

5-8-2008

# Advanced Algorithms for VLSI: Statistical Circuit Optimization and Cyclic Circuit Analysis

Osama Neiroukh  
*Portland State University*

Follow this and additional works at: [https://pdxscholar.library.pdx.edu/open\\_access\\_etds](https://pdxscholar.library.pdx.edu/open_access_etds)



Part of the [Electrical and Computer Engineering Commons](#)

**Let us know how access to this document benefits you.**

---

## Recommended Citation

Neiroukh, Osama, "Advanced Algorithms for VLSI: Statistical Circuit Optimization and Cyclic Circuit Analysis" (2008). *Dissertations and Theses*. Paper 6150.  
<https://doi.org/10.15760/etd.8010>

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: [pdxscholar@pdx.edu](mailto:pdxscholar@pdx.edu).

ADVANCED ALGORITHMS FOR VLSI:  
STATISTICAL CIRCUIT OPTIMIZATION  
AND  
CYCLIC CIRCUIT ANALYSIS

by

OSAMA NEIROUKH

A dissertation submitted in partial fulfillment of the  
requirements for the degree of

DOCTOR OF PHILOSOPHY  
in  
ELECTRICAL AND COMPUTER ENGINEERING


Portland State University  
©2008

## DISSERTATION APPROVAL

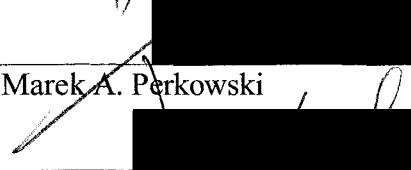
The abstract and dissertation of Osama Neiroukh for the Doctor of Philosophy in Electrical and Computer Engineering were presented May 8, 2008, and accepted by the dissertation committee and the doctoral program.

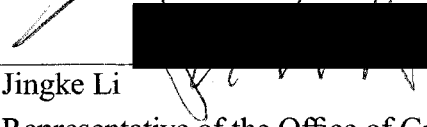
### COMMITTEE APPROVALS

  
Xiaoyu Song, Chair

  
Garrison W. Greenwood

  
Douglas V. Hall

  
Marek A. Perkowski

  
Jingke Li  
Representative of the Office of Graduate Studies

### DOCTORAL PROGRAM APPROVAL:

  
Malgorzata Chizanowska-Jeske, Director  
Electrical and Computer Engineering Ph.D. Program

## ABSTRACT

An abstract of the dissertation of Osama Neiroukh for the Doctor of Philosophy in Electrical and Computer Engineering presented May 8, 2008.

Title: Advanced Algorithms for VLSI: Statistical Circuit Optimization and Cyclic Circuit Analysis

This work focuses on two emerging fields in VLSI. The first is use of statistical formulations to tackle one of the classical problems in VLSI design and analysis domains, namely gate sizing. The second is on analysis of non-traditional digital systems in the form of cyclic combinational circuits.

In the first part, a new approach for enhancing the process-variation tolerance of digital circuits is described. We extend recent advances in statistical timing analysis into an optimization framework. Our objective is to reduce the performance variance of a technology-mapped circuit where delays across elements are represented by random variables which capture the manufacturing variations. We introduce the notion of statistical critical paths, which account for both means and variances of performance variation. An optimization en-

gine is used to size gates with a goal of reducing the timing variance along the statistical critical paths. Circuit optimization is carried out using a gain-based gate sizing algorithm that terminates when constraints are satisfied or no further improvements can be made. We show optimization results that demonstrate an average of 72% reduction in performance variation at the expense of average 20% increase in design area.

In the second part, we tackle the problem of analyzing cyclic circuits. Compiling high-level hardware languages can produce circuits containing combinational cycles that can never be sensitized. Such circuits do have well-defined functional behavior, but wreak havoc with most tools, which assume acyclic combinational logic. As such, some sort of cycle-removal step is usually necessary. We present an algorithm able to quickly and exactly characterize all combinational behavior of a cyclic circuit. It used a combination of explicit and implicit methods to compute input patterns that make the circuit behave combinational. This can be used to restructure the circuit into an acyclic equivalent, report errors, or as an optimization aid. Experiments show our algorithm runs several orders of magnitude faster than existing ones on real-life cyclic circuits, making it useful in practice.

Dedicated to my wife Sophia.

## **Acknowledgments**

I wish to thank my parents who saw to it that I get the best education possible without regard to the financial burdens imposed by such a pursuit. My wife and children have exhibited immeasurable patience as my doctoral degree spanned the better part of eight years while I juggled a multitude of responsibilities to God, family, work, and community in addition to a full time job. Several people at Portland State University influenced my education and inspired me in different ways. My advisor Dr Xiaoyu Song helped provided valuable input along the years that positively affected my research direction, my publications, and overall steps towards degree completion. The first course I took at PSU was with Dr Perkowski who is a great inspiration to anyone looking for research advice. He brings high passion to electrical engineering despite having spent decades in the field. The remainder of the faculty on my PhD committee have also provided useful inputs that shaped my output. Thanks are due to Intel Corporation who provide financial support for continuing education and foster a continuous improvement attitude.

# Table of Contents

<b>Acknowledgments</b>	<b>ii</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Figures</b>	<b>viii</b>
<b>Chapter 1. Introduction</b>	<b>1</b>
<b>Chapter 2. Statistical Optimizations of Digital Circuits</b>	<b>8</b>
2.1 Introduction . . . . .	8
2.2 Literature Survey . . . . .	9
2.2.1 Statistical Yield Optimization . . . . .	10
2.2.2 Gate Sizing . . . . .	11
2.2.3 SSTA: Statistical Static Timing Analysis . . . . .	14
2.3 SSTA Overview . . . . .	16



2.3.1	Introduction to SSTA . . . . .	16
2.3.2	Challenges and Assumptions in SSTA . . . . .	19
2.4	SSTA-based Circuit Optimization: Problem Overview . . . . .	21
2.4.1	Problem Formulation . . . . .	21
2.4.2	Overview of Research . . . . .	26
2.5	Statistical Gate Sizing . . . . .	28
2.5.1	Overview of Algorithm . . . . .	28
2.5.2	FULLSSTA : Full Statistical Static Timing Analysis . . . . .	29
2.5.3	FASSTA : Fast Statistical Static Timing Analysis . . . . .	34
2.5.3.1	Statistical Critical Path Identification . . . . .	39
2.5.3.2	Subcircuit extraction and ranking . . . . .	43
2.5.4	Experimental results . . . . .	44
2.5.5	Concluding Remarks . . . . .	49
2.5.6	Benefits of Research . . . . .	49
2.6	Summary . . . . .	50
<b>Chapter 3. An Efficient Algorithm for Analysis of Cyclic Circuits</b>		<b>52</b>
3.1	Introduction . . . . .	52

3.2	Notation and Definitions . . . . .	55
3.3	Literature Survey . . . . .	57
3.3.1	Origins of Cyclic Circuits . . . . .	57
3.3.2	Analysis of Cyclic Circuits . . . . .	58
3.3.3	Synthesis of Cyclic Circuits . . . . .	60
3.3.4	Most Recent Publications on Analysis of Cyclic Circuits	61
3.4	Types of Cycles . . . . .	62
3.5	Our Circuit Model . . . . .	63
3.6	Combinational Circuits . . . . .	65
3.7	Finding a Combinational Cover for a Cyclic Circuit . . . . .	68
3.7.1	Theoretical Background . . . . .	68
3.7.2	Searching for combinational behavior . . . . .	71
3.7.3	Merging partial assignments . . . . .	73
3.7.4	Another Example . . . . .	79
3.8	Experimental Results . . . . .	83
3.9	Benefits of Proposed Research . . . . .	83
3.10	Conclusions . . . . .	83

3.11 Summary . . . . .	86
<b>Chapter 4. Conclusions</b>	<b>90</b>
4.1 Statistical Optimization of Digital Circuits . . . . .	90
4.2 An Efficient Algorithm for Analysis of Cyclic Circuits . . . . .	92
<b>Bibliography</b>	<b>94</b>

## List of Tables

2.1	Experimental Results: $\lambda = 3$ . . . . .	45
2.2	Experimental Results: $\lambda = 9$ , runtime is in minutes . . . . .	46
3.1	Comparison with Edwards [22] . . . . .	84

## List of Figures

2.1	Mapping from circuit to timing graph for timing analysis. . . .	17
2.2	Examples of cumulative and probability distribution functions for a circuit's timing. . . . .	19
2.3	Example of circuit output Delay PDFs . . . . .	25
2.4	Example of circuit output Delay CDFs . . . . .	26
2.5	Overview of Statistical Sizer Algorithm . . . . .	30
2.6	Extracting Subcircuit Cost for Statistical Sizer . . . . .	31
2.7	Probabilistic event representing delay at a given edge in an SSTA timing graph . . . . .	31
2.8	Shift with scaling and grouping techniques to perform convolu- tion of input and gate-delay PDFs to compute the output-delay PDF . . . . .	32

2.9	Tracing worst negative statistical slack (WNSS) path. Numbers in parenthesis are $(\mu, \sigma)$ of arrival time. The shaded nodes indicate the WNSS using our method. . . . .	40
2.10	Normalized Mean-Std Variation for C432 at different $\lambda$ . The x-axis shows the mean while the y-axis has the standard variation. . . . .	48
3.1	A trivial cyclic circuit and its truth table . . . . .	53
3.2	Cyclic circuit for illustrating definitions . . . . .	56
3.3	Rivest's Circuit . . . . .	58
3.4	Cyclic circuit arising from resource sharing due to Stok [56] . . . . .	59
3.5	Examples of true and false cycles . . . . .	63
3.6	The three-valued simulation algorithm, which takes a circuit $\langle G, I, W \rangle$ , an input function $x$ , and an infinite schedule of gates $s$ . It evaluates gates until it reaches a fixed point using EVAL, which updates a single (NAND) gate. . . . .	66
3.7	(a) A cyclic circuit. (b) Partial assignments and their induced frontiers—the boundary between defined and X-valued gates after applying inputs. . . . .	67

3.8	Our algorithm for finding a minimal set of PAs for a circuit (SCC) that together cover all its combinational behavior. . . .	72
3.9	Illustration of merging PAs at a gate. . . . .	77
3.10	Our PA merging algorithm: return a set of PAs that apply non- controlling values to every input of a gate. . . . .	80
3.11	Small cyclic circuit for illustrating partial assignment extraction	81
3.12	PAs from applying controlling values to each input in isolation. All frontiers are either gate $V$ or gate $Z$ . . . . .	88
3.13	Partial assignment extraction on a small cyclic circuit (a) POS and final ISOP for frontier gate $V$ . (b) POS and ISOP for $Z$ . (c) A minimal set of partial assignments that reproduce all combi- national behavior. . . . .	89

# **Chapter 1**

## **Introduction**

Advances in VLSI technology continue to present both challenging and exciting opportunities for advanced research in electrical and computer engineering. Moore's law has continued to motivate designers to keep fulfilling its prediction by continually shrinking down device geometries and packing more devices per square micron while meeting numerous challenges brought on by the most recent technologies. These challenges include several parameters such as power density and dissipation, supply voltage droop, and reliability under wide operating conditions. However, the most pressing difficulty facing designers today is the decreasing correlation between physical verification (PV) models used in pre-silicon design and optimization and behavior of manufactured circuits on silicon. Manufacturing variations and its adverse effect on predictability on silicon behavior have spurred designers to seek new tools and methodologies to deal with these variations in order to better predict



performance of circuits during design cycle.

This dissertation focuses on two emerging fields in VLSI. The first is use of statistical formulations to tackle one of the classical problems in VLSI design and analysis domains, namely gate sizing. The second is on analysis of non-traditional digital systems in the form of cyclic combinational circuits. Neither field is really new, early publications in both topics can be traced back to the 60's and 70's as our literature survey will show. However, both fields have received renewed interest recently, with statistical approaches in particular featured prominently at all major CAD conferences nowadays and getting increased coverage in journals.

Usage of statistical approaches has been well-known in parametric yield analysis for post-manufacturing die sorting and analysis, but had not made its foray yet into pre-silicon analysis or optimization areas. It began to receive increased focus around the turn of the 21st century when Physical Verification (PV) models of pre-silicon behavior started to diverge substantially from actual silicon measurements. The field has exploded in the past 5 years, with almost every analysis or optimization problem in VLSI revisited from a statistical perspective. It remains to be seen whether this is merely an academic curiosity or whether statistical analysis and optimization techniques will narrow the widening gap between pre-silicon models and post-silicon behavior.

As of this writing, IBM is the only company which has publicly claimed to deploy statistical static timing analysis in pre-silicon PV models of industrial circuits [32,33]. It is not clear to what extent does IBM use this methodology, and whether it merely augmented or completely replaced standard static timing tools as the golden timing verification model. At the same time, the field cannot be neglected. At least for the time being, it provides a rich field for research, though competition is stiff with numerous researchers both in academia and industry attacking a variety of CAD problems using statistical techniques very aggressively.

Cyclic circuits appear to have been a black sheep of digital circuits. Convincing examples of cyclic circuits that had provably less gates than any acyclic equivalents have been around for decades. Nevertheless, cyclic circuits have not received much attention in industry as candidates for deployment in real-life ASICs or custom designs. While circuits that have registers (flops or latches) that depend on current state for future state and output calculation are commonplace in state machine design, purely combinational cyclic circuits are not intuitive to reason about. Despite this, the lure of area savings and potential for other advantages has continued to spur researchers to study these circuits. More recently, a synthesis engine was proposed that produces cyclic implementations at an area saving compared to traditional synthesis. The re-

search was well-accepted, receiving best paper award at Design Automation Conference in 2003 [49].

An alternate motivation for tackling cyclic circuits arises during processing of high level hardware modeling languages such as ESTEREL [8]. Some of the literature on cyclic circuit analysis was contributed by researchers who were trying to grapple with ESTEREL and other synchronous programming languages such as LUSTRE [25] and ARGOS [36]. Synthesizing these languages into digital circuits often yields loops that are difficult for regular CAD tools to handle. Ability to handle cycles became imperative for compilation of these languages, forcing researchers to find ways to take out these cycles as a post-processing step before handing off these circuits to other tools.

This dissertation tackles both areas separately. The first part focuses on usage of statistical analysis in a digital circuit optimization setting. The main contribution is an adaptation of well-known gate sizing techniques to use a statistical timing model toward reducing the performance variation of a circuit at design time. The second part of this dissertation investigates more efficient methods for analysis of cyclic circuits. We note that the two topics are distinct with no overlap in our context.

At a high level, there are similarities in the two research areas we pro-

pose here. Both problems involve netlist level analysis and optimization at the gate-level granularity. Gate sizing is NP-complete while cyclic circuit analysis is considered to be co-NP-complete. There are also distinct differences between the two areas. Gate sizing belongs to the class of electronic design automation problems, while cyclic circuit analysis is an enumeration problem as we will show.

There are advantages to tackling such disparate problems as part of a single PhD research. The field of IC design is increasingly becoming more vertical, with design, analysis and verification becoming strongly coupled with an expectation that a designer can move between these area with ease. Another rationale here was that a practitioner in the field of VLSI design would do well to understand in depth both analysis and design optimization domains of CAD techniques from an algorithmic and practical perspective as they present uniquely different challenges. Our characterization below of the differences is rather subjective but reflects the author's combined industrial and academic experience with these fields.

Automated circuit optimization techniques are much more heuristics based, with many decisions and tunings that can work for one design but not other designs. In addition, there is a possibility of oscillations or other unexpected problems where the algorithm seems to go astray. Given that most of the

problems EDA tools attempt to solve are NP-complete, a thorough understanding of the challenges of overcoming local optima and explaining otherwise odd outputs is a daily struggle for any engineer attempting to use and steer EDA design tools. This has been a perennial component of this author's job at Intel as an automation design engineer covering automated synthesis, placement, sizing, and routing tools. The research that was done in this field has helped the author tremendously with understanding the difficult tradeoffs these tools are juggling especially as the given timing, area, and power constraints that are usually impossible to meet at once.

The chief challenge with analysis techniques in CAD is reducing runtime while keeping peak memory usage within reasonable bounds. Analysis algorithms have different requirements than optimization algorithms. A design optimization algorithm might be successful even if terminates due to exceeding acceptable runtime or runs out of memory and stops earlier than it would otherwise having achieved a satisfactory result. On the other hand, an analysis algorithm must complete its execution; a partial result is of no value in practice. This makes data representation and programming methodology used critical to a successful implementation. Usage of existing technologies such as SAT, BDD manipulation, and ATPG techniques should be considered as much as possible by mapping the given problem into one of these formulations. This

enables designers to reuse efficient solvers that are publicly available for each of these formulations and improving the state of the art by focusing on the unique problem at hand.

The rest of this dissertation is structured as follows. Chapter 2 starts with an overview of statistical analysis and optimization of digital circuits. It presents an extensive literature survey covering the topic and gives an introduction of the problem we addressed and motivation for it. It presents our proposed algorithm for statistical gate sizing and provides experimental results and detailed analysis of the algorithm's performance on tested circuits. Chapter 3 presents a literature survey on cyclic circuits and presents motivation for the problem we tackle. We provide theoretical underpinnings for our circuit model and present our original algorithm for cyclic circuit analysis as well in-depth step-by-step review of how it works in practice with aid of examples. Finally chapter 4 gives concluding remarks about our contributions and directions for future research.

## **Chapter 2**

### **Statistical Optimizations of Digital Circuits**

#### **2.1 Introduction**

Recent advances in VLSI have continued to shrink device geometries at a steady rate in accordance with Moore's Law. However, this advancement has also been accompanied by increasing variations in the performance of fabricated circuits. Numerous factors have contributed to this trend including clock PLL jitter, noise, PV model inaccuracies, and manufacturing variations. Nevertheless, it is often desirable to manufacture ASICs on advanced technology nodes due to substantial increase in available device count, reduction in power consumption, higher yields and lower costs due to the larger 300mm wafers.

Researchers have recently focused on statistical analysis approaches in an attempt to grapple with these sources of performance variations. Statistical static timing analysis (SSTA) is a modification of static timing analysis (STA)

for determining delay across a circuit. SSTA models delay arcs across gates as random variables rather than discrete values which are used in regular STA. SSTA propagate timing constraints across a circuit using probability distribution functions (pdfs). A virtual sink is often used for all the circuits' outputs producing a single pdf that represents delay across the circuit.

When this research was first conceived, a substantial focus had gone into the analysis aspect of this problem [1, 28]. However, research into statistical optimization of circuits had been surprisingly diminutive. Circuit optimization was done in [29] by using LANCELOT [17] but had severe limitation on circuit size and used unrealistically simple gate delay models. A concept of criticality of gates was used in [27] but did not address the variance of the timing path delays. A transistor level approach was presented in [4]. Several yield-specific techniques were presented in [21].

## **2.2 Literature Survey**

An extensive review of prior work on areas related to this research area was undertaken before research into this area was started. Below is a summary of contributions in this field. It should be noted that research into application of statistical techniques to mainstream EDA problems continues to advance



at a very rapid pace, with almost all major CAD conferences dedicating at least one or two sessions to statistical analysis and optimization approaches. For example, the entire 2004 ACM/IEEE TAU Workshop on Timing Issues in the Specification and Synthesis of Digital Systems was dedicated to statistical approaches to timing analysis. Many topics that had previously appeared to mature such as static timing analysis, power analysis, and gate-level design optimization are now being re-examined using statistical formulations. We survey publications in a number of research thrusts below. However, we stress that such a survey is only a sampling of what is rapidly becoming a vast body of literature covering all aspects of electronic design and analysis.

### **2.2.1 Statistical Yield Optimization**

A wide variety of methods for yield optimization has been developed over the last few decades. A comprehensive reference that covers an exposition of representative techniques is [21]. Traditional statistical optimization methods define the yield as the probability of a random variable that represents a performance metric belonging to an acceptability region. This acceptability region can be expressed as a multi-dimensional integral which is typically evaluated by Monte-Carlo based methods or by relying on analytical expressions for the circuit performance parameters of interest. Monte-Carlo techniques are

far too expensive to deploy for digital circuit design due to the dimensionality of the statistical space.

While Monte-Carlo techniques find many uses in analysis of circuits, they are rarely deployed in a circuit optimization context. Modeling of performance metrics such delay along with possible variations using analytical expressions is also intractable especially in deep submicron technologies. In light of this, we found that traditional yield optimization techniques while being highly useful in parametric yield contexts are not directly usable for the problem at hand.

### **2.2.2 Gate Sizing**

Gate sizing has been studied extensively in the literature. Gate sizing is typically performed after technology mapping during logic synthesis and repeated several times during the physical design process. The aim of gate sizing is to assign sizes to all gates in a circuit such that some objective function is satisfied, possibly under some constraints. Typical formulations include minimizing area or power subject to a maximum delay constraint. Various gate delay models have been proposed in the literature such as Load-Independent Delay Model (LIDM) and Load Dependent Delay Model (LDDM).

The choice of which gate delay model to use has a direct impact on choice and efficacy of the gate sizing algorithm to be deployed. Since the output load of a gate has a great impact on delay across it, LIDM is of little value in real optimization contexts. Gate sizing has been shown to be NP-complete under LDDM which rules out finding globally optimal solutions for real-life circuits which consist of tens to hundreds of thousands of gates.

Research in circuit sizing has been carried out both at the transistor as well as gate level. Transistor level sizing is more accurate but presumes ability to size and therefore adjust layout on a per transistor basis, which is becoming less common due to layout complexity of recent processes. It is also limited to smaller circuits compared to gate-level approaches. Gate sizing relies on standard cell libraries that can come from library vendors which are designed in discrete sizes, laid out, and pre-characterized for timing, area and power. A typical cell characterization produces lookup tables for every input-pin output-pin transition. Timing characterization tables represent input slope and output capacitance as inputs with output slopes and delay through gate as outputs.

Gate sizing algorithms can be classified into one of two categories: global approaches and local approaches. Global approaches solve gate sizing in the continuous domain by relying on optimization techniques such as convex programming with posynomials [23], linear programming [5], sequential

quadratic programming [37], or Lagrangian Relaxation [13]. While these approaches can claim a globally optimum solution, they have two drawbacks. The presumption of a convex problem where a single global optimum exists is not supported by practical evidence. More importantly, library gates tend to come in pre-determined discrete sizes and solving the problem in the continuous domain requires snapping back size assignments to closest available gate sizes. Since standard cell library gates tend to be sized in a geometric progression of drive strength, this discretization may assign drive strengths significantly different from the values obtained in the continuous domain. Advantages of global approaches include a global solution without oscillations and faster run-time compared to local approaches.

Local sizing approaches assign gate sizes using local gain-based or greedy heuristics. Examples of this approach are available in [14,19,40]. Most of these algorithms share several common elements. The critical path, sometimes referred to as the Worst Negative Slack (WNS) path, is usually targeted for optimization. We note that the WNS path can change as the optimization proceeds so the path being evaluated for resizing must be updated at specific intervals in the optimization iteration. The algorithms can be run in a constrained mode where delay for example is optimized first then area is recovered as far as possible without violating a delay constraint. Other constraints can be similarly

satisfied either during optimization by not violating some cost/benefit ratio or in a recovery mode after unconstrained optimization.

Coudert [19] argues that accurate delay models make gate sizing a non-linear, non-convex, constrained, discrete optimization problem. Our experience corroborates this assertion, especially for deep-submicron technologies which are the target domain for this research. Many of the commercial tools for logic and physical synthesis such as Design Compiler® and Physical Compiler® from Synopsys® also use local approaches for gate sizing as these approaches are more accurate despite being slower than global approaches.

### **2.2.3 SSTA: Statistical Static Timing Analysis**

The earliest paper that suggested a statistical approach to timing analysis known to the author is [41]. The author attempted to determine the distribution of delay from source to sink of an acyclic directed graph that had probability distributions associated with its elements. However, the focus on use of statistical approaches in timing analysis is relatively new. Pioneering works in this field appeared in [11, 20, 30].

While difficulties in deterministic timing analysis such as false path detection carry into statistical approaches, the latter also introduce their own set

of challenges. In particular, deterministic timing analysis relies on two operations for propagating timing through a network, sum and max. The summing operation adds arrival times at inputs of gates to delays from those input pins to the output. The max operation decides which of these to propagate for max frequency analysis. Performing these calculations on pdfs is more expensive computationally than their counterparts in the deterministic case. Moreover, the degree of correlation between two pdfs arriving at a gate's inputs due to reconvergent fanouts needs be taken into account for accurate calculations.

In the past few years statistical techniques for timing analysis of digital circuits have received tremendous focus with representative works including [2, 12, 34, 47]. A recent paper [9] reviews many of the recent developments in SSTA. It discusses its underlying models and assumptions, then surveys the major approaches, and closes by discussing its remaining key challenges. It also has a large number of references which constitute a compendium of recent publications on statistical techniques in timing analysis and optimization.

## 2.3 SSTA Overview

This section provides an overview of statistical timing analysis based on [9]. For more in-depth treatments, the reader is referred to the recent overview paper [9] and the references that paper cites.

### 2.3.1 Introduction to SSTA

Traditional timing analysis abstracts a timing graph from a combinational circuit as follows. The nodes of the timing graph represent primary inputs/outputs of the circuit and gate input/output pins. The edges of the timing graph represent the timing elements of the circuit, namely, the gate input-output-pin delay and wire delay from a driver to a receiver, as shown in Figure 2.1.

The weight on these edges represents the delay of the corresponding timing element. For a combinational circuit, it is convenient to connect all primary inputs to a virtual source node with virtual edges having weight equal to the input arrival times. Similarly, all the primary outputs are connected to a virtual sink node through virtual edges with weights representing the required arrival times. The resulting timing graph, therefore, has a single source and sink node.

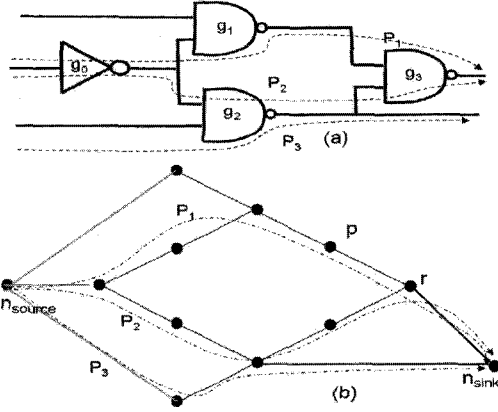


Figure 2.1: Mapping from circuit to timing graph for timing analysis.

SSTA uses the same fundamental concept but uses random variables (RVs) to model gate delays. The random variables capture the uncertainty introduced by the manufacturing variations which are prevalent in deep submicron technologies. A formal definition of statistical timing analysis follows.

**Definition 1.** A timing graph  $G = (N, E, n_s, n_f)$  is a directed graph having exactly one source node  $n_s$  and one sink node  $n_f$ , where  $N$  is a set of nodes, and  $E$  is a set of edges. The weight associated with an edge corresponds to either the gate delay or the interconnect delay. The timing graph is said to be a statistical timing graph if  $i$ th edge weight  $d_i$  is an RV.



In traditional DSTA, the most basic goal of the analysis is to find the maximum delay between the source node and the sink node of a timing graph, which is the delay of the longest path in the circuit. When modeling process-induced delay variations, the sample space is the set of all manufactured dies. In this case, the device parameters will have different values across this sample space, hence the critical path and its delay will change from one die to the next. Therefore, the delay of the circuit is also an RV, and the first task of SSTA is to compute the characteristics of this RV. This is performed by computing its probability-distribution function (PDF) or cumulative-distribution function (CDF) (see Figure 2.2). Alternatively, only specific statistical characteristics of the distribution, such as its mean and standard deviation, can be computed.

Note that the CDF and the PDF can be derived from one another through differentiation and integration. Given the CDF of circuit delay of a design and the required performance constraint the anticipated yield can be determined from the CDF. Conversely, given the CDF of the circuit delay and the required yield, the maximum frequency at which the set of yielding chips can be operated at can be found.

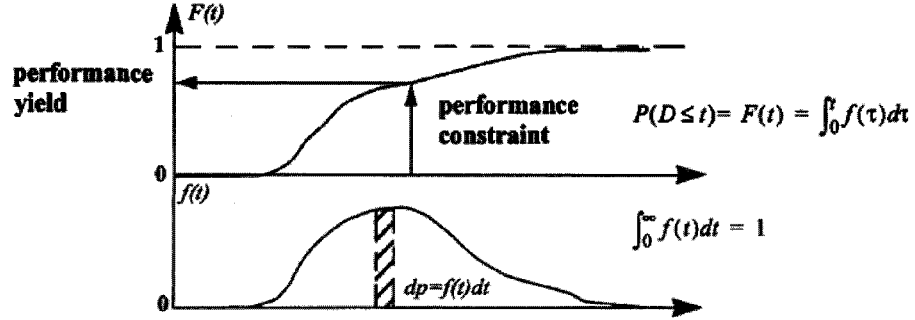


Figure 2.2: Examples of cumulative and probability distribution functions for a circuit's timing.

### 2.3.2 Challenges and Assumptions in SSTA

This section goes through various underlying assumptions and challenges pertaining to usage of SSTA in digital circuits

- *Gates versus wires*: Most literature to date presumes that gates are far more susceptible to variations than interconnects. There is some evidence for this in what little real silicon data has surfaced. As such, in this work, we will also stay with this assumption, modelling gate delays as random variables and ignoring wire delays as they do not impact our analysis or optimization

- *Normal distributions:* Again, we find that most literature assumes that gate delays can be represented by random variables. As [9] notes, normal or Gaussian distributions are found to be the most commonly observed distributions for RVs, and a number of elegant analytical results exist for them in the statistics literature. Hence, most of the research in SSTA assumed normal distributions for physical device parameters, electrical device parameters, gate delays, and arrival times. However, some physical device parameters may have significantly nonnormal distributions. Moreover, one of the two operations dominant in timing analysis, the max operator, is nonlinear and produces a nonnormal distribution when applied to two normal distributions. Nonnormal delay and arrival-time distributions introduce significant challenges for efficient SSTA.
- *Correlation:* Due to reconvergent fanouts from the same gate, inputs arriving at a given gate may have some common sources. This must be accounted for if we are to produce exact timing analysis. As per [9], the input arrival times at the reconvergent node become dependent on each other because of the shared edge delay. This dependence leads to so-called topological correlation between the arrival times and complicates the maximum operation at the reconvergent node. To perform accurate analysis, the SSTA algorithm must capture and propagate this correla-

tion so that it is correctly accounted for during the computation of the maximum function.

## **2.4 SSTA-based Circuit Optimization: Problem Overview**

This section will provide an overview of the problem we propose to solve and provide a mathematical formulation and motivation for this research direction.

### **2.4.1 Problem Formulation**

The starting point for our problem is a technology mapped digital circuit. Without loss of generality, this paper focuses on combinational circuits. We ignore interconnect delay though accounting for them can be readily accommodated. This is in line with other published literature and practical findings, as analysis of manufactured circuits indicates devices are much more susceptible to manufacturing variations than interconnect nets.

Our method uses discrete probability distribution functions (pdfs) throughout.

**Definition 2.** *A discrete pdf for random variable  $X$  is defined as one or more*

points where

$$f(x) = \Pr(X = x) \quad (2.1)$$

The mean and variance of a discrete random variable are given by

$$\mu_X = \sum x_i f(x_i) \quad (2.2)$$

$$\sigma_X^2 = \sum (x_i - \mu_X)^2 f(x_i) \quad (2.3)$$

We shall also use the cumulative distribution function (cdf) to illustrate concepts and results. The cdf for a discrete random variable  $X$  is defined as

$$F(x) = \Pr(X \leq x) \quad (2.4)$$

We assume that every gate delay in the circuit is represented by a normally distributed random variable which is consistent with other published literature. In line with other researchers, we focus our work on gate delays and sizes and ignore second order factors such as slope propagation or capacitance variations. We shall have more to say about modeling of transistor variations in the conclusions chapter.

Both the mean and standard deviation of delay through a bigger gate are less than those of a smaller gate. Arrival times are propagated throughout the circuit as pdfs.

We define the unconstrained timing variance minimization problem for a circuit as

$$\text{Minimize } \sigma_O^2 \tag{2.5}$$

where

$$\mu_O = \text{Mean} (RV_O)$$

$$\sigma_X^2 = \text{Variance} (RV_O)$$

$$RV_O = \text{Max}_{i \in OUT} (RV_i) \text{ where the Max is the statistical Max operator on random variables}$$

$$RV_i = \text{Random variable representing propagated arrival time of output } o_i$$

$$OUT = \{o_1, o_2, \dots, o_N\} \text{ are the circuits outputs}$$

As we shall see later, due to the gain-based nature of the algorithm we propose, a constrained version is possible by terminating it once certain constraints are satisfied. From this point onwards, we shall focus on the unconstrained problem without loss of generality. However, in the course of making local optimization, we show how a user-defined weight multiplier can in fact steer the optimization towards different goals.

We note that the random variable  $RV_O$  characterizes the mean and variance of the entire circuit. It should be highlighted that a circuit may have multiple outputs with close mean delays but different variances. In this case, all such

outputs will contribute to the overall variance  $\sigma_X^2$  of the circuits performance. Alternatively, an output with the highest variance may have a much smaller mean than other outputs and reducing its variance will have minimal effect on overall variance of the circuits performance. Any algorithm that attempts to alter  $RV_O$  must account for both means and variances of delays simultaneously.

Figure 2.3 gives PDF plots of  $RV_O$  at different optimization points while Figure 2.4 provides the equivalent plots using CDFs instead of PDFs. The original line represents a pdf obtained by optimizing a circuit with a goal of minimizing the mean of the longest delay in the circuit. Such a circuit will typically exhibit the widest spread in performance due to high usage of smaller devices which exhibit more manufacturing variability. Depending on target application of circuit, such a performance variance around the center can represent undesirable uncertainty that should be minimized. In [48] reduction of uncertainty was shown to be a key strategy for designing leading edge industrial designs. Decreasing variance can increase the overall yield of a design. An example of this is *optimization 1* in Figure 2.3 which yields more functional units at period  $T$  relative to the original design. However, our technique is quite general and is not limited to yield optimization.

Decreasing performance variance is also desirable on several other accounts even if it means relaxing the original timing targets. For example, cir-

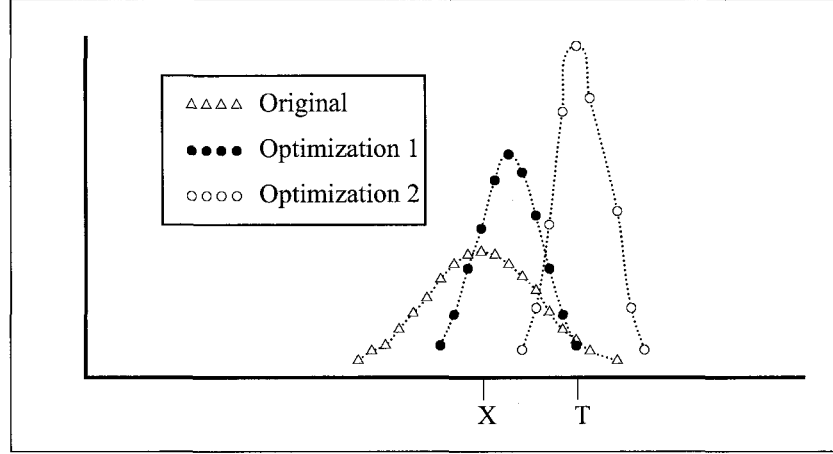


Figure 2.3: Example of circuit output Delay PDFs

circuits on the original curve to the left of  $X$  in Figure 2.3 below will exhibit undesirable variance in power consumption due to both dynamic and leakage power variations. These variations in turn contribute uncertainties in thermal dissipation and reliability verification. The effects of such performance variations can adversely product qualification and time-to-market. In such instances, the second optimization design criteria shown in Figure 2.3 labeled *optimization 2* can be more desirable due to better tolerance to manufacturing variations. Our research is aimed at providing designers with a statistically aware gate sizing methodology that allows arbitrary tradeoffs between mean and variance of  $RV_O$ .



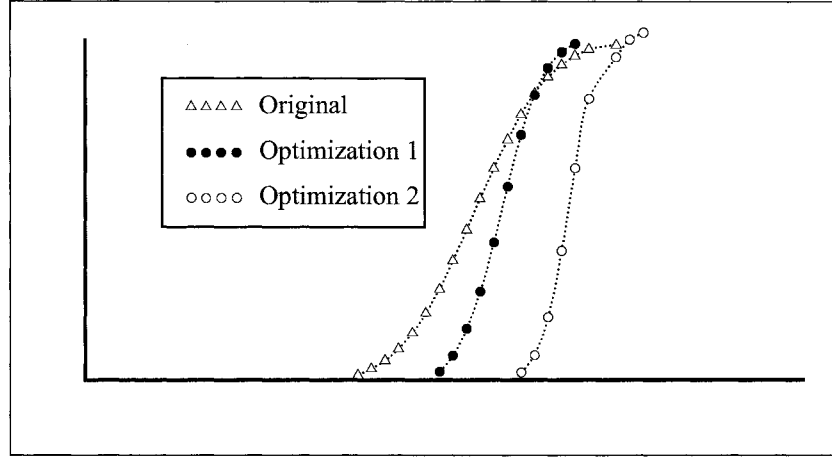


Figure 2.4: Example of circuit output Delay CDFs

#### 2.4.2 Overview of Research

We propose a gate sizing algorithm that uses statistical delay models for gates. We propose to extend local gate sizing approaches in such a way that they can be run in a statistical mode. Our choice of using a local gain-based sizing approach is based on our experience with real-life libraries that are almost exclusively characterized as lookup-tables which use input-slope/output-capacitance as inputs and produce output-slope/delay-across-gate. Such libraries do not lend themselves to accurate modeling with analytical formulae. Our specific objective with this sizing is to improve the reliability of the final circuit by reducing the spread of uncertainty in the timing model as produced

by statistical static timing analysis (SSTA).

The primary focus of our research is on reducing the variations of a digital circuit as measured by statistical static timing analysis (SSTA) before manufacturing. Our success criteria is reduction of  $\sigma$  of the SSTA distribution measured at a virtual sink of all the outputs of the circuits. An obvious way to reduce variations would be to prohibit usage of all devices smaller than a given dimension as smallest devices exhibit the most variation. However, this is not a very practical approach, since usage of these devices for non-critical paths saves both area and power. Instead, we show that variations in such devices can in fact be canceled out by appropriately sizing up subsequent gates with tolerable increases in circuit area.

Regular gain-based sizing algorithms operate on the worst negative slack (WNS) paths, continuously updating them as the algorithm proceeds. Our approach introduces concept of worst negative statistical slack paths (WNSS). These paths are the statistical counterpart of well known WNS paths, except that they track both mean and variance of delay. Our research enables a designer to choose appropriate tradeoffs between mean and variance of delay for a given circuit.

Optimization engines typically use different timing engines for opti-

mization versus final analysis. The core of an optimization algorithm requires a fast engine for evaluating sizing or other optimization choices. We expect that a side-product of our research will be a fast engine for performing statistical static timing analysis on small circuit segments. We use a more accurate but slower engine for analysis and tracking of WNSS paths which relies on the sampled PDF for propagation of timing edges while keeping the faster engine which uses point values for mean and standard variation for the core of the optimization engine.

## **2.5 Statistical Gate Sizing**

Our research in this area combines statistical techniques as well as circuit optimization using gate sizing. This section will provide an overview of the proposed algorithm, develop the mathematical apparatus needed for algorithm implementation, present experimental results and analysis thereof. We also highlight benefits of research in context of design automation.

### **2.5.1 Overview of Algorithm**

We studied several deterministic sizing techniques to evaluate their fitness as a basis for statistical sizing. Our preference for accurate gate delay

models steered us away from methods [13, 37, 58], which require convex analytical expressions for gate delays. Such models not adequately capture the nonlinearities in current and foreseeable DSM technologies where manufacturing variations are prevalent. The main procedure of our approach is shown in Figure 2.5, with supporting function shown in Figure 2.6. This builds on the deterministic algorithms presented in [19, 40] which are quite versatile and form basis of commercially available optimization tools. The next sections show how we deal with new challenges that arise when timing constraints are represented by random variables.

### **2.5.2 FULLSSTA : Full Statistical Static Timing Analysis**

Our full statistical analysis engine is based on [34]. This approach discretizes pdfs at a user controlled sampling rate. We used 10-15 samples per pdf as a reasonable tradeoff between accuracy and speed. Note that the discrete PDFs are renormalized after sampling to ensure that the sum of the probabilities for the discrete events is equal to one. An example of a discrete pdf for delay is given in Figure 2.7

The operations sum and max are performed on discrete pdfs using shifting, scaling, and min/max reduction. An example of this process is shown

```

1: procedure STATISTICAL SIZER(Circuit C)
2:   repeat
3:     Run FULLSSTA on C
4:     Trace critical path (WNSS) of C           ▷ WNSS is dynamic
5:     foreach  $g \in$  (gates on WNSS)
6:       extract subcircuit S around g
7:        $SB = \text{Cost}(S)$ 
8:        $GC = \text{CurrentSize}(g)$ 
9:        $GB = GC$ 
10:      foreach  $I \in$  (sizes of g)
11:         $g \text{ in } S \leftarrow I$ 
12:         $SN = \text{Cost}(S)$ 
13:        if  $SN < SB$  then
14:           $GB = I$ 
15:           $SB = SN$ 
16:          if  $GB \neq GC$  then           ▷ Better size was found
17:             $g.\text{nextSize} \leftarrow GB$    ▷ Schedule g for resizing
18:      Resize scheduled gates
19:   until constraints met or no further improvement

```

Figure 2.5: Overview of Statistical Sizer Algorithm

- 1: **function** COST(Subcircuit S)
- 2:     Perform FASSTA on S
- 3:     Return ObjectiveFunction(S)

Figure 2.6: Extracting Subcircuit Cost for Statistical Sizer

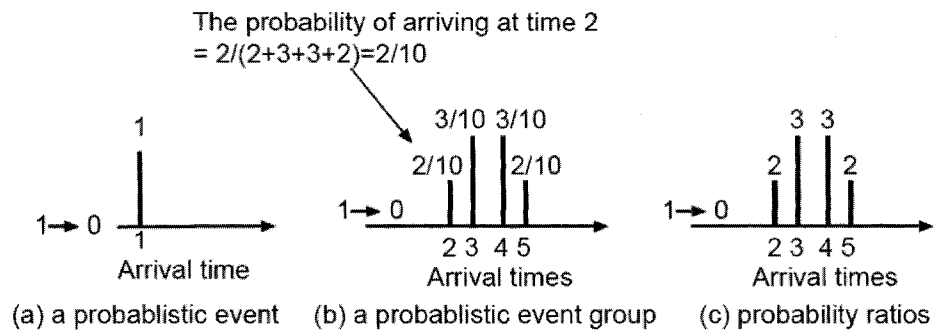


Figure 2.7: Probabilistic event representing delay at a given edge in an SSTA timing graph

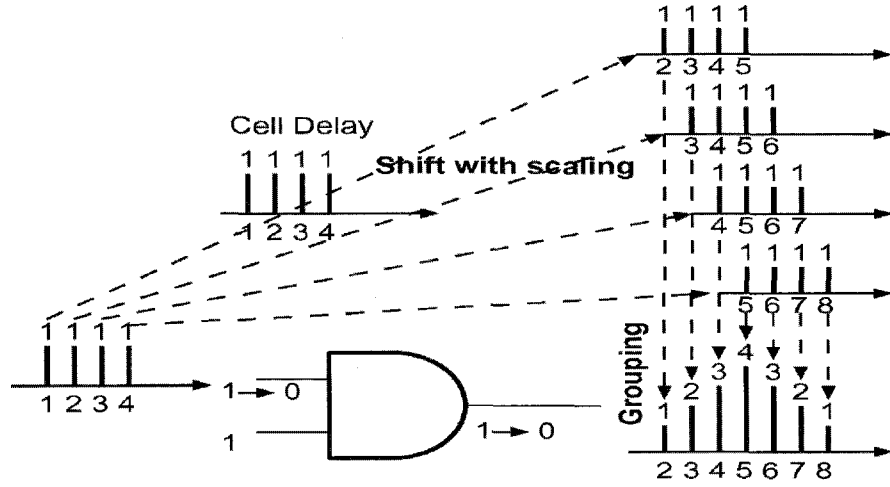


Figure 2.8: Shift with scaling and grouping techniques to perform convolution of input and gate-delay PDFs to compute the output-delay PDF

in Figure 2.8.

The approach utilizes discrete sum and maximum operations for arrival-time propagation. In the case of a degenerate or deterministic input-delay distribution, the sum operation is simple, and the output-delay PDF is obtained by simply shifting the gate-delay distribution by the input delay. However, in the case where the input-delay PDF is nondegenerate, a set of shifted output-delay distributions is generated, as shown in [34]. Each of these shifted PDFs corre-

sponds to a discrete event from the input-delay PDF. This set of shifted PDFs is then combined using Bayes theorem the shifted PDFs are first scaled, where the scaling factor is the probability of the associated discrete input event. The scaled events are then grouped by summing the probability at each of the discrete time points. The actual probability of an event can be obtained by dividing the total value for each discrete point of the PDF by the sum of the numbers corresponding to all the events in each discrete PDF. The overall computation can be expressed as

$$f_s(t) = \sum_{i=-\infty}^{i=\infty} f_x(i)f_y(i-t) = f_x(t) \otimes f_y(t) \quad (2.6)$$

where  $s = x + y$ , and implies that the PDF of the sum of two RVs can be expressed as a convolution of their PDFs. The statistical maximum is computed using the relation

$$f_z(t) = F_x(t)f_x(t) + F_y(t)f_y(t) \quad (2.7)$$

where  $z = \text{maximum}(x, y)$ ,  $f$  and  $F$  represent the PDF and CDF of the RV, respectively, and  $x$  and  $y$  are assumed to be independent. The previous equation expresses mathematically that the probability that the maximum of two discrete RVs has a value  $t_0$  is equal to the probability that one of the RVs has a value equal to  $t_0$  and the other has a value less than or equal to  $t_0$ .



In addition to propagating pdfs, we also calculate the mean and variance at every node and store these values for use in the fast timing engine (FASSTA). This component in our algorithm can be updated as needed to track the latest emerging research in statistical timing analysis and represents the outer loop for our iterations.

### **2.5.3 FASSTA : Fast Statistical Static Timing Analysis**

As we pointed out in 2.3, statistical analysis methods such as FULLSSTA are expensive and impractical for use alone in an optimization setting. This section presents new approximations for fast statistical static timing analysis (FASSTA). This allows us to quickly evaluate costs of subcircuits in the body of the optimization algorithm. The two operations needed in static timing analysis are sum and max. The FASSTA engine relies on the point values for means and delays calculated in FULLSSTA rather than the complete discrete pdf representations.

We start with two normally distributed independent random variables  $A$  and  $B$  with expected values  $\mu_A$  and  $\mu_B$  and with variances  $\sigma_A^2$  and  $\sigma_B^2$  respectively. Let random variable  $C$  be the sum of  $A$  and  $B$ . The mean and variance

of  $C$  are given by:

$$\mu_C = \mu_A + \mu_B \quad (2.8)$$

$$\sigma_C^2 = \sigma_A^2 + \sigma_B^2 \quad (2.9)$$

To calculate the max, we shall expand on the formulation in [15]. We use the following notation:

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \quad (2.10)$$

$$\Phi(x) = \int_{-\infty}^x \varphi(t) dt \quad (2.11)$$

$$a^2 = \sigma_A^2 + \sigma_B^2 \quad (2.12)$$

$$\alpha = \frac{\mu_A - \mu_B}{a} \quad (2.13)$$

The first two moments of  $\max(A, B)$  are given by

$$v_1 = \mu_A \Phi(\alpha) + \mu_B \Phi(-\alpha) + a\varphi(\alpha) \quad (2.14)$$

$$v_2 = (\sigma_A^2 + \sigma_B^2)\Phi(\alpha) + (\sigma_A^2 + \sigma_B^2)\Phi(-\alpha) + (\mu_A + \mu_B)a\varphi(\alpha) \quad (2.15)$$

The variance of  $\max(A, B)$  is given by

$$Var_{\max(A, B)} = v_2 - v_1^2 \quad (2.16)$$

These formulae cannot be evaluated directly because the integrals do not have analytical expressions. We found them to be expensive to compute numerically. Instead, we derive next an original approximation on how they can be avoided altogether and show bounds for the magnitude of approximation error. We reformulate the integral:

$$\Phi(x) = \int_{-\infty}^x \varphi(t)dt \quad (2.17)$$

$$\Phi(x) = \int_{-\infty}^0 \varphi(t)dt + \int_0^x \varphi(t)dt \quad (2.18)$$

$$\Phi(x) = \frac{1}{2} + \frac{1}{2}erf\left(\frac{x}{\sqrt{2}}\right) \quad (2.19)$$

where  $erf$  denotes the error function. To calculate the error function, we use the following quadratic approximation [59] which is accurate to two decimal places

$$\frac{1}{2}erf\left(\frac{x}{\sqrt{2}}\right) \cong \begin{cases} 0.1x(4.4 - x) & 0 \leq x \leq 2.2 \\ 0.49x & 2.2 < x < 2.6 \\ 0.50x & \geq 2.6 \end{cases} \quad (2.20)$$

We also note that the error function is odd:

$$erf(-x) = -erf(x) \quad (2.21)$$

These formulae give us a quick method to approximate the error function for any value. We substitute this approximation in Eq 2.15. We note that if

$$\alpha = \frac{\mu_A - \mu_B}{a} \geq 2.6 \quad (2.22)$$

then

$$\Phi(\alpha) \approx 1 \quad (2.23)$$

$$\Phi(-\alpha) \approx 0 \quad (2.24)$$

$$\varphi(\alpha) \approx 0 \quad (2.25)$$

and we have

$$v_1 = \mu_A \quad (2.26)$$

$$v_2 = \mu_A^2 + \sigma_A^2 \quad (2.27)$$

which gives

$$Mean_{max(A,B)} \approx \mu_A \quad (2.28)$$

$$Var_{max(A,B)} \approx \sigma_A^2 \quad (2.29)$$

Similarly, for

$$\alpha = \frac{\mu_A - \mu_B}{a} \leq 2.6 \quad (2.30)$$

then

$$Mean_{max(A,B)} \approx \mu_B \quad (2.31)$$

$$Var_{max(A,B)} \approx \sigma_B^2 \quad (2.32)$$

We observed that in the vast majority cases, one of Eq 2.22 or Eq 2.30 would apply obviating need for any calculation for max, while in other cases the approximations above provide quick estimates. These formulae assume independence of random variables which does not always hold. However, this approach emphasizes speed while retaining a reasonable degree of accuracy for small subcircuits. We stress that this approach is only used for the inner loop of the optimizations, while the outer loop relies on the more accurate discrete pdfs manipulation approach that can track correlations due to reconvergent paths using Principal Component Analysis [12] or other methods as long as runtime is managed appropriately.

### 2.5.3.1 Statistical Critical Path Identification

As was pointed out in Section 2.2.2, circuit optimization engines typically focus their effort on the critical or WNS path to improve the performance

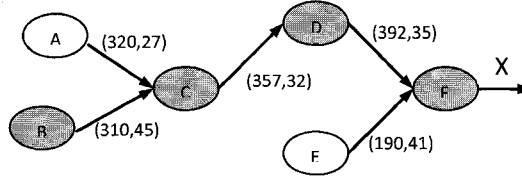


Figure 2.9: Tracing worst negative statistical slack (WNSS) path. Numbers in parenthesis are  $(\mu, \sigma)$  of arrival time. The shaded nodes indicate the WNSS using our method.

of the circuit. This section describes how we extend this concept to trace the Worst Negative Statistical Slack (WNSS) path in a circuit.

Consider a circuit consisting of 6 gates such as the one shown in Figure 2.9. The first number in the parenthesis represents the statistical mean of delay for that arc while the second one represents the standard variation. We wish to determine the critical path with the biggest contribution to the variance at the output of node  $X$ . We note that, unlike the deterministic case, one cannot simply pick the input with the higher mean or variance to determine which input is most responsible for the variance at the output. This is due to the non-linearity of the statistical max operation where all inputs contribute to

the output max. This is in contrast with deterministic max operation where only the maximum of the inputs contributes to the output.

We proceed to solve this problem by considering the sensitivity of the variance at the output of a node with respect to the inputs as follows. Starting from a given gate, we compare its inputs pair-wise. If either of Eq 2.22 or Eq 2.30 are satisfied, then we pick the input with the higher mean as clearly having the dominant influence on the output of this gate. If neither of these equations is satisfied, we compare

$$\frac{\partial Var_{max(A,B)}}{\partial \mu_A} \quad (2.33)$$

versus

$$\frac{\partial Var_{max(A,B)}}{\partial \mu_B} \quad (2.34)$$

Our justification for taking the partial derivatives with respect to the means of the delays is that the variances have a random component not under our direct control. On the other hand, using available gate sizes for a given circuit function gives us a direct ability to control means of delays.

One approach to obtaining these sensitivities is to differentiate Eq 2.33-Eq 2.34 directly. We found the resultant expressions to be complex requiring expensive floating-point computations not suited for the core of our optimization engine. Instead, we chose to use an approximation for differentiation as



follows. Rewriting

$$Var_{max(A,B)} = f(\mu_A, \mu_B, \sigma_A, \sigma_B) \quad (2.35)$$

We use a forward finite-difference formula to approximate the partial derivative:

$$\frac{\partial Var_{max(A,B)}}{\partial \mu_A} \approx \frac{f(\mu_A + h, \mu_B, \sigma_A + g, \sigma_B) - f(\mu_A, \mu_B, \sigma_A, \sigma_B)}{h} \quad (2.36)$$

We used values for  $h$  of the order of 1% of the mean. It should be noted that  $\mu$  and  $\sigma$  along a given path are correlated and one cannot expect to change one value without the other being impacted. The change in  $\sigma_A$  that can result out of altering  $\mu_A$  is indicated by  $g$ . We also note that it is impossible in general to determine  $g$  accurately as the relationship between  $\mu$  and  $\sigma$  along a given path is governed by a combination of gate performance variations inversely proportional to their dimensions as well unsystematic random variations that are unpredictable. For purposes of ranking inputs, the following linear approximation linking these two was found to be adequate:

$$g \cong \Delta\sigma \cong c\Delta\mu \quad (2.37)$$

We used values for  $c$  equal to those assumed to relate mean delay through a gate to its variance.

### 2.5.3.2 Subcircuit extraction and ranking

For every gate being evaluated for resizing, our algorithm extracts a subcircuit around this gate based on a user-controlled depth. We have found that using two levels of transitive fanins and fanouts is sufficiently accurate without being too costly to evaluate. However, this is one of the many knobs that can be altered at will as a tradeoff between runtime and accuracy.

For every available size for this gate, we use FASSTA to calculate mean and variance of delay at the outputs of this subcircuit. We derive a cost function that allows us to rank the the relative merits of gate sizing in this subcircuit quickly as follows. For all outputs of the subcircuit  $O_1 \dots O_n$ , we calculate a weighted sum of mean and standard variation:

$$Cost(O_i) = \mu_i + \lambda\sigma_i \quad (2.38)$$

where  $\lambda$  is a user-specified weight multiplier that ranks relative importance of minimizing standard variation against mean of delay. By choosing higher values for  $\lambda$ , the user can place more emphasis on variance reduction. We provide more analysis on effect of varying  $\lambda$  in the conclusions section at the end of the paper. The cost of the subcircuit is given by the maximum of  $Cost(O_i)$  across all outputs. We then pick the gate size that minimizes subcircuit cost across all gate sizes for candidate gate.

#### 2.5.4 Experimental results

The approach introduced above was implemented in Java and run on an Intel PC running at 2.53 GHz. We tested the algorithm on various circuits from the ISCAS benchmarks and various sized ALU circuits.

The circuits were first synthesized using Design Compiler [57] using an industrial 90nm lookup-table based standard cell library with 6-8 sizes per gate type. In line with other researchers, we added variations to the gate delays based on [16, 43]. Two variations components were added to the gate delays: one proportional to delay through gate and another random source corresponding to unsystematic manufacturing variations.

Table 2.1 shows the results of our optimization for two representative multiplier values,  $\lambda = 3$  and  $\lambda = 9$ . The ratio of  $\sigma$  to  $\mu$  obtained by optimizing for mean delay is shown in the first column entitled original. We then ran our algorithm at various values for multiplier  $\lambda(7)$ . Results are shown for optimization under two different values for  $\lambda$ , 3 and 9. We observed that increasing  $\lambda$  any further could not yield further reduction in variance in general though the highest value for  $\lambda$  was different for different circuits. This is due to the unsystematic variations whose effects cannot be totally eliminated regardless of gate sizes deployed.

Table 2.1: Experimental Results:  $\lambda = 3$

Circuit		Original		$\lambda = 3$ , runtime is in minutes			
Name	Gates	$\sigma/\mu$	$\Delta\sigma$	$\Delta\mu$	$\sigma/\mu$	$\Delta A$	Runtime (mins)
alu1	234	0.124	+4%	-54%	0.055	+16%	1.5
alu2	161	0.147	+3%	-71%	0.041	+14%	1.3
alu3	215	0.127	+7%	-61%	0.046	+16%	1.5
c432	203	0.093	+2%	-58%	0.038	+11%	1.6
c499	381	0.077	+5%	-63%	0.027	+13%	1.5
c880	301	0.092	+4%	-57%	0.038	+17%	1.5
c1355	378	0.081	+5%	-63%	0.057	+13%	1.7
c1908	563	0.076	+3%	-44%	0.041	+7%	3.7
c2670	820	0.068	+2%	-42%	0.039	+11%	9.8
c3540	1245	0.062	+4%	-56%	0.026	+12%	14.7
c5315	2318	0.043	+2%	-36%	0.027	+12%	36
c6288	2980	0.021	+1%	-28%	0.015	+5%	44
c7552	2763	0.043	+2%	-50%	0.021	+11%	31

Table 2.2: Experimental Results:  $\lambda = 9$ , runtime is in minutes

Circuit		Original					$\lambda = 9$	
Name	Gates	$\sigma/\mu$	$\Delta\sigma$	$\Delta\mu$	$\sigma/\mu$	$\Delta A$	Runtime (mins)	
alu1	234	0.124	+6%	-80%	0.023	+24%	1.6	
alu2	161	0.147	+4%	-86%	0.020	+29%	1.4	
alu3	215	0.127	+9%	-75%	0.029	+25%	1.7	
c432	203	0.093	+4%	-75%	0.022	+21%	1.7	
c499	381	0.077	+8%	-76%	0.017	+21%	1.8	
c880	301	0.092	+5%	-79%	0.018	+23%	1.7	
c1355	378	0.081	+7%	-71%	0.022	+19%	1.9	
c1908	563	0.076	+4%	-71%	0.021	+16%	3.8	
c2670	820	0.068	+7%	-76%	0.015	+18%	9.1	
c3540	1245	0.062	+8%	-70%	0.017	+21%	13.1	
c5315	2318	0.043	+7%	-68%	0.013	+15%	34	
c6288	2980	0.021	+2%	-47%	0.011	+9%	41	
c7552	2763	0.043	+4%	-66%	0.014	+17%	33	

Figure 2.10 shows a plot of  $\mu$  against  $\sigma$  for various values of  $\lambda$  for circuit C432. As the multiplier  $\lambda$  is increased, the mean is increased in exchange for a gradual reduction in standard variation of delay across the circuit.

Several observations can be made from these results. Our algorithm consistently reduces the standard variation while increasing mean delay and area. This behavior is expected since our algorithm favors bigger gate sizes that reduce the variance of delay across them. The algorithms focus on minimizing variance also causes it to upsize gates near the outputs to reduce the overall variance at circuits output. This is done even if that path does not have the highest mean delay which is in contrast to a worst mean-delay optimizer which would not upsize such gates. This increases overall delay due to higher loading slowing down predecessor gates.

Another important observation is that the number of gates along a timing path is inversely proportional to the variance along that path and the ability to optimize it away. Paths with a shorter number of gates tend to be more susceptible to variations. The smaller ALU circuits exhibit significant variations as a percentage of their mean. Our algorithm can reduce this variation substantially but at a higher increase in area. On the other hand, circuit C6288 which is a  $16 \times 16$  bit multiplier has the longest depth of any of the circuits in the table. We note that it has the lowest improvement due to its already low  $\sigma$  to  $\mu$  ratio.

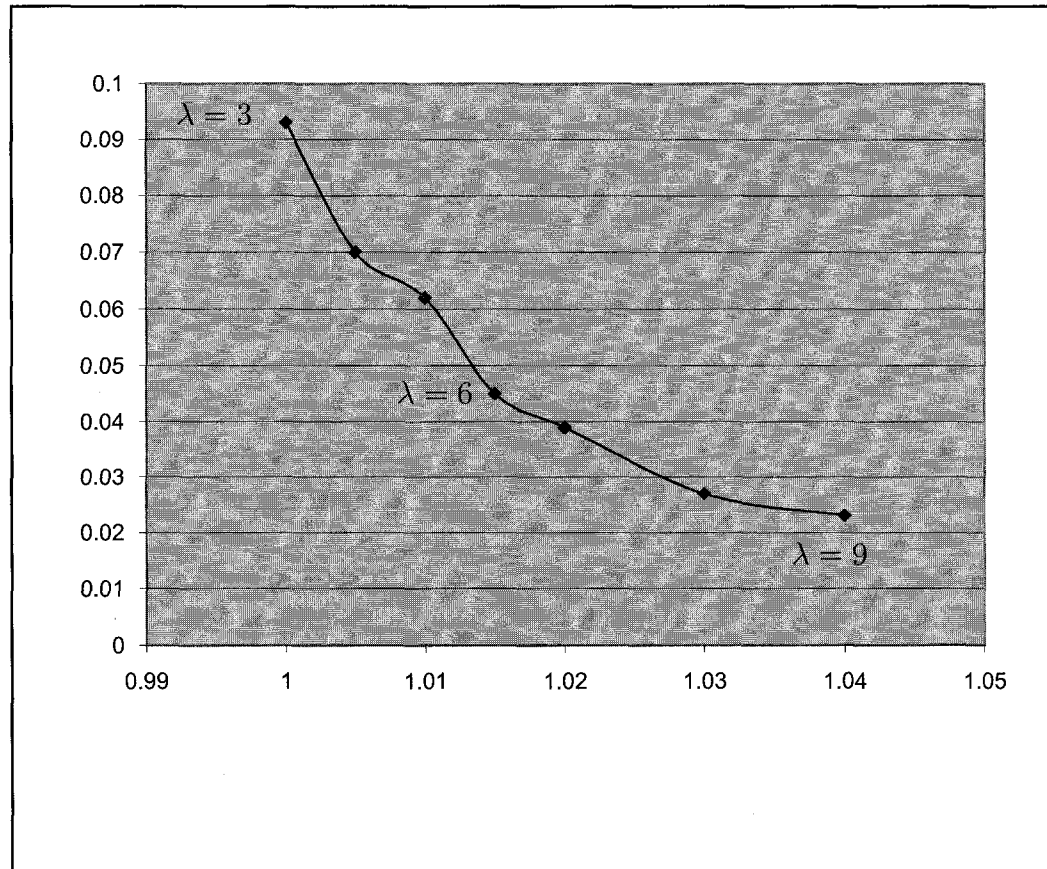


Figure 2.10: Normalized Mean-Std Variation for C432 at different  $\lambda$ . The x-axis shows the mean while the y-axis has the standard variation.

### **2.5.5 Concluding Remarks**

We introduced a new concept of a worst negative statistical slack path and derived a procedure for tracing and optimizing such paths. In the process, we also derived a new approximation for the max operation on random variables for use in circuit optimization. Our approach allows us to steer the optimization process towards different mean-variance goals. The significance of this work is that it can be used during design cycle to increase tolerance for the effects of manufacturing variations by trading off circuit delay and area requirements for reduced timing variance with user controlled weights. We demonstrated efficacy of our approach on ISCAS benchmarks with consistent variance reduction in exchange for moderate increases in area and low increases in mean delays.

### **2.5.6 Benefits of Research**

As previously mentioned, our research can be deployed where predictability of performance of a manufactured circuit is of paramount importance. Manufacturing variations from pre-silicon PV models causes variance in power consumption due to both dynamic and leakage power variations. These variations in turn contribute uncertainties in thermal dissipation and reliability



verification. The effects of such performance variations can adversely product qualification and time-to-market. In such instances, our proposed optimization becomes more desirable due to better tolerance to manufacturing variations. Our research is aimed at providing designers with a statistically aware gate sizing methodology that allows arbitrary tradeoffs between mean and variance of the delays across a circuit. It can be seen as adding a third tradeoff vector in addition to the well known area-timing tradeoffs designers work with.

## **2.6 Summary**

Most recent deep submicron manufacturing technologies exhibit both inter- and intra-die variations, some of which are systematic and others which are random. The aggregate of these variations poses a significant challenge for circuit designers, who can either make worst case assumptions on all design axes such as delay, area, and power which severely limit the design space and may corner it into a wrong design point. Alternately, designers can use mean values of delays in the design phase with an expectation of widespread variations in silicon performance. Our proposed research is targeted to help designers navigate the available design space by using statistical models in the analysis and optimization of the circuit before it reaches silicon. We introduce

a new concept of a worst negative statistical slack path and plan to derive a procedure for tracing and optimizing such paths. In the process, we also derive a new approximation for the max operation on random variables for use in circuit optimization. Our approach allows us to steer the optimization process toward different mean-variance goals. The significance of this work is that it can be used during design cycle to increase tolerance for the effects of manufacturing variations by trading off circuit delay and area requirements for reduced timing variance with user controlled weights. We demonstrated fidelity of our approach on ISCAS benchmarks with variance reduction in exchange for increases in area and low increases in mean delays.

## **Chapter 3**

# **An Efficient Algorithm for Analysis of Cyclic Circuits**

### **3.1 Introduction**

Cyclic circuits are those which contain loops or cycles within them. We will present a more formal definition later on in the chapter. An example of a cyclic circuit is shown in Figure 3.1. Cyclic circuits can be produced inadvertently during high-level synthesis from high level hardware languages such as ESTEREL [8]. They are also the most compact representation for certain circuits such as arbiters [50].

A key challenge for cyclic circuits is that correct operation is only guaranteed in specific cases. For certain input patterns, such circuits are well-behaved (functional), i.e., do not exhibit oscillations or state-holding behavior.

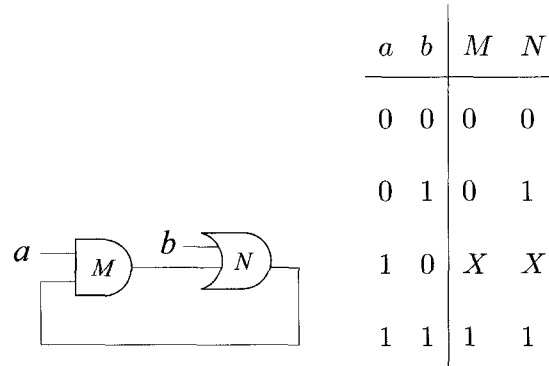


Figure 3.1: A trivial cyclic circuit and its truth table

Despite this, most circuit analysis tools forbid the presence of cycles. The central challenge of cyclic circuits is their data-dependent evaluation order, meaning their gates have no topological order. This causes difficulties for many tools such as static timing analyzers that rely on such a static order. Furthermore, applying regular logic simulation to these circuits is cumbersome.

Consider the small cyclic circuit in Figure 3.1. From its truth table, we see the circuit is well-behaved unless  $a = 1$  and  $b = 0$ . It is impossible to predict with certainty how the circuit behaves when presented with this pattern. For all other input patterns, the circuit behaves combinational because the feedback loop is broken by a controlling input on one of the gates.

We shall use the terminology of partial assignments for our exposi-

tion. A *partial assignment* is an assignment to one or more inputs to the loop;  $\{a = 0\}$  is one such partial assignment. Our proposed algorithm will produce a set of partial assignments that provide a concise representation of the conditions under which a cyclic circuit is well-behaved. For example, the set of partial assignments  $\{\{a = 0\}, \{b = 1\}\}$  constitutes necessary and sufficient conditions for combinational operation of the circuit in Figure 3.1: at least one of these must hold in order for the circuit to operate functionally.

In this research, we propose a novel algorithm that can rapidly identify all possible combinational behavior of a cyclic circuit. The algorithm takes a circuit containing one or more loops and produces a set of partial assignments that represent every condition under which the circuit behaves combinational. Our algorithm relies on the fact that gates such as ANDs and ORs have controlling inputs (0 and 1 respectively) that break feedback loops to aggressively prune the search space. The set of partial assignments our algorithm produces can be used to rule out non-constructive operation of circuits produced by high level compilers such as Esterel [6], or they can be used to create an equivalent acyclic circuit [22].

## 3.2 Notation and Definitions

This section defines the basic terminology needed for an exposition of material on cyclic circuits.

We represent circuits with a *directed graph (digraph)*. A digraph  $G$  is a pair  $(V, E)$  where  $V$  is a set of vertices and  $E$  is a set of edges. An *edge* is an element of  $V \times V$  with distinct vertices. We represent a circuit as a digraph whose vertices correspond to gates and whose edges correspond to nets. A *controlling value* for a gate  $G$  is the value that applied to any input of  $G$  uniquely determines  $G$ 's output independent of other inputs. To simplify our exposition, we only consider simple logic gates: NOT, AND/NAND, and OR/NOR. This is not a limitation as more complex gates can be represented as combinations of these gates. Loops or cycles are formally defined using graph theory in terms of strongly connected components.

**Definition 3.** *A strongly connected component (SCC) of a digraph  $G = (V, E)$  is a maximal subset of vertices  $C \subseteq V$  such that any vertex in  $C$  is reachable from any other vertex in  $C$ . Inputs of an SCC are inputs of gates that are part of the SCC that are not driven by gates inside the SCC.*

Figure 3.2 shows a circuit with a single SCC. Nets  $a$ ,  $b$ , and  $c$  are inputs to the SCC. When analyzing an input circuit, we first decompose it into SCCs

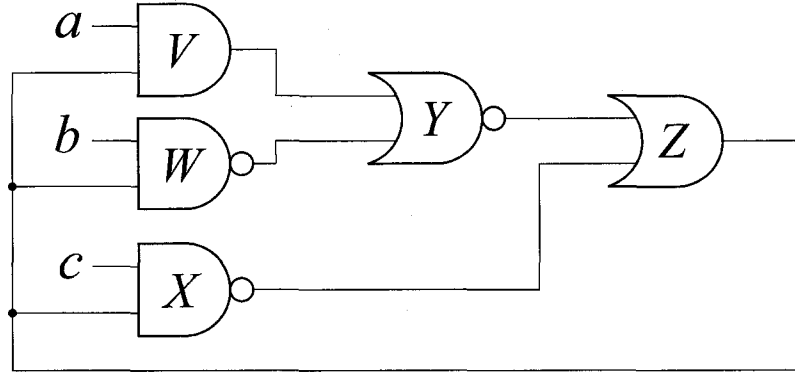


Figure 3.2: Cyclic circuit for illustrating definitions

using a standard algorithm [18]. We note that finding SCCs in a circuit is linear in the size of the circuit. If the input circuit contains more than one SCC, we consider each SCC separately in a topological order.

Our analysis methodology and logic simulation use a ternary domain consisting of  $\{0, 1, X\}$  where  $X$  denotes an unknown digital value.

**Definition 4** (Malik [35]). *A circuit is combinational for an input assignment if three-valued simulation starting with all internal nodes set to  $X$  resolves the output of every gate in the circuit to either 0 or 1 under the assignment.*

Literature on cyclic circuits also refers to this behavior as “well-behaved” and “constructive” [51]. Combinational behavior is equivalent to stating that the circuit behaves as if it were acyclic with no  $X$ ’s and no oscillations.

### 3.3 Literature Survey

An extensive review of prior work on cyclic circuits was undertaken. Below is a summary of contributions in this field covering period from 1960 to late 2005.

#### 3.3.1 Origins of Cyclic Circuits

Short [55] is earliest published work to suggest that cyclic structures can save area in relay networks. In 1970, Kautz [31] showed that the minimal form of certain circuits contained combinational loops. Rivest [50] came to a similar conclusion, suggesting that combinational loops are more than just a nuisance. Rivest's circuits were the first convincing example of cyclic circuits that were provably smaller than any equivalent acyclic versions. An example is shown in Figure 3.3. This circuit can be extended to use  $n$  inputs and produce  $2n$  unique outputs.

Stok [56] observed how cyclic circuits can arise from resource-sharing in high-level synthesis. An example of such a circuit is given in Figure 3.4. The function of this circuit can be described as

$$O = \text{if } (c) \text{ then } F(G(x)) \text{ else } G(F(x))$$

Note that  $G$  and  $F$  can be any operators such as shifters, adders, or



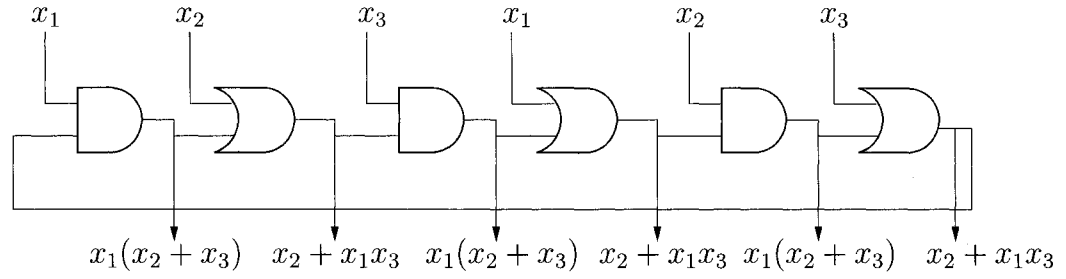


Figure 3.3: Rivest's Circuit

multipliers and may have additional operand inputs. Stok stipulated that an equivalent acyclic circuit would be always bigger due to need to duplicate operator circuits.

### 3.3.2 Analysis of Cyclic Circuits

Malik's work [35] on analyzing combinational circuits was the forerunner with respect to analysis of cyclic circuits. Malik showed an equivalence between combinational cyclic circuits and least-fixed-points in three-valued simulation. Shiple, Berry, and Touati [51] extended this idea and applied it to the Esterel language [6, 7], whose hardware translation [8] often produces combinational cycles. Their approach uses a symbolic state-space traversal followed by an  $O(n^2)$  replication procedure to remove cycles. Their enhancement to Malik's algorithm relies on Bourdoncle's [10] algorithm for reducing number

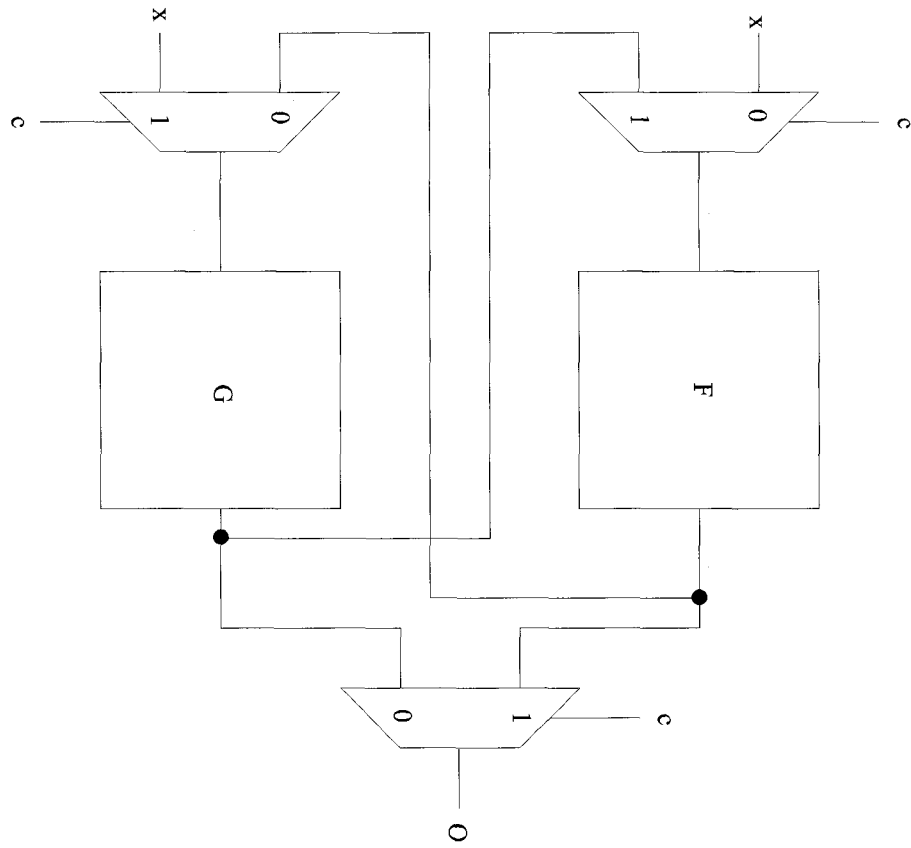


Figure 3.4: Cyclic circuit arising from resource sharing due to Stok [56]

of simulation iterations. However, Bourdoncle’s algorithm provides a general framework for static scheduling of strongly connected components. Our proposed algorithm pays more attention to both the structure and function of the circuit. We also believe that our proposed algorithm when coupled with the resynthesis technique of Edwards [22] will produce smaller circuits.

Shiple et. al. [52] also extended their analysis to combinational cycles within sequential circuits. The BDD-based algorithm of Halbwachs and Maraninchi [26] takes a brute-force approach, ignoring the structure of the circuit. Namjoshi and Kurshan [42] take a very different approach, showing that any fixed-point is interesting, not just the least. Their analysis merely answers whether a circuit is combinational.

### **3.3.3 Synthesis of Cyclic Circuits**

Recently, Riedel and Bruck [49] applied Rivest’s observations to synthesize very compact combinational circuits that contain cycles. As part of their synthesis step, they check whether the circuit they generated is combinational using a fairly expensive BDD construction; our proposed algorithm could potentially be used in that setting. More practically, the cyclic combinational circuits they generate have topologies complex enough to stymie the

de-cyclification algorithm of Edwards [22], which this work will build on.

### **3.3.4 Most Recent Publications on Analysis of Cyclic Circuits**

The algorithm of Edwards [22] for de-cyclification consists of two steps. The first step enumerates all combinational behavior in a cyclic circuit. The result of this step are necessary and sufficient conditions under which the circuit is well-behaved, or combinational. This search get exponentially slower as the circuits get bigger, and fails to terminate except on the smallest circuits. The second step in Edwards' algorithm collects the acyclic fragments implied by the first step and combines them into a single acyclic circuit.

An algorithm was proposed in [3] for combinationality checks. However, the algorithm presumes existence of an acyclic equivalent circuit apriori and merely checks for equivalence against this circuit. Our approach doesn't presume existence of such a circuit, and is in fact directed at producing an equivalent acyclic circuit.

Another approach that was presented in [24] infers level-sensitive latches to make cyclic circuit acyclic. This approach changes the semantics of the circuit from a combinational circuit to a synchronous one, and cannot claim to produce a drop-in equivalent circuit to the original version. There is a substan-

tial area penalty which the authors admit to being bounded by double the area of the original loop. In addition, the authors did not attempt their approach on more complex cyclic circuits such as those produced by CYCLIFY [49].

### 3.4 Types of Cycles

Cycles encountered in circuits might be divided into two types, which we shall call true cycles and false cycles.

True cycles are those where presence or absence of a logical cycle depends on the input vector into the circuit. The simple cyclic circuit in Figure 3.5 is an example of a true cycle. This cycle can be sensitized when  $a = 1$  and  $b = 0$ .

False cycles are those which only exist in a topological sense, but can never be sensitized electrically regardless of input pattern. An example is shown in Figure 3.5. Some CAD tools are able to deal this type of circuit using a so called false path mechanism, which explicitly indicate to the tool to ignore such a path. Tools that are known to handle such a concept include most static timing analysis engines. However, not all tools contain such a feature. For example, cyclic circuits are often problematic for test generation tools, and require manual intervention. Alternatively, one is left with creating an acyclic

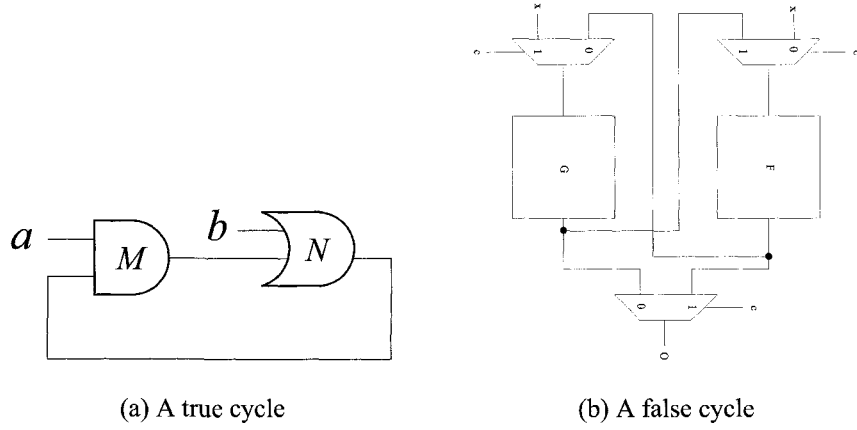


Figure 3.5: Examples of true and false cycles

version as the only way to pass such circuits through these tools. Our proposed research supports the latter, by enabling creation of equivalent acyclic circuits with same inputs and outputs as the original circuit but without the troublesome cycles.

### 3.5 Our Circuit Model

We use a simple gate-level circuit model: a circuit  $C$  is a tuple  $C = \langle G, I, W \rangle$  where  $G$  is a finite set of gates,  $I$  is a finite set of primary inputs, and  $W \subseteq (G \cup I) \times G$  is the set of wires. Each gate computes the logical NAND of its inputs; we assume more complex gates have been dismantled into NANDs.

Note that primary inputs have no incoming edges. We consider every gate to be an output.

We treat nodes as taking one of three values: 0, 1, and  $\perp$ . The first two values are self-explanatory; we write  $\perp$  instead of the X usually used in three-valued logic simulation to emphasize the connection with lattices and partial orders.

The three wire values are partially ordered with a relation  $\sqsubseteq$  that satisfies  $\perp \sqsubseteq 0$  and  $\perp \sqsubseteq 1$  and is transitive ( $x \sqsubseteq y$  and  $y \sqsubseteq z$  implies  $x \sqsubseteq z$ ), reflexive ( $x \sqsubseteq x$ ), and anti-symmetric ( $x \sqsubseteq y$  and  $y \sqsubseteq x$  implies  $x = y$ ).

The relation  $\sqsubseteq$  can be thought of as an information ordering:  $\perp$  is less-defined than 0 or 1, but neither  $0 \sqsubseteq 1$  nor  $1 \sqsubseteq 0$  since both represent the same amount of information, i.e., a defined value. The pointwise extension of this relation to vectors reinforces this intuition:  $(x_1, \dots, x_n) \sqsubseteq (y_1, \dots, y_n)$  iff  $x_1 \sqsubseteq y_1, \dots$ , and  $x_n \sqsubseteq y_n$ . More informally, if  $X \sqsubseteq Y$ , then each element of  $Y$  is either the same as its counterpart in  $X$  or a  $\perp$  has become a 0 or 1. Any 0s or 1s in  $X$  must also be in  $Y$ .

**Definition 5.** *A controlling value for a gate  $G$  is the non- $\perp$  value that applied to any input of  $G$  uniquely sets  $G$ 's output to a non- $\perp$  value independent of assignment to other inputs.*

It follows from this definition that for a gate's output to be set to non- $\perp$ , either all inputs must be set to non-controlling values or at least one input must be set to a controlling value. For a NAND gate, 0 is a controlling value and 1 is non-controlling.

**Definition 6.** *A strongly connected component (SCC) of a circuit  $C$  is a maximal subset of gates  $V \subseteq G$  such that there is a path of wires from any gate in  $V$  to any other gate in  $V$ . Inputs of an SCC are inputs of gates that are part of the SCC that are not driven by gates inside the SCC.*

### 3.6 Combinational Circuits

Like Malik [35], we say a circuit  $C = \langle G, I, W \rangle$  is combinational for an input  $x$  if  $f(C, x)(g) \neq \perp$  for all  $g \in G$  (i.e., three-valued simulation does not lead to any  $\perp$ -valued gates). Again, because of Shiple [54], this is equivalent to insisting that the circuit always stabilizes and never holds state for any delay assignment. Literature on cyclic circuits also refers to this behavior as “well-behaved” and “constructive” [53].

Since we consider all gate outputs to be primary outputs, our definition of combinational insists that every part of the circuit stabilizes. This is actually a conservative definition of combinational behavior: if the environment does



```

function SIMULATE( $\langle G, I, W \rangle, x, s$ )

 $v_0(n) = \begin{cases} x(n) & \text{if } n \in I, \\ \perp & \text{if } n \in G. \end{cases}$ 

 $i \leftarrow 0$ 

while for some  $g$ ,  $\text{EVAL}(W, v, g) \neq v$ 

     $i \leftarrow i + 1$ 

     $v_i = \text{EVAL}(W, v_{i-1}, s_i)$ 

return  $v_i$ 

function EVAL( $W, v, g$ )

 $o = \begin{cases} 0 & \text{if } v(d) = 1 \text{ for all } d \text{ s.t. } (d, g) \in W, \\ 1 & \text{if } v(d) = 0 \text{ for some } d \text{ s.t. } (d, g) \in W, \\ \perp & \text{otherwise.} \end{cases}$ 

    Let  $v'(g) = o$  and  $v'(n) = v(n)$  otherwise.

return  $v'$ 

```

Figure 3.6: The three-valued simulation algorithm, which takes a circuit  $\langle G, I, W \rangle$ , an input function  $x$ , and an infinite schedule of gates  $s$ . It evaluates gates until it reaches a fixed point using EVAL, which updates a single (NAND) gate.

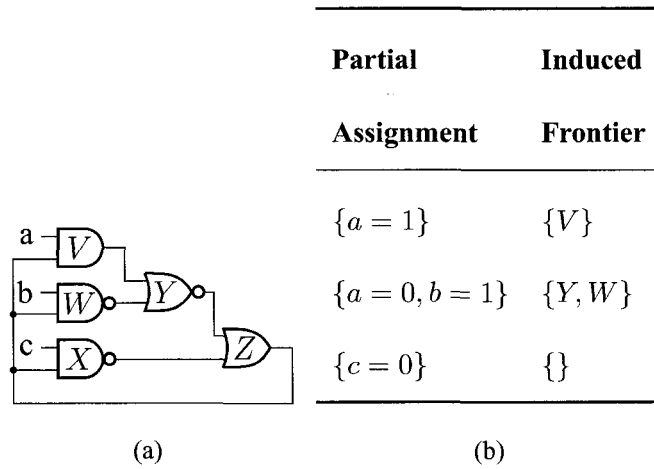


Figure 3.7: (a) A cyclic circuit. (b) Partial assignments and their induced frontiers—the boundary between defined and X-valued gates after applying inputs.

not observe the output of, say, an oscillator, should its presence really matter? Arguments can be made on both sides, but we stipulate that a designer who wants a combinational circuit does not want any state-holding or oscillatory behavior period.

Our goal is to produce an acyclic circuit whose behavior matches that of a cyclic circuit for inputs that are combinational. We assume that non-combinational behavior, if any, was unintended and treat inputs that induce it as don't-care patterns.

Figure 3.7 shows a circuit consisting of a single SCC whose inputs are  $a$ ,  $b$ , and  $c$ . When analyzing a circuit, we first decompose it into SCCs using a standard algorithm [18]. If the input circuit contains more than one SCC, we consider each SCC separately in a topological order.

### 3.7 Finding a Combinational Cover for a Cyclic Circuit

We now present our original algorithm for efficiently extracting a cover for all combinational behavior of a cyclic circuit.

#### 3.7.1 Theoretical Background

**Definition 7.** *Let the set  $\{x_1, \dots, x_n\}$  represent the inputs into an SCC. We define a partial assignment (PA) as a set*

$$PA = \{x_i = v_k : x_i \in (x_1, \dots, x_n) \wedge v_k \in \{0, 1\}\}$$

In this work, we are only concerned with partial assignments to inputs of SCCs. A PA is always associated with some SCC. A valid PA for the circuit in Figure 3.7 is an assignment to one or more of the inputs  $\{a, b, c\}$ , such as  $\{a = 0\}$ ,  $\{b = 0, c = 1\}$ , and  $\{b = 1, c = 1\}$ .

We shall rely on the following two theorems that are key to the cor-

rectness and efficiency of our algorithm. These were first presented in Edwards [22].

**Theorem 1** (Edwards [22]). *For a circuit with a strongly connected component (SCC) to behave combinationally, at least one input to a gate in the SCC must be driven to a controlling value.*

For example, controlling assignments to SCC inputs for the circuit in Figure 3.7 are  $a = 0$ ,  $b = 0$ , and  $c = 0$ . Theorem 1 tells us that at least one of these is required for combinational behavior.

**Theorem 2** (Edwards [22]). *If a partial assignment  $p$  is combinational, then any further assignments that do not contradict any in  $p$  can also be computed combinationally by the circuit fragment implied by  $p$ .*

Consider the PA  $\{c = 0\}$  applied to Figure 3.7. This breaks the connectivity of the SCC, making the circuit behave combinationally. This theorem indicates that additional assignments beyond  $\{c = 0\}$  cannot reverse the combinational behavior already implied by this PA. This permits us to avoid further consideration of acyclic PAs once we have identified them. This supports one of our objectives for the algorithm: generation of *minimal* PAs that capture all combinational behavior. We explain the notion of minimal PAs in Section 3.7.3.

The main difficulty with SCCs is lack of order in which they can be analyzed as SCC gates cannot be sorted topologically. To get around this, we first introduce a novel device which greatly simplifies SCC analysis.

**Definition 8.** *The cyclic controllability frontier of a PA, or frontier for short, is the set of SCC gates that have at least one non- $\perp$  input but whose output is  $\perp$ .*

The frontier captures the notion of a boundary between gates whose output is defined and those whose output is not. A frontier is always associated with a PA. When calculating the frontier for a PA, we use ternary simulation to propagate partial assignments from SCC inputs as far as possible then check for cyclic behavior. Figure 3.7b lists some frontiers induced by partial assignments for the circuit in Figure 3.7a.

**Theorem 3.** *A PA makes an SCC combinational if and only if its frontier is empty.*

*Proof.* If part: If the frontier is empty, then either no gates have any inputs assigned or none have an output of  $\perp$ . From Theorem 1, we know that at least one gate must be driven by a controlling value for combinational behavior. If none have an output of  $\perp$ , then the circuit under that PA is combinational by definition.

Only if part: This follows directly from definition of combinational behavior. □

This theorem tells us that non-empty frontiers only exist in presence of SCCs. For example, the PA  $\{c = 0\}$  in Figure 3.7 yields an empty frontier. Stated differently, we broke the loop without having to assign specific values to the inputs  $\{a, b\}$ .

### 3.7.2 Searching for combinational behavior

We use Theorem 1 to seed our search space with a pool of PAs, each corresponding to a controlling assignment to an SCC input. Any combinational behavior is guaranteed to be present in supersets of one or more of these PAs. Our algorithm proceeds by recording the frontier associated with each PA and uses them to look for opportunities to merge PAs in an attempt to find empty frontiers.

Figure 3.8 shows our technique for identifying all combinational behavior. The algorithm takes a circuit with any number of SCCs and produces a set of PAs under which the circuit is combinational. These PAs control SCC inputs.

The algorithm attacks one SCC at a time (line 4), finding a minimal set

```

1:  $A = \emptyset$                                 ▷ Set of acyclic PAs, the eventual result
2:  $K = \emptyset$                             ▷ All known cyclic PAs, used for merging
3: Clear  $F$                                 ▷ A map from frontier gate  $\rightarrow$  set of PAs
4: while circuit has SCCs
5:   Find next SCC
6:    $P =$  controlling values for SCC inputs    ▷ Initial PAs
7:   while  $P \neq \emptyset$ 
8:      $G = \emptyset$                         ▷ Frontier gates for this iteration
9:     foreach  $p \in P$                       ▷ Consider each candidate PA
10:       simulate  $p$ 
11:       if circuit is combinational under  $p$  then
12:         add  $p$  to  $A$ 
13:       else
14:         add  $p$  to  $K$                         ▷ Remember the PA for merging
15:         foreach gate  $g$  in the frontier induced by  $p$ 
16:           add  $g$  to  $G$                     ▷ Record the frontier gate
17:           add  $p$  to  $F(g)$                 ▷ Remember  $p$  induced  $g$ 
18:          $P = \emptyset$                     ▷ Compute new candidate PAs
19:         foreach frontier gate  $g \in G$ 
20:           if  $|F(g)| > 1$  then            ▷ Need  $\geq 2$  PAs to merge
21:             add each PA from  $\text{mergeAtGate}(K, g)$  to  $P$ 
22: return  $A$ 

```

Figure 3.8: Our algorithm for finding a minimal set of PAs for a circuit (SCC)

that together cover all its combinational behavior.

of covering partial assignments for each. For each SCC, it begins by considering PAs that place a single controlling value on each SCC input (line 6), then enters into a loop (lines 7–21) which alternates between testing whether any of the currently-considered PAs (set  $P$ ) induce combinational behavior (lines 10–17) and attempting to merge already-observed PAs (set  $K$ ) to generate a new set of PAs (lines 18–21). Its goal in this second phase is to break logjams by combining PAs to set the outputs of the latest set of frontier gates it has discovered. The map  $F$  records PAs that affect frontier gates: if  $g$  is a gate, then  $F(g)$  is the set of all PAs that put at least one non-controlling value at an input of  $g$ .

The algorithm in Figure 3.8 is guaranteed to find all combinational behavior within the subject circuit. Starting from individual controlling inputs into SCCs, our frontiers allow us to identify all opportunities where PAs can merge to extend controllability over more gates in an SCC. As we merge these PAs and continue the searching, other acyclic PAs are explored. We continue this cycle of search and merge terminating when we fail to generate new PAs.

### 3.7.3 Merging partial assignments

Here, we describe a key operation used in our main algorithm (Figure 3.8): the generation of new PAs to break the logjam at a frontier gate.



Given a set of PAs and a gate, the algorithm in Figure 3.10 generates a set of PAs that apply non-controlling values to *every* input of the gate, thus setting its output. This algorithm is the key improvement over the technique we presented earlier [44].

We store PAs in a simulated state that captures all assigned nodes and their values. The main algorithm (Figure 3.8) only tries to merge PAs for a gate when at least two PAs set an input on the gate. Merging attempts to produce new PAs by propagating known values across these frontier gates to extend the set of gates whose output is not  $\perp$ .

Consider the example in Figure 3.9. (Figure 3.9b) shows a 3-input (frontier) gate  $g$  for PAs  $p_0, \dots, p_4$ . As always, these PAs control inputs (here  $a, \dots, f$ ) to the SCC that contains  $g$ , not usually the gate's inputs. Note that a gate can only be a frontier for a PA if that PA puts a non-controlling value on one or more of the gate's inputs. We wish to consider merging these PAs to extend the frontier beyond  $g$ . A desirable merge of PAs at a gate  $g$  must be

1. a *gate cover*: merged PAs must define every input of  $g$ .
2. *consistent*: merged PAs must not contain conflicting assignments to inputs. In Figure 3.9, PAs  $p_1$  and  $p_4$  cannot be combined due to a conflicting assignment for  $b$ .

3. *complete*: PAs must be merged such that all permissible combinations are considered. The example in Figure 3.9 provides some degrees of freedom to cover every input that must all be considered. This ensures that our final PAs encapsulate both necessary and sufficient conditions for combinational behavior.
4. *minimal*: merged PAs must not contain any PA that can be removed while satisfying the previous conditions. For example, the merge candidate  $p_0 \cup p_3 \cup p_4$  is rejected since  $p_0$  dominates  $p_4$  (i.e.,  $p_0$  controls both first and third gate inputs;  $p_4$  only controls the third). This condition keeps the final output PAs as concise as possible by not including redundant conditions. Such redundancy when present has two drawbacks: it burdens subsequent stages of the algorithm as it increases memory usage and it also makes testing of merge conditions against other candidate PAs more tedious.

The gate cover, consistency, and completeness conditions are necessary for correctness (without the first two, the analysis does not make sense; the third one guarantees we do not miss any necessary PAs), but minimality is merely desirable—it improves both the running time of our algorithm and the quality of the final result. Our algorithm satisfies the first three conditions and

approximates the minimality by computing an irredundant sum-of-products, as we describe below.

We can merge PAs by merely verifying that there are no conflicts to the assigned primary inputs of the SCC. In other words, we do not need to check for conflicts of every internal node. This greatly speeds our consistency check procedure.

The argument for this is a proof by contradiction. Let two partial assignments  $A$  and  $B$  have non-conflicting controlling assignments to SCC inputs, and assume some intermediate node  $I$  has conflicting values under assignments  $A$  and  $B$  (i.e., one is 0, the other 1; there is no conflict if either is  $\perp$ ). The gate that produces  $I$  must either have one input set to a controlling value or all inputs set to non-controlling values. We can repeat the analysis on those input(s) until we find conflicting inputs at SCC inputs, which contradicts the original assumption.

Merging PAs is an instance of binate covering problem (BCP) because we must cover all gate inputs and because conflicts between PAs prevent certain combinations, making it binate. However, our need for a complete enumeration is not typical of BCPs.

Figure 3.10 shows our algorithm for merging PAs. We construct a BDD

Name	Assignments
------	-------------

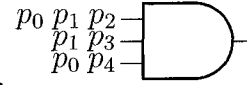
$p_0 \quad \{a = 1\}$

$p_1 \quad \{b = 0, c = 1\}$

$p_2 \quad \{c = 1, d = 1\}$

$p_3 \quad \{e = 0\}$

$p_4 \quad \{b = 1, f = 1\}$



(a)

(b)

$$(p_0 + p_1 + p_2)(p_1 + p_3)(p_0 + p_4)(\overline{p_1} + \overline{p_4})$$

(c)

$$(p_0 p_1 \overline{p_4}) + (p_0 \overline{p_1} p_3) + (\overline{p_1} p_2 p_3 p_4)$$

(d)

$$p_0 \cup p_1 = \{a = 1, b = 0, c = 1\}$$

$$p_0 \cup p_3 = \{a = 1, e = 0\}$$

$$p_2 \cup p_3 \cup p_4 = \{b = 1, c = 1, d = 1, e = 0, f = 1\}$$

(e)

Figure 3.9: Illustration of merging PAs at a gate.

comprising our covering problem at the gates and the conflicts therein as a product of sums (POS). The covering at each gate input is encoded as a sum term comprising all PAs that can control that input. By definition, these are all non-controlling input assignments, as otherwise the PA would have continued past this gate. To set the gate's output to a deterministic value, it is necessary that we select PAs covering all the gate's inputs, hence the sum of products. However, we must account for the PAs containing conflicting and therefore non-compatible assignments to the inputs into the SCC. We thus augment our POS expression with clauses which capture the conflicts as pair-wise sums of negation of PAs that conflict.

We then use the Minato-Morreale algorithm [38] to generate an irredundant sum of products in ZDD [39] form. We use these to continue propagation. Note that the addition of conflicts causes the irredundant sum of products to contain negated terms, which we discard.

Figure 3.9 illustrates this working on an example. The five PAs in Figure 3.9a control the inputs of the three-input AND gate in Figure 3.9b. Our merging algorithm (Figure 3.10) starts by expressing the constraint at the AND gate as a product of sums (Figure 3.9c): each input must be controlled by at least one PA (the first three terms) and conflicting PAs (i.e., those that insist on contradictory assignments to inputs: here  $p_1$  sets  $b = 0$  and  $p_4$  sets  $b = 1$ , so

both  $p_1$  and  $p_4$  are illegal together) are prohibited. Next, these constraints are transformed to an irredundant sum-of-products (Figure 3.9d). Finally, negations are removed from each term in the ISOP, leading to a new set of PAs Figure 3.9e. By construction, each of these PAs controls all three gate inputs and has no conflicting input assignments.

### 3.7.4 Another Example

We will use the cyclic circuit in Figure 3.11 to illustrate use of frontiers for extraction of PAs as well as how negated literals arise and how we deal with them.

We start by applying a controlling value to each input separately. Figure 3.12 summarizes the results. Note that when  $a$  is 0, the circuit is combinational since the feedback path is broken, so we include the assignment  $\{a = 0\}$  as part of our minimal cover and will not consider any further assignments that contain  $\{a = 0\}$  (Theorem 2).

Consider setting  $b = 0$ . Although this is a controlling value for gate  $R$  (its output becomes 0 regardless of  $Q$ ), by itself this is not enough to force the whole circuit to behave combinational because a 0 on  $R$  is a non-controlling value on the OR gate  $V$ . Each of the assignments  $c = 0$  and  $d = 0$  also have

```

1: function MERGEATGATE( $K, g$ )
2:    $R = \emptyset$  ▷ Generated set of PAs
3:    $POS = 1$  ▷ Product of Sums
4:   foreach input  $i$  of gate  $g$ 
5:      $p_i =$  PAs in  $K$  that set  $i$  and induce  $g$  as a frontier
6:     if  $p_i = \emptyset$  then return  $\emptyset$  ▷ Cannot control some input
7:      $P = \bigvee_{\forall i} p_i$ 
8:      $POS = POS \wedge P$ 
9:   foreach Conflicting PA pairs  $\{p_i, p_j\} \in K$ 
10:     $POS = POS \wedge (\overline{p_i} \vee \overline{p_j})$ 
11:    $zddISOP = ISOP(POS)$ 
12:   remove negated literals and duplicates from  $zddISOP$ 
13:   add products to  $R$ 
14:   return  $R$ 

```

Figure 3.10: Our PA merging algorithm: return a set of PAs that apply non-controlling values to every input of a gate.

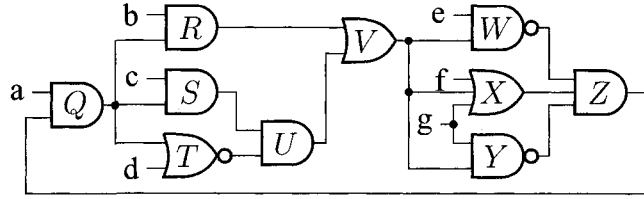


Figure 3.11: Small cyclic circuit for illustrating partial assignment extraction

a frontier of  $V$ . Similar analysis shows different assignments to  $e$ ,  $f$ , and  $g$  all yielding  $Z$  as their frontier.

The SCC input  $g$  has both 0 and 1 present as controlling assignments since it is connected to a NAND and an OR. Constructing a PA that includes such conflicting assignments is meaningless. Our algorithm tracks and caches conflicting partial assignments to guard against composing a PA from such conflicting assignments. As we stated previously, positive and negated literals in our initial POS indicate presence or absence of PAs respectively; we do not use negated literals to imply inverting the individual assignments within a given PA.

Next, we analyze the frontiers we have obtained from logic simulation. Only two gates,  $V$  and  $Z$ , appear in any frontier; we will attempt to set the outputs of these gates by judiciously combining sets of PAs that might completely



define values at inputs of these gates. At every frontier gate, we compose a covering problem in the form of a product of sums (POS), where each sum represents candidate PAs that define a given input of that gate. We add to this POS pair-wise conflicts between PAs that cannot be merged.

At gate  $V$ , the top input can only be defined by assignment  $p_1$ , so this becomes the first sum term in our POS: Figure 3.13a. The lower input can be defined by either of  $p_2$  or  $p_3$ , so we add  $(p_2 + p_3)$  as our second sum term. We note that none of these assignments conflict, so there is no need to add any additional assignments. As a matter of computation runtime though, we have found that adding conflicts does not materially affect the subsequent AllSat computation. The alternative which is to compute and add only relevant conflicts at every frontier gate input was found empirically to be more expensive. We store conflicting assignments in a cache which we update as we create new assignments. These are added to all POS expressions. This is not shown in Figure 3.13 for brevity, where we only show relevant conflicts. Similar analysis at gate  $Z$  yields the POS shown in Figure 3.13d.

Our algorithm now computes all satisfying assignments to each of the POS expressions at frontier gates. We remove negated literals as well as identical products from within each sum. The output of this computation is shown next to each POS in Fig. 3.13a and 3.13b. This computation yields three new

PAs. Each leads to an empty frontier and (therefore) an acyclic circuit. Our algorithm terminates and returns the partial assignments listed in Figure 3.13c.

### **3.8 Experimental Results**

### **3.9 Benefits of Proposed Research**

As already described in Section 3.3, cyclic circuits show up often in domain of high level language synthesis. As such, one immediate application of this research is in synthesis post-processing steps that eliminate cycles. A second application of our research is analysis of cyclic circuits produced by cyclic synthesis engines such as CYCLIFY [49]. Most circuit analysis engines are incapable of handling the outputs of such circuits directly. Our algorithm when combined with other published algorithms can produce small equivalent acyclic circuits that are guaranteed to reproduce the combinational behavior of the original cyclic circuits.

### **3.10 Conclusions**

We presented a new algorithm for identifying all the combinational behavior of a cyclic circuit. The algorithm is useful for evaluating cyclic specifi-

Table 3.1: Comparison with Edwards [22]

Example	Gates		Edwards [22]		Our Approach		Acyclic
	Total SCC		PAs	Time	PAs	Time	
arbiter5	213	25	257	1.3s	25	0.1s	14
arbiter6	248	30	745	8	29	0.1	16
arbiter7	283	35	2205	69	33	0.2	18
arbiter8	318	40	6581	656	37	0.3	20
expo	124	69	54517	2868	23260	2.0	338
exlo	150	47	43777	2341	232	1.0	10
garyo	177	32	-	> 1h	290	0.6	11
planeto	253	51	-	> 1h	1489	0.3	22
s1488o	272	61	-	> 1h	588	0.2	89
table3o	311	49	-	> 1h	3604	1.0	38

cations that often arise from high-level synthesis [6, 7]. One application of our algorithm is transforming cyclic combinational circuits to an acyclic equivalent; it replaces the first half of the procedure described by Edwards [22].

The chief contribution of our work is a speed improvement of several orders of magnitude over Edwards [22] due to much more clever pruning of the search space and use of implicit method for merging PAs. It is therefore able to deal with practical-sized cyclic circuits.

Our algorithm analyzes all possible inputs into SCCs without considering whether such patterns can in fact occur in the original circuit (i.e., whether they are controllability don't-cares). This saves us from performing an image computation on the surrounding circuit, making the analysis much faster. However, it is possible that considering the don't-care set would reduce the number of PAs we consider and further speed the search. We have yet to explore the trade-off between computing don't-cares and reducing the number of PAs.

Independent of these further refinements, we have presented a practical algorithm that is able to quickly characterize all the combinational behavior of a realistic-sized cyclic circuit. Our intended application is the construction of an acyclic equivalent of a cyclic circuit to make it palatable to existing synthesis tools, but we believe our algorithm has other important applications in analysis

and formal equivalence verification of cyclic circuits.

### 3.11 Summary

Compiling high-level hardware languages can produce circuits containing combinational cycles that can never be sensitized. Such circuits do have well-defined functional behavior, but wreak havoc with most logic synthesis and timing tools, which assume acyclic combinational logic. As such, some sort of cycle-removal step is usually necessary for handling these circuits.

Cyclic circuits have also been shown to the most compact representation for certain classes of circuits. This property was exploited recently by the synthesis engine of Riedel and Bruck [49], which won the best paper award at DAC-2003. It remains to be seen whether cyclic circuits will ever get used in ASICs or microprocessors due to complexity of enhancing all CAD tools to support them. While the area saving are attractive, the need for a non-standard static timing methodology might make such circuits outside reach of most designers and automated design flows. At the same time, synthesizing circuits into cyclic forms may reveal interesting properties about these circuits that can be exploited for analysis and optimization back in the acyclic domain.

Our research advanced addresses an important requirement for both ar-

as above by providing a bridge from cyclic to acyclic circuits.

Label	Assignment	Frontier	At Frontier	Acyclic
$p_0$	$\{a = 0\}$	$\{\}$		$\checkmark$
$p_1$	$\{b = 0\}$	$\{V\}$	$R = 0$	
$p_2$	$\{c = 0\}$	$\{V\}$	$U = 0$	
$p_3$	$\{d = 1\}$	$\{V\}$	$U = 0$	
$p_4$	$\{e = 0\}$	$\{Z\}$	$W = 1$	
$p_5$	$\{f = 1\}$	$\{Z\}$	$X = 1$	
$p_6$	$\{g = 0\}$	$\{Z\}$	$Y = 1$	
$p_7$	$\{g = 1\}$	$\{Z\}$	$X = 1$	

Figure 3.12: PAs from applying controlling values to each input in isolation.

All frontiers are either gate  $V$  or gate  $Z$

$$(p_1)(p_2 + p_3) \implies (p_1p_2) + (p_1p_3)$$

(a)

$$(p_4)(p_5 + p_7)(p_6)(\overline{p_6} + \overline{p_7}) \implies (p_4p_5p_6)$$

(b)

Product Term	Assignment
$p_0$	$\{a = 0\}$
$p_1p_2$	$\{b = 0, c = 0\}$
$p_1p_3$	$\{b = 0, d = 1\}$
$p_4p_5p_6$	$\{e = 0, f = 1, g = 0\}$

(c)

Figure 3.13: Partial assignment extraction on a small cyclic circuit (a) POS and final ISOP for frontier gate  $V$ . (b) POS and ISOP for  $Z$ . (c) A minimal set of partial assignments that reproduce all combinational behavior.



## **Chapter 4**

### **Conclusions**

#### **4.1 Statistical Optimization of Digital Circuits**

We introduced a new concept of a worst negative statistical slack path and derived a procedure for tracing and optimizing such paths. In the process, we also derived a new approximation for the max operation on random variables for use in circuit optimization. Our approach allows us to steer the optimization process towards different mean-variance goals. The significance of this work is that it can be used during design cycle to increase tolerance for the effects of manufacturing variations by trading off circuit delay and area requirements for reduced timing variance with user controlled weights. We demonstrated fidelity of our approach on ISCAS benchmarks with consistent variance reduction in exchange for moderate increases in area and low increases in mean delays.

A large number of publications advocating statistical approaches continues to appear at every major CAD and VLSI conference. It is difficult to predict where this research will end up. By and far, the most significant gap in this research is in availability of bottom up transistor variations models in practice and how these variations manifest at the gate, circuit, and system level. Genuine fabrication data about transistor variations is very highly guarded by the companies and foundries. While it is mathematically convenient to assume that variations are gaussian in nature and proceed with analysis and optimization using this assumption, we have no idea how closely this matches reality. In perusing literature on statistical approaches, the author has yet to find a bottom up driven models describing variations corresponding to a true deep submicron technology node such as 65nm or 45nm. Notwithstanding the intellectual property concerns, it is imperative that we strive to continue further research around more realistic data and build up bottom up models with more basis in reality on how electronic designs vary in response to transistor level variations. Publications arising out of this work are [46].

## 4.2 An Efficient Algorithm for Analysis of Cyclic Circuits

We presented a new algorithm for identifying all the combinational behavior of a cyclic circuit. The algorithm is useful for evaluating cyclic specifications that often arise from high-level synthesis [6, 7]. One application of our algorithm is transforming cyclic combinational circuits to an acyclic equivalent; it replaces the first half of the procedure described by Edwards [22].

The chief contribution of our work is a speed improvement of several orders of magnitude over Edwards [22] due to much more clever pruning of the search space. It is therefore able to deal with practical-sized cyclic circuits.

Our algorithm analyzes all possible inputs into SCCs without considering whether such patterns can in fact occur in the original circuit (i.e., whether they are controllability don't-cares). This saves us from performing an image computation on the surrounding circuit, making the analysis much faster. However, it is possible that considering the don't-care set would reduce the number of PAs we consider and further speed the search. We have yet to explore the trade-off between computing don't-cares and reducing the number of PAs.

Although our algorithm performs quite well, it can be improved further. The current performance bottleneck arises when merging PAs at a frontier gate to produce more PAs to consider. Most of our PAs are generated here and most

are later discarded. A more clever approach, perhaps Espresso-based, might reduce both the number of new PAs generated and the time it takes to derive them.

Independent of these further refinements, we have presented a practical algorithm that is able to quickly characterize all the combinational behavior of a realistic-sized cyclic circuit. Our intended application is the construction of an acyclic equivalent of a cyclic circuit to make it palatable to existing synthesis tools, but we believe our algorithm has other important applications in analysis and formal equivalence verification of cyclic circuits. Publications arising out of this work are [44] and [45].

## Bibliography

- [1] ACM/IEEE. *ACM/IEEE TAU Workshop*, 2004.
- [2] Aseem Agarwal, David Blaauw, Vladimir Zolotov, and Sarma Vrudhula. Statistical timing analysis using bounds and selective enumeration. In *TAU '02: Proceedings of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems*, pages 16–21, New York, NY, USA, 2002. ACM Press.
- [3] Vineet Agarwal, Navneeth Kankani, Ravishankar Rao, Sarvesh Bhardwaj, and Janet Wang. An efficient combinationality check technique for the synthesis of cyclic combinational circuits. In *Proc. of ASP-DAC*, 2005.
- [4] Xiaoliang Bai, Chandu Visweswariah, and Philip N. Strenski. Uncertainty-aware circuit optimization. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 58–63, New York, NY, USA, 2002. ACM Press.

- [5] Michel R. C. M. Berkelaar and Jochen A. G. Jess. Gate sizing in MOS digital circuits with linear programming. In *EURO-DAC '90: Proceedings of the conference on European design automation*, pages 217–221, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.
- [6] G. Berry. The constructive semantics of pure Esterel. Draft book, 1999.
- [7] G. Berry. *The foundations of Esterel*. MIT Press, 2000.
- [8] Gérard Berry. Esterel on hardware. *Philosophical Transactions of the Royal Society of London. Series A*, 339:87–103, April 1992. Issue 1652, Mechanized Reasoning and Hardware Design.
- [9] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical timing analysis: From basic principles to state of the art. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(4):589–607, April 2008.
- [10] F. Bourdoncle. Efficient chaotic iteration strategies with widenings. *Lecture Notes in Computer Science*, 735:128–141, 1993.
- [11] R. B. Brawhear, N. Menezes, C. Oh, L. Pillage, , and R. Mercer. Predicting circuit performance using circuit-level statistical timing analysis. In *European Design and Test Conference, 1994*, pages 332–337, 1994.

- [12] Hongliang Chang and Sachin S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single Pert-like traversal. In *2003 International Conference on Computer-Aided Design (ICCAD'03), November 9-13, 2003, San Jose, CA, USA*, pages 621–626, 2003.
- [13] Chung-Ping Chen, Chris C. N. Chu, and D. F. Wong. Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 617–624, New York, NY, USA, 1998. ACM Press.
- [14] De-Sheng Chen and Majid Sarrafzadeh. An exact algorithm for low power library-specific gate re-sizing. In *DAC '96: Proceedings of the 33rd annual conference on Design automation*, pages 783–788, New York, NY, USA, 1996. ACM Press.
- [15] C. E. Clark. The greatest of a finite set of random variables. In *Operations Research*, volume 9, pages 145–162, 1961.
- [16] J. Cong. Challenges and opportunities for design innovations in nanometer technologies, 1997.

- [17] A. R. Conn, N. I. M. Gould, and Ph L. Toint. *Lancelot: A FORTRAN Package for Large-Scale Nonlinear Optimization (Release A)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.
- [18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.
- [19] Olivier Coudert. Gate sizing for constrained delay/power/area optimization. *IEEE Trans. Very Large Scale Integr. Syst.*, 5(4):465–472, 1997.
- [20] Srinivas Devadas, Horng-Fei Jyu, Kurt Keutzer, and Sharad Malik. Statistical timing analysis of combinational circuits. In *ICCD '92: Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, pages 38–43, Washington, DC, USA, 1992. IEEE Computer Society.
- [21] S. W. Director and W. Maly, editors. *Statistical Approach to VLSI*. Elsevier Science B.V., 1994.
- [22] S. Edwards. Making cyclic circuits acyclic. In *Proc. Design Automation Conference*, pages 159–162, 2003.
- [23] J. P. Fishburn. LATTIS: an iterative speedup heuristic for mapped logic. In *DAC '92: Proceedings of the 29th ACM/IEEE conference on Design*



*automation*, pages 488–491, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

- [24] A. Gupta and Charley Selvidge. Acyclic modeling of combinational loops. In *Proc. International Conference on Computer-Aided Design*, 2005.
- [25] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.
- [26] N. Halbwachs and F. Maraninchi. On the symbolic analysis of combinational loops in circuits and synchronous programs. In *Proc. Euromicro*, pages 345–348, 1995.
- [27] Masanori Hashimoto and Hidetoshi Onodera. A performance optimization method by gate sizing using statistical static timing analysis. In *ISPD '00: Proceedings of the 2000 international symposium on Physical design*, pages 111–116, New York, NY, USA, 2000. ACM Press.
- [28] IEEE Computer Society / ACM. *2003 International Conference on Computer-Aided Design (ICCAD'03), November 9-13, 2003, San Jose, CA, USA*, 2003.

- [29] E. Jacobs and M. Berkelaar. Gate sizing using a statistical delay model. In *Proc. Design and Test in Europe*, pages 283–291, 2000.
- [30] Horng-Fei Jyu and Sharad Malik. Statistical timing optimization of combinational logic circuits. In *Proceedings of the 1993 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, Washington, DC, USA, 1993. IEEE Computer Society.
- [31] W. Kautz. The necessity of closed circuit loops in minimal combinational circuits. *IEEE Trans. Comput.*, C-19:162–164, February 1970.
- [32] David Lammers. Designers wary as IBM embraces statistical timing. *EE Times Online*, 2004. <http://www.eetimes.com/story/OEG20040209S0006>.
- [33] David Lammers. IBM uses EinsStat statistical analysis timing tool. *EE Times Online*, 2004. <http://www.eedesign.com/news/showArticle.jhtml?articleId=17601636>.
- [34] Jing-Jia Liou, Kwang-Ting Cheng, Sandip Kundu, and Angela Krstic. Fast statistical timing analysis by probabilistic event propagation. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 661–666, New York, NY, USA, 2001. ACM Press.

- [35] S. Malik. Analysis of cyclic combinational circuits. *IEEE Trans. Computer-Aided Design*, 13(7):950–956, July 1994.
- [36] F. Maraninchi. The Argos language: graphical representation of automata and description of reactive systems. In *Proc. International Conference on Visual Languages*, Kobe, Japan, 1991.
- [37] Noel Menezes, Ross Baldick, and Lawrence T. Pileggi. A sequential quadratic programming approach to concurrent gate and wire sizing. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 144–151, Washington, DC, USA, 1995. IEEE Computer Society.
- [38] Shin-ichi Minato. Fast generation of irredundant sum-of-products forms from binary decision diagrams. In *Proceedings the Synthesis and Simulation Meeting and International Exchange (SASIMI)*, pages 64–73, Kobe, Japan, April 1992.
- [39] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. pages 272–277, 1993.

- [40] Rajeev Murgai. *Logic Synthesis and Verification*, chapter Technology-based transformations, pages 141–165. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
- [41] Arthur Nadas. Probabilistic PERT. In *IBM journal of research and development*, pages 339–347, 1978.
- [42] Kedar S. Namjoshi and Robert P. Kurshan. Efficient analysis of cyclic definitions. In *Computer Aided Verification*, volume 1633 of *LNCS*, pages 394–405, Trento, Italy, July 1999.
- [43] Sani Nassif. Delay variability: sources, impact, and trends. In *Proceedings of ISSCC*, 2000.
- [44] Osama Neiroukh, Stephen A. Edwards, and Xioyu Song. An efficient algorithm for the analysis of cyclic circuits. In *Proceedings of the Symposium on VLSI (ISVLSI)*, pages 303–308, Karlsruhe, Germany, March 2006.
- [45] Osama Neiroukh, Stephen A. Edwards, and Xioyu Song. Transforming cyclic circuits into equivalent acyclic circuits. Submitted to *IEEE Transactions on Computer Aided Design*, in review.

- [46] Osama Neiroukh and Xiaoyu Song. Improving the process-variation tolerance of digital circuits using gate sizing and statistical techniques. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 294–299, Washington, DC, USA, 2005. IEEE Computer Society.
- [47] Michael Orshansky and Kurt Keutzer. A general probabilistic framework for worst case timing analysis. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 556–561, New York, NY, USA, 2002. ACM Press.
- [48] Stephen E. Rich, Matthew J. Parker, and Jim Schwartz. Reducing the frequency gap between asic and custom designs: a custom perspective. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 432–437, New York, NY, USA, 2001. ACM Press.
- [49] M. Riedel and J. Bruck. The synthesis of cyclic combinational circuits. In *Proc. Design Automation Conference*, pages 163–168, 2003.
- [50] Ronald L. Rivest. The necessity of feedback in minimal monotone combinational circuits. *IEEE Trans. Comp.*, 26(6):606–607, 1977.

- [51] T. Shiple, G. Berry, and H. Touati. Constructive analysis of cyclic circuits. In *Proc. European Design and Test Conf.*, pages 328–333, 1996.
- [52] T. R. Shiple, V. Singhal, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Analysis of combinational cycles in sequential circuits. In *Proc. IEEE International Symposium Circuits and Systems (ISCAS)*, pages 592–595, 1996.
- [53] Thomas R. Shiple, Gérard Berry, and Hervé Touati. Constructive analysis of cyclic circuits. In *Proceedings of the European Design and Test Conference*, pages 328–333, Paris, France, March 1996.
- [54] Thomas Robert Shiple. *Formal Analysis of Synchronous Circuits*. PhD thesis, October 1996. Memorandum UCB/ERL M96/76.
- [55] R. A. Short. A theory of relations between sequential and combinational realizations of switching functions. Technical report, Stanford Electronics Laboratories, 1960.
- [56] L. Stok. False loops through resource sharing. In *Proc. International Conference on Computer-Aided Design*, pages 345–348, 1992.
- [57] Synopsys Corporation. *Design Compiler*.

- [58] Hiran Tennakoon and Carl Sechen. Gate sizing using lagrangian relaxation combined with a fast gradient-based pre-processing step. In *IC-CAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 395–402, New York, NY, USA, 2002. ACM.
- [59] Eric W. Weisstein. *CRC Concise Encyclopedia of Mathematics*. CRC Press, 1999.