MOSFET and interconnect modeling for early circuit design". In *Proc. of CICC*, pages 201–204, 2000.

[126] Y. Gao and D.F. Wong. "A Graph Based Algorithm for Optimal Buffer Insertion under Accurate Delay Models". In *Procs of DATE*, 2001.

[127] Y. Zhan, A. J. Strojwas, X. Li, L. T. Pileggi, D. Newmark, and M. Sharma. "Correlation-aware statistical timing analysis with nongaussian delay distributions". In *Procs of DAC*, 2005.

[128] Yanbin Jiang, Sachin Sapatnekar et al. "Combined Transistor Sizing with Buffer Insertion for Timing Optimization". In *Proceedings of the IEEE Custom Integrated Circuits Conference*, 1998.