12-1-2013

# Max Operation in Statistical Static Timing Analysis on the Non-Gaussian Variation Sources for VLSI Circuits

Abu M. Baker
*University of Nevada, Las Vegas*

# MAX OPERATION IN STATISTICAL STATIC TIMING ANALYSIS ON THE NON-GAUSSIAN VARIATION SOURCES FOR VLSI CIRCUITS

by

Abu M Baker

Bachelor of Science in Electrical and Electronic Engineering
Bangladesh University of Engineering and Technology
1997

Master of Science in Electrical Engineering
Oklahoma State University
2001

Master of Science in Electrical Engineering
University of South Alabama
2004

A thesis submitted in partial fulfillment
of the requirements for the

Doctor of Philosophy in Electrical Engineering

Department of Electrical and Computer Engineering
College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2013

THE GRADUATE COLLEGE

We recommend the dissertation prepared under our supervision by

**Abu Baker**

entitled

**Max Operation in Statistical Static Timing Analysis on the Non-Gaussian Variation Sources for VLSI Circuits**

is approved in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy in Engineering - Electrical Engineering**
Department of Electrical and Computer Engineering

Yingtao Jiang, Ph.D., Committee Chair

Biswajit Das, Ph.D., Committee Member

Mei Yang, Ph.D., Committee Member

Hui Zhao, Ph.D., Graduate College Representative

Kathryn Hausbeck Korgan, Ph.D., Interim Dean of the Graduate College

**December 2013**

## ABSTRACT

MAX operation in Statistical Static Timing Analysis on the non-Gaussian Variation

Sources for VLSI Circuits

by

Abu M Baker

Dr. Yingtao Jiang, Examination Committee Chair
Professor of Electrical and Computer Engineering
University of Nevada, Las Vegas

As CMOS technology continues to scale down, process variation introduces significant uncertainty in power and performance to VLSI circuits and significantly affects their reliability. If this uncertainty is not properly handled, it may become the bottleneck of CMOS technology improvement. As a result, deterministic analysis is no longer conservative and may result in either overestimation or underestimation of the circuit delay. As we know that Static-Timing Analysis (STA) is a deterministic way of computing the delay imposed by the circuits design and layout. It is based on a predetermined set of possible events of process variations, also called corners of the circuit. Although it is an excellent tool, current trends in process scaling have imposed significant difficulties to STA. Therefore, there is a need for another tool, which can resolve the aforementioned problems, and Statistical Static Timing Analysis (SSTA) has become the frontier research topic in recent years in combating such variation effects.

There are two types of SSTA methods, path-based SSTA and block-based SSTA. The goal of SSTA is to parameterize timing characteristics of the timing graph as a function of the underlying sources of process parameters that are modeled as random variables. By performing SSTA, designers can obtain the timing distribution (yield) and its sensitivity to various process parameters. Such information is of tremendous value for both timing sign-off and design optimization for robustness and high profit margins. The block-based SSTA is the most efficient SSTA method in recent years. In block-based SSTA, there are two major atomic operations max and add. The add operation is simple; however, the max operation is much more complex.

There are two main challenges in SSTA. The Topological Correlation that emerges from reconvergent paths, these are the ones that originate from a common node and then converge again at another node (reconvergent node). Such correlation complicates the maximum operation. The second challenge is the Spatial Correlation. It arises due to device proximity on the die and gives rise to the problems of modeling delay and arrival time.

This dissertation presents statistical Nonlinear and Nonnormals canonical form of timing delay model considering process variation. This dissertation is focusing on four aspects: (1) Statistical timing modeling and analysis; (2) High level circuit synthesis with system level statistical static timing analysis; (3) Architectural implementations of the atomic operations (max and add); and (4) Design methodology.

To perform statistical timing modeling and analysis, we first present an efficient and accurate statistical static timing analysis (SSTA) flow for non-linear cell delay model with non-Gaussian variation sources.

To achieve system level SSTA we apply statistical timing analysis to high-level synthesis flow, and develop yield driven synthesis framework so that the impact of process variations is taken into account during high-level synthesis.

To accomplish architectural implementation, we present the vector thread architecture for max operator to minimize delay and variation. Finally, we present comparison analysis with ISCAS benchmark circuits suites.

In the last part of this dissertation, a SSTA design methodology is presented.

ACKNOWLEDGEMENT

First of all, I would like to express my deepest thanks and appreciation to my academic advisor Professor Yingtao Jiang. During the past few years, Professor Yingtao Jiang provided me with the opportunities to work on the most cutting-edge and important research topics in VLSI Electronic Design Automation area. From him, I have learned how to formulate problems with both scientific and technological significance and how to solve them in an efficient and systematic way. He also gave me countless advice in academic writing and my future career path. Without his guidance and generous support, I could not have proceeded so far in my PhD study. I sincerely appreciate his patient help, precious advice, and consistent encouragement.

I also would like to thank my PhD committee members, Professor Biswajit Das, Dr. Mei Yang, and Dr. Hui Zhao for their precious time in reviewing my thesis.

Finally, I owe thanks to my parents who have always stood by me and guided me through my pursuit of PhD degree. My life is more meaningful because of their love and care. Words cannot express the gratitude I owe them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

As integrated circuits have continued to scale down further, the manufacturing process has become less predictable. After manufacturing, the process parameters and the dimensions of the fabricated devices and wires can be very different from their designed values. For example, an oxide thickness that is nominally 25 Å may turn out to be, after manufacturing, thicker than the designed value at 27 Å, or thinner at 24 Å. Such variations in the process parameters can induce substantial fluctuations in the performance of VLSI circuits. Performance parameters such as timing and power may be affected either positively or negatively, and the net result of this may be a low manufacturing yield, as a majority of the manufactured dies fail to meet the performance specifications. Therefore, manufacturing process induced variation, or process variation, is an important consideration in VLSI circuit design and yield analysis.

## *1.1 The Impact of Rising Process Variations*

Most semiconductor product improvements over the past decades are direct or indirect consequences of the perpetual shrinking of devices and circuits, allowing performance enhancements at lower fabrication cost. A paralleling trend is that process variations and intra-die variability increase with each technology node. Since most high-performance analog circuits depend on matched devices and differential signal paths, this trend has begun to diminish yields and reliabilities of chip designs. Fundamentally, the problem is that parameters of devices on the same die show increasing intra-die variations, thereby exhibiting different characteristics. For example, Table 1 displays the evolution of the typical transistor threshold voltage standard deviation $\sigma\{V_{Th}\}$ normalized by the threshold voltage ($V_{Th}$) for several technologies, as reported in [1]. Also notice that $V_{Th}$ exhibits further dependence on gate length variations through the drain-induced-barrier-lowering (DIBL) effect under large drain-source voltage bias conditions, as demonstrated by the characterization in [2] using a 65nm technology. Since DIBL worsens as the channel is scaled down, this additional impact on threshold voltage variations can be assumed to be even stronger beyond the 65 nm technology node.

TABLE 1: INTRA-DIE VARIABILITY VS CMOS TECHNOLOGY

| **Technology node** | **250 nm (%)** | **180 nm (%)** | **130 nm (%)** | **90 nm (%)** | **65 nm (%)** | **45 nm (%)** |
|---|---|---|---|---|---|---|
| $\sigma\{V_{Th}\}/V_{Th}$ | 4.7 | 5.8 | 8.2 | 9.3 | 10.7 | 16 |

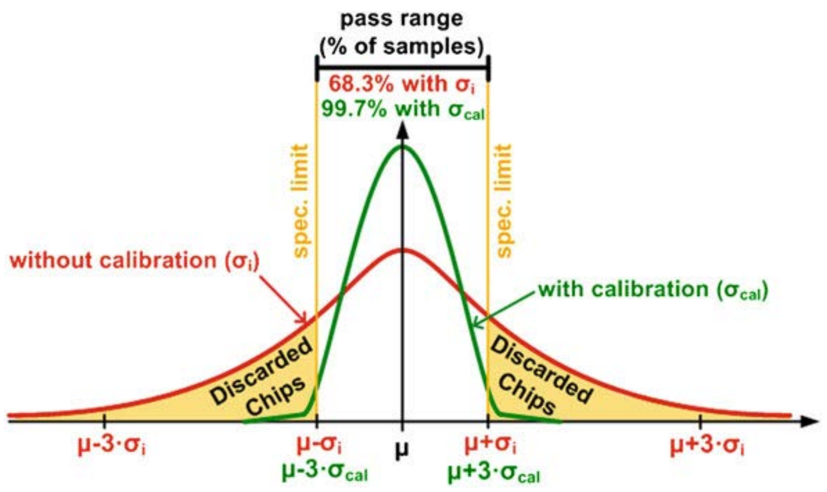


Figure 1: Specification variation impact on the fraction of discarded chips.

A direct consequence of device parameter variations is a decrease in production yields because block-level and system-level parameters will show a corresponding increase in variations. This relationship between variations and yield can be inferred from the visualization in Figure 1, where the Gaussian distribution of a specification with a standard deviation σ around the mean value μ is shown together with the specification limits (±3σ in this example). For standalone analog circuits, parameters such as gain may have an upper and/or lower specification limit, and the samples that exceed the limit(s) during production testing must be discarded. Guardbands are often defined to account for measurement uncertainties by following procedures such as repeating the same test or performing other more comprehensive tests to determine whether the part can be sold to customers, which incurs additional test cost in a manufacturing environment.

An important observation from Figure 1 is that an increase of variation (σ) widens the Gaussian distribution, which leads to a higher percentage of parts that fall within the highlighted ranges that require them to be scrapped or retested. Clearly, there is a direct relationship between the amount of process variations and production cost due to low yields. In the case of wireless mixed-signal integrated systems, the trend towards increasing integration and complexity has also been paralleled by technical challenges and rising cost of testing, which can amount up to 40–50% of the total manufacturing cost [3, 4].

## *1.2 Paradigm Shift from Deterministic STA to Statistical STA*

Static-timing analysis (STA) has been one of the most ubiquitous and popular analysis engines in the design of digital circuits for the last 20 years. However, in recent years, the increased loss of predictability in semiconductor devices has raised concern over the ability of STA to effectively model statistical variations. This has resulted in all-encompassing research in the so-called SSTA, which marks a significant departure from the traditional STA framework.

Traditional STA tools are deterministic and compute the circuit delay for a specific process condition. Hence, all parameters that impact the delay of a circuit, such as device gate length and oxide thickness, as well as operating voltage and temperature, are assumed to be fixed and are uniformly applied to all the devices in the design. In STA, process variation is modeled by running the analysis multiple times, each at a different process condition. For each process condition a so-called corner file is created that specifies the delay of the gates at that process condition. By analyzing a sufficient number of process conditions the delay of the circuit under process variation can be bounded.

The fundamental paleness of STA is that while global shifts in the process (referred to as die-to-die variations) can be approximated by creating multiple corner files, there is no statistically rigorous method for modeling variations across a die (referred to as within-die variations). However, with process scaling progressing well into the nanometer regime, process variations have become significantly more pronounced and within-die variations have become a non-negligible component of the total variation. It is shown that the incapability of STA to model within-die variation can result in either an over or underestimate of the circuit delay, depending on the circuit topology. Hence, STA’s desirable property of being conservative may no longer hold for certain circuit topologies Rather, at the same time, STA may be overly pessimistic for other circuit topologies. The accuracy of STA in advanced processes is therefore a serious concern. Therefore, the need for an effective modeling of process variations in timing analysis has led to extensive research in statistical STA.

## *1.3 Circuit Performance Analysis under Process Variations*

Since process variations can significantly affect circuit performance parameters such as timing and power, it is important to analyze the relation between these in order to predict their impact on circuit performance, for parametric yield prediction as well as variation-aware circuit design and optimization. We will now overview several classes of analysis techniques.

### *1.3.1 Multi-Corner-Based Methodology*

In general, the value of a process parameter after manufacturing falls into a bounded range from a minimum to a maximum value. A process corner corresponds to a set of values of process variables in the parameter space where each parameter in the space takes either the minimum or maximum value. A worst-case corner is defined as the corner where the process parameters take their extreme values that can result in the worst behavior for a typical circuit. Traditional circuit analysis deals with process variations by predicting the worst-case circuit behavior evaluated at worst- case corners. Unfortunately, with the number and magnitude of process variables increasing, checking a small set of worst-case corner could be risky if it may not cover the region sufficiently, or excessively conservative, if the corners are chosen to embody a pessimistic worst-case [5, 6]. Therefore, a multi-corner-based method, which predicts the circuit behavior by analyzing the circuit at all enumerative corners, has to be used to evaluate worst-case behavior. However, the multi-process corner based methodology also suffers from the following disadvantages.

First, the method is too computationally intensive: on the one hand, as the number of varying process parameters increases, the number of process corners to enumerate, which grows exponential with the number of process variables, grows too high; on the other hand, under intra-die variation, the process parameter values of devices [wires] in the same chip can vary differently, and therefore, the number of process corners required must also consider region-based analysis (alluded to in Section 1.1), which worsens the exponential behavior.

Second, the approach is too conservative and pessimistic in that the process corner corresponding to the worst-case performance may have a very low probability of occurrence, which results in an over-pessimistic results. As an example, suppose there are two independent sources of variations $p_1$ and $p_2$ with Gaussian distribution $N(\mu_1, \sigma_1)$ and $N(\mu_2, \sigma_2)$, respectively. Then, using the corner-based method, the worst-case could be found by inspecting the corners are at $(p_1, p_2) = (\mu_1 \pm 3\sigma_1, \mu_2 \pm 3\sigma_2)$. However, the probability of each of the $(p_1, p_2)$ corners is as low as $1.96 \times 10^{-5}$, significantly less than at the $3\sigma$ point. This pessimism is liable to become especially severe as the number of varying process parameters grows higher. Amending this procedure so that the corners correspond to $3\sigma$ points does not help either: fundamentally, the problem here is that the level sets of the Gaussian are ellipsoids, and worst-casing over the corners of a multidimensional box is doomed to failure.

### *1.3.2 Monte Carlo Simulation Approach*

The effects of process variations on circuit performance can also be predicted by Monte Carlo simulation method [7, 8, 9]. The approach is an iterative process where each iteration consists of two basic steps, sampling and simulation. In each sampling step, a set of sampled values of process parameters are generated according to the distribution of process parameter variations, or samples as delay/power for all circuit nodes generated according to their distributions. The simulation step then simply runs a circuit/timing/power simulation, using the generated sample values. The Monte Carlo method is very accurate in predicting the distribution of circuit performance. However, for an integrated circuit, the number of iterations required for convergence is generally greater than 10,000. Although smart techniques can be used to reduce the sampling size, it is still a large number so as to achieve desirable accuracy of simulation result. Therefore the approach is highly computationally expensive, and is not practical even for medium size circuits.

## *1.4 Statistical Analysis Method*

Statistical performance analysis methods provide a good possibility for analyzing circuit performance with good accuracy and efficient run-time. These approaches directly exploit the statistical information of the process parameters and utilize efficient stochastic techniques [8] to determine the probability distribution of the circuit performance. In these methods, instead of using fixed values of process parameters (as is done in each multi-corner analysis), random variables are used to model the uncertainty of process parameters. In timing analysis, the delays of gates and interconnects and arrival times at intermediate nodes are all random variables. Therefore, unlike conventional deterministic STA which computes timing based on deterministic values, the SSTA method stochastically computes delays and arrival times on a set of random variables. Therefore, probabilistic characteristics, such as the probability density function (PDF) of circuit timing, can be obtained and yield of timing can also be predicted from the computation. Similarly, for statistical leakage power analysis, the leakage power of each gate is modeled as a random variable and the result of computation is probability distribution and yield of full-chip leakage.

It is worth mentioning that under process variations, circuit optimization techniques should be also adapted to be capable of considering the effects the process variations. Therefore, the importance of analyzing circuit performance under process variation is not limited to yield prediction, but also for variation-aware circuit design and optimization. Multiple-process corner based methods are too pessimistic, and may result in over-constrained circuit optimization. Therefore, although more computational effort goes into reoptimizing the circuit to meet the worst-case performance requirement over all the corners, this does not significantly contribute to improving the yield of circuit performance. The alternative of using accurate Monte Carlo methods suffers from a different drawback: the expensive run-time prohibits these methods from being used within a circuit optimization algorithm. In contrast to these, statistical methods for circuit performance analysis are computationally efficient and can achieve good accuracies, and therefore, have the potential to be practically be integrated into various steps of the design flows, such as technology mapping, synthesis, and physical design.

## *1.5 Our Contributions*

In modern chip design, circuit performance is greatly constrained by timing. In nanometer-scale technologies, leakage power which can be derived from timing has become a major component of total chip power dissipation, and it is highly sensitive to manufacturing variations due to its exponential dependency on some process parameters. Therefore, in this thesis, we will focus on the analysis of timing, and propose efficient statistical performance analysis methods for timing under the effect of inter-die and intra-die variations. As intra-die variations exhibit spatial correlation, i.e., devices [wires] spatially located close to each other tend to experience more similar variations than those placed far away, the effect of spatial correlations are also considered in the analysis using a model proposed in Chapter 2. The major contributions of the thesis are:

***Statistical timing analysis with non-Gaussian distributed process parameters and nonlinear delay functions***

Statistical timing analysis methods that assume process variations to take the form of linear functions of Gaussians can be very run-time efficient. However, as delay shows nonlinear sensitivities to some process parameters, and some process variations, which show non-Gaussian distributions and cannot be well approximated with Gaussians, it is essential to develop an SSTA technique that can handle non-Gaussian process parameters and nonlinear delay functions to achieve desirable accuracy. For this purpose, we first present a novel block based SSTA modeling in this thesis that is designed to consider both global correlations and path correlations. We develop a model encompassed with numerical computations and tightness probabilities to conditionally approximate the MAX/MIN operator by a linear mixing operator. We extend the commonly used canonical timing model to be able to represent all possible correlations, including the path correlations, between timing variables in the circuit. We show that developing SSTA technique that is capable of incorporating non-Gaussian sources of process variations and/or nonlinear delay functions is important to correctly predict the circuit timing. This work was published in [10].

***High Level Circuit Synthesis with System Level Statistical Static Timing Analysis under Process Variation***

Process variations are of great concern in deep sub-micron technology. Early prediction of their effects on the circuit performance and parametric yield is extremely useful. Due to the increase of the design complexity in today's SoC chips, a demand for high level design has increased. Therefore, we propose the timing analysis model so that the impact of process variations is taken into account during high level synthesis. High-level synthesis (HLS) is a synthesis technique that allows designers to move up the design chain to a higher level of abstraction. This means that instead of designing at the register transfer level (RTL), where a designer must specify all the timing of the circuit, the designer can work at a behavioral level, where only the data flow of the required circuit has to be specified. This frees the designer from the burden of many low-level details of circuit design, allowing for productivity increases of up to 10 times and code reductions of up to 100 times [11]. As manufacturing technologies continue to shrink, HLS is becoming a powerful technique to decrease the amount of time required to design a chip. In this dissertation, we apply statistical timing analysis to high-level synthesis, and develop yield driven synthesis framework so that the impact of process variations is taken into account during high-level synthesis.

***Architectural Level Statistical Static Timing Analysis under Process Variation***

SSTA is a very complex solution and computationally intense. It is also proved that the run time complexity of the SSTA algorithm is $O(p.n.(N_g + N_I))$, which is $p.n$ times that of deterministic STA, where $n$ is total number of grids into which the chip is divided and $p$ is the number of spatially correlated parameters considered, $N_g$ is the total number of gates and $N_I$ the number of net connections in the circuit. Also, the run-time of the SSTA algorithm can be divided into four folds. They are as following: a. The times required to find the delay distribution of the gate and interconnect, b. The time required to evaluate the max function, c. The time required to compute output transition time at each gate output, and d. The time required to evaluate the *sum* function.

There is a new architecture called vector-thread (VT) which is an architectural paradigm describes a class of architectures that unify the vector and multithreaded execution models. VT architectures compactly encode large amounts of structured parallelism in a form that lets simple microarchitectures attain high performance at low power by avoiding complex control and datapath structures and by reducing activity on long wires. We present a run time complexity analysis here to show which factors most greatly affect the CPU time of the algorithm.

***Design Methodology***

We present SSTA flow by using the fast statistical timing analysis flow [58] for transistor level macros that can compute the delay distributions due to process variations of all paths in the macros. It first groups the macro transistors into logic gates called xcells by applying special grouping technique [65]. The method used by block-based SSTA engine. It is based on simultaneous application of the usual static as well as statistical static timing analysis. At the first stage usual STA is applied and at the second stage - SSTA. The offered method of the analysis allows reaching acceptable analysis results from the practical point of view of accuracy at rather small expenses of machine runtime. SSTA engine determines delay distributions for all paths in the macro using the variation libraries. The timing yield step estimates the required arrival time based on the most critical path due to variation.

# CHAPTER 2

# LITERATURE OVERVIEW

In this chapter, we first study the key sources of variation in timing prediction, which make timing analysis a challenging task for nanoscale digital circuits.

## *2.1 Static Timing Analysis*

Static Timing Analysis is to verify the timing behavior of a circuit in the deterministic case (without variations). An alternative approach for timing verification is using simulation. It requires applying a large set of data patterns to the input pins. Therefore, timing simulation is more time-consuming compared to STA. STA is a tool, which is widely used to determine the delay of integrated digital circuits. In order to have a properly operating circuit, not only the design needs to be well done, but also its operating points must be determined. For an arbitrary digital circuit, its worst-case delay determines the maximum speed (frequency) at which the circuit will operate as expected. Therefore, Static-Timing Analysis provides a key measurement for the circuit performance. But the limitations of traditional static timing analysis techniques lie in their deterministic nature. An alternative approach that overcomes these problems is SSTA, which treats delays not as fixed numbers, but as probability density functions, taking the statistical distribution of parametric variations into consideration while analyzing the circuit.

## *2.2 Sources of Timing Variation*

We first discuss different types of uncertainties that arise as a design moves from specification to implementation and final operation in the field. We then focus on process variations in more detail and discuss the distinction between die-to-die and within-die variations and the source of so-called spatial correlations. Finally, we will discuss the impact of different types of process variations on the timing of a circuit.

## *2.3 Process, Environmental, and Model Uncertainties*

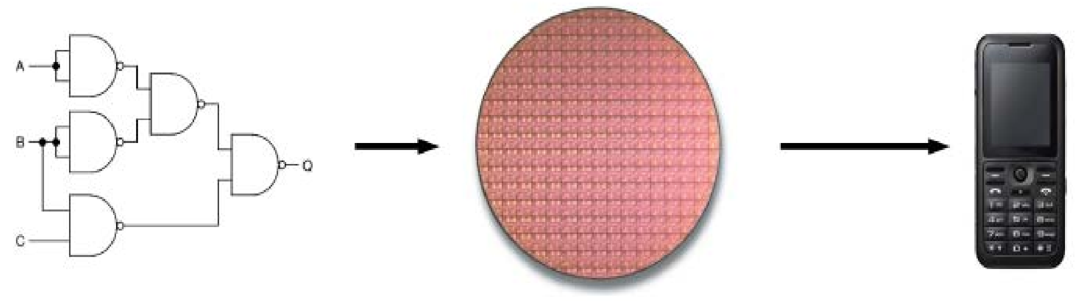
The uncertainty in the timing estimate of a design can be classified into three main categories:

a. Modeling and analysis errors: inaccuracy in device models, in extraction and reduction of interconnect parasitics, and in timing-analysis algorithms,

b. Manufacturing variations: uncertainty in the parameters of fabricated devices and interconnects from die to die and within a particular die,

c. Operating context variations: uncertainty in the operating environment of a particular device during its lifetime, such as temperature, supply voltage, mode of operation, and lifetime wear-out.

To illustrate each of these uncertainties, consider the stages of design, from initial specification to final operation, as shown in Figure 2. The design process starts with a broad specification of the design and then goes through several implementation steps, such as logic synthesis, buffer insertion, and place and route. At each step, timing analysis is used to guide the design process. However, timing analysis is subject to a host of inaccuracies, such as undetected false paths, cell-delay error, error in interconnect parasitics, SPICE models, etc. These modeling and analysis errors result in a deviation between the expected performance of the design and its actual performance characteristics. For instance, the STA tool might utilize a conservative delay-noise algorithm resulting in certain paths operating faster than expected.



Model Variations Process Variations Operating Context Variation

Figure 2: Steps of the design process and their resulting timing uncertainties.

In the next stage, the design is fabricated and each individual die incurs additional manufacturing-related variations due to equipment imprecisions and process limitations. Finally, a manufactured die is used in an application such as a cell phone or a laptop. Each particular die then sees different environmental conditions, depending on its usage and location. Since environmental factors such as temperature, supply voltage, and workload affect the performance of a die, they give rise to the third class of uncertainty. To achieve the required timing specification for all used die throughout their entire lifetime, the designer must consider all three sources of uncertainty. However, a key difference between the three classes of uncertainty is that each has a sample space that lies along a different dimension. Hence, each class of uncertainty calls for a different analysis approach.

First, we recall that the sample space of an experiment or a random trial is the set of all possible outcomes. The timing uncertainty caused by modeling and analysis errors has as its sample space the set of design implementations resulting from multiple design attempts. Each design attempt results in an implementation that triggers particular inaccuracies in the models and tools, resulting in a timing distribution across this sample space. However, a design is typically implemented only once and their needs to be a high level of confidence that the constraints will be met in the first attempt. Hence, the designer is interested in the worst-case timing across this sample space. Thus, margins are typically added to the models to create sufficient confidence that they are conservative and will result in a successful implementation. Although a statistical analysis is a model and analysis uncertainty is uncommon, it could aid in a more accurate computation of the delay with a specified confidence level.

In the case of process variations, the sample space is the set of manufactured die. In this case, a small portion of the sample space is allowed to fail the timing requirements since those die can be discarded after manufacturing. This considerably relaxes the timing constraints on the design and allows designers to significantly improve other performance metrics, such as power dissipation. In microprocessor design, it is common to perform so-called binning where die are targeted to different applications based on their performance level. This lessens the requirement that all or a very high percentage of the sample space meets the fastest timing constraint. Instead, each performance level in the sample space represents a different profit margin, and the total profit must be maximized.

The sample space of environmental uncertainty is across the operational life of a part and includes variations in temperature, modes of operation, executed instructions, supply voltage, lifetime wear-out, etc. Similar to model and analysis uncertainty, the chip is expected to function properly throughout its operational lifetime in all specified operating environments. Even if a design fails only under a highly unusual environmental condition, the percentage of parts that will fail at some point during their operational life can still be very high. Therefore, a pessimistic analysis is required to ensure a high confidence of correct operation throughout the entire lifetime of the part.

Naturally, this approach results in a design that operates faster than necessary for much of its operational life, leading to a loss in efficiency. For instance, when a part is operating at a typical ambient temperature the device sizing or supply voltage could be relaxed, reducing power consumption. One approach to address this inefficiency is to use runtime adaptivity of the design [12], [13].

Since each of the three discussed variabilities represents orthogonal sample spaces, it is difficult to perform a combined analysis in a meaningful manner. Environmental uncertainty and uncertainty due to modeling and analysis errors are typically modeled using worst-case margins, whereas uncertainty in process is generally treated statistically. Hence, most SSTA research works, as well as this dissertation, focus only on modeling process variations. However, the accuracy gained by moving from DSTA to SSTA methods must be considered in light of the errors that continue to exist due to the other sources of timing error, such as analysis and modeling error, uncertainty in operating conditions, and lifetime wear-out phenomena. We will discuss in the next section the sources of process variation in more detail.

## *2.4 Sources of Process Variation*

### ***Physical Parameters, Electrical Parameters, and Delay Variation***

The semiconductor manufacturing process has become more complex; at the same time process control precision is struggling to maintain relative accuracy with continued process scaling. As a result, a number of steps throughout the manufacturing process are prone to fluctuations. These include effects due to chemical mechanical polishing (CMP), which is used to planarize insulating oxides and metal lines, optical proximity effects, which are a consequence of patterning features smaller than the wavelength of light [14-16], and lens imperfections in the optical system. These, as well as other numerous effects, cause variation of device and interconnect physical parameters such as gate length (or critical dimension-CD), gate-oxide thickness, channel doping concentration, interconnect thickness and height, etc., as shown in Figure 3.

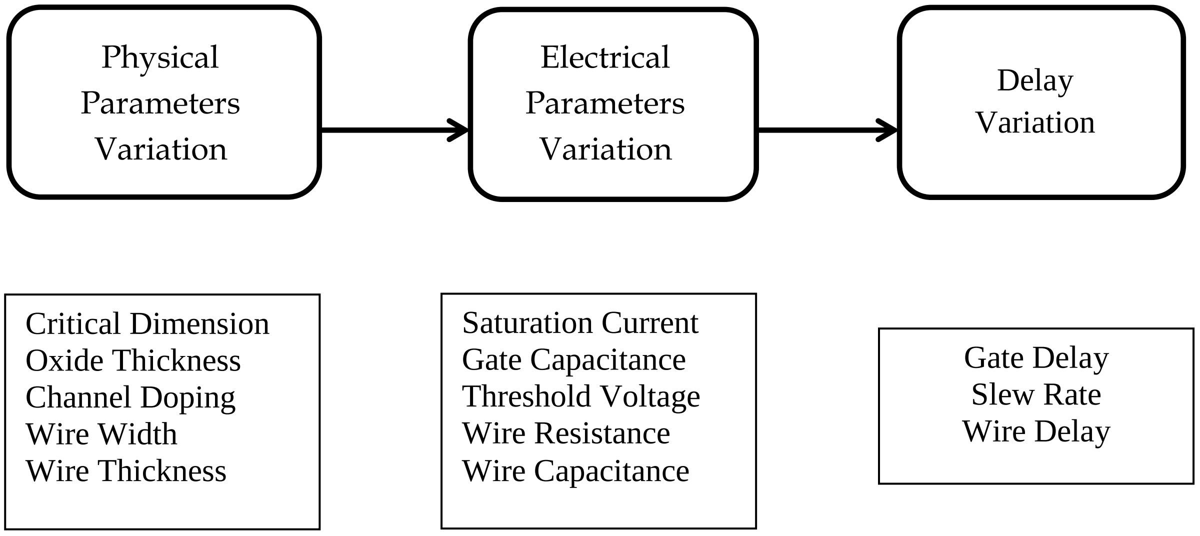


Figure 3: Variation Propagation.

Among these, CD variation and channel doping fluctuations have typically been considered as dominant factors. However, many SSTA methods model a much wider range of physical parameters. Variations in these physical parameters, in turn, result in variations in electrical device characteristics, such as the threshold voltage, the drive strength of transistors, and the resistance and capacitance of interconnects. Finally, the variations in electrical characteristics of circuit components result in delay variations of the circuit.

It is important to note that more than one electrical parameter may have a dependence on a particular physical parameter. For example, both resistance and capacitance of an interconnect are affected by variation in wire width. An increase in interconnect width reduces the separation between wires, resulting in an increased coupling capacitance while decreasing the resistance of the wire. Similarly, perturbations in the gate-oxide thickness influence the drive current, the threshold voltage, and the gate capacitance of the transistors. Dependence of two or more electrical parameters on a common physical parameter gives rise to correlation of these electrical parameters and ignoring this correlation can result in inaccurate results. For instance, if we ignore the negative correlation between capacitance and resistance, there is a nonzero probability that both resistance and capacitance are at their worst-case values. However, this is physically impossible and leads to unrealistic *RC* delay estimates. In [17], the authors present a method to determine the process-parameter values that result in a more realistic worst case delay estimate.

It would be ideal to model each process step in the manufacturing process to determine the variations and correlations in the physical parameters. However, such an analysis is complex and impractical due to the number of equipment-related parameters in each fabrication step and the total number of steps. Hence, most SSTA approaches have taken the physical parameters themselves (such as CD, doping concentration, and oxide thickness) to be the basic random variables (RVs). These variables are either assumed to be independent or to have well-understood correlations.

## *2.5 Classification of Physical-Parameter Variation*

Physical-parameter variations can be classified based on whether they are deterministic or statistical and based on the spatial scale over which they operate, as shown in Figure 4.

Systematic variations are components of physical parameter variation that follow a well-understood behavior and can be predicted upfront by analyzing the designed layout. Systematic variations arise in large part from optimal proximity effects, CMP, and its associated metal fill. These layout-dependent variations can be modeled premanufacturing by performing a detailed analysis of the layout. Therefore, the impact of such variations can be accounted for using deterministic analysis at later stages of the design process [18], [19] and particularly at timing sign-off. However, since we do not have layout information early in the design process, it is common to treat these variations statistically. In addition, the models required for analysis of these systematic variations are often not available to a designer, which makes it advantageous to treat them statistically, particularly when it is unlikely that all effects will assume their worst case values.

Nonsystematic or random variations represent the truly uncertain component of physical parameter variations. They result from processes that are orthogonal to the design implementation. For these parameters, only the statistical characteristics are known at design time, and hence, they must be modeled using RVs throughout the design process. Line-edge roughness (LER) and random dopant fluctuations (RDF) are examples of nonsystematic random sources of variation.

It is common that earlier in the design flow, both systematic and nonsystematic variations are modeled statistically. As we move through the design process and more detailed information is obtained, the systematic components can be modeled deterministically, if sufficient analysis capabilities are in place, thereby reducing the overall variability of the design.

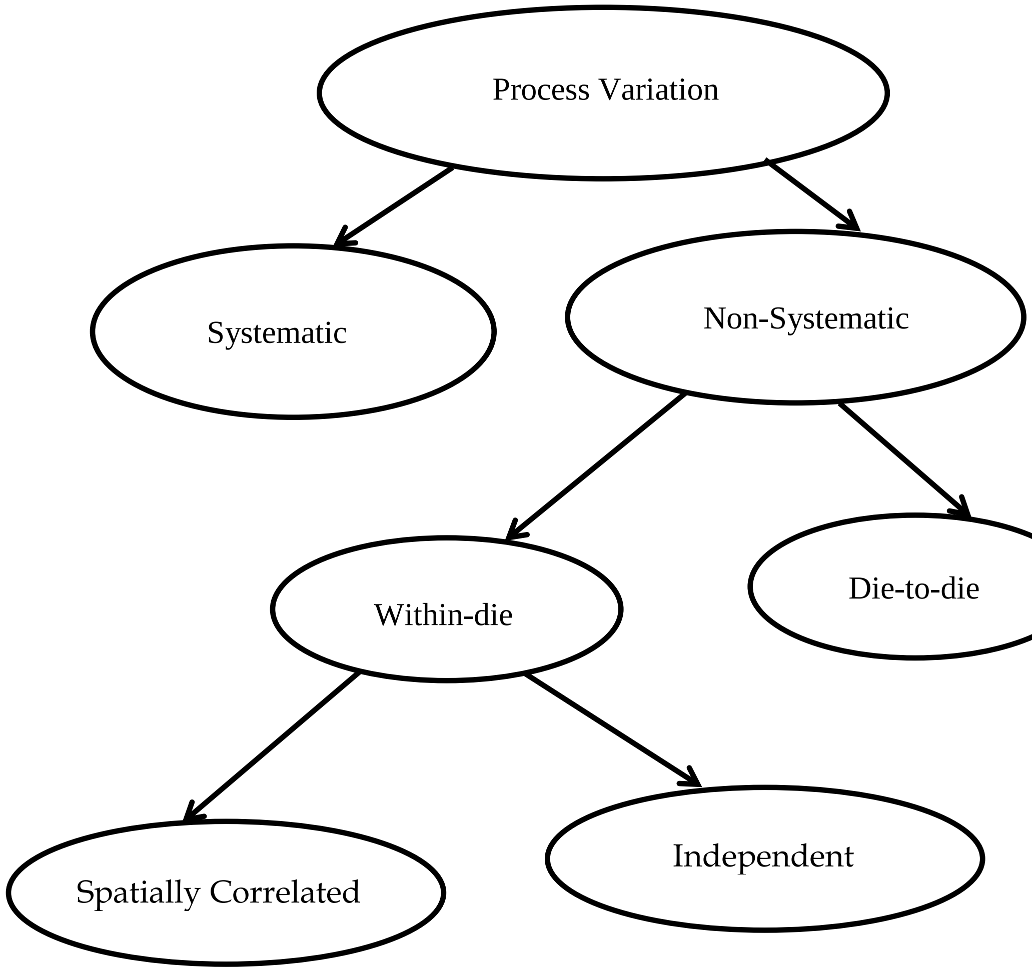


Figure 4: Types of Variation.

Nonsystematic variations can be further analyzed by observing that different sources of variations act on different spatial scales. Some parameters shift when the equipment is loaded with a new wafer or between processing one lot of wafers to the next which this can be due to small unavoidable changes in the alignment of the wafers in the equipment, changes in the calibration of the equipment between wafer lot processing, etc. On the other hand, some shift can occur between the exposures of different reticles on a wafer, resulting in reticle-to-reticle variations. A reticle is the area of a wafer that is simultaneously exposed to the mask pattern by a scanner. The reticle is approximately 20 mm × 30 mm and will typically contain multiple copies of the same chip layout or multiple different chip layouts. At each exposure, the scanner is aligned to the previously completed process steps, giving rise to a variation in the physical parameters from one reticle to the next. Finally, some shift can occur during the reticle exposure itself. For instance, a shift in a parameter, such as laser intensity, may occur while a particular reticle is scanned leading to within reticle variations. Another example is non-uniform etch concentration across the reticle, leading to the variation in the CD.

These different spatial scales of variation give rise to a classification of nonsystematic variations into two categories.

Die-to-die variations (also referred to as global or inter die variations) affect all the devices on the same die in the same way. For instance, they cause the CD of all devices on the same chip to be larger or smaller than nominal. We can see that die-to-die variations are the result of shifts in the process that occur from lot to lot, wafer to wafer, reticle to reticle, and across a reticle if the reticle contains more than one copy of a chip layout.

Within-die variations (also referred to as local or intra die variations) affect each device on the same die differently. In other words, some devices on a die have a smaller

CD, whereas other devices on the same die have a larger CD than nominal. Within-die variations are only caused by across-reticle variations within the confines of a single chip layout as illustrated in Figure 5. Finally, within-die variations can be categorized into spatially correlated and independent variations as discussed as follows.

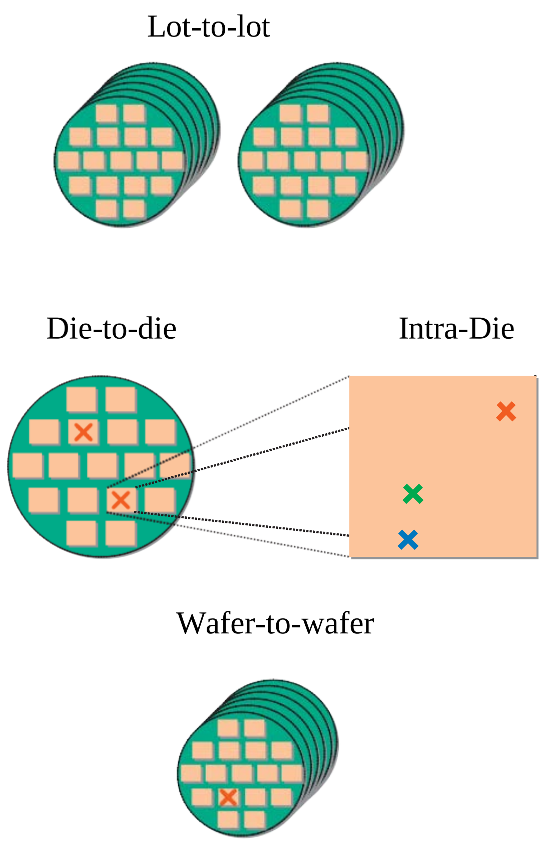


Figure 5: Classification of physical variations.

***Spatially correlated variations***

Many of the underlying processes that give rise to within-die variation change gradually from one location to the next. Hence, these processes tend to affect closely spaced devices in a similar manner, making them more likely to have similar characteristics than those placed far apart. The component of variation that exhibits such spatial dependence is known as spatially correlated variation.

***Independent variations***

The residual variability of a device that is statistically independent from all other devices and does not exhibit spatially dependent correlations is referred to as independent variation. These variations include effects such as RDF and LER. It has been observed that with continued process scaling, the contribution of independent within-die variation is increasing.

## *2.6 Impact of Correlation on Circuit Delay*

Once the parameter variations and their respective distributions are known the challenge of computing the delay of the circuit emerges. For any circuit there are basically two types of delay that need to be computed. One is the total delay of a path consisting of devices connected in series (Single Path Delay). The second one is the maximum delay between two or more parallel paths (Maximum Delay of Multiple Paths).

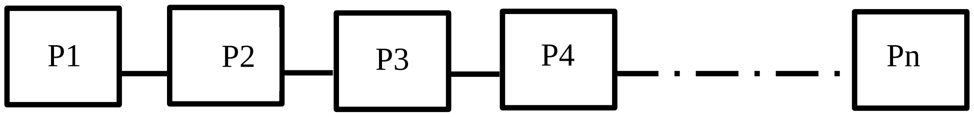


Figure 6: Devices connected in series.

The most straightforward case is Single Path Delay with devices having independent delays. Figure 6 depicts this case, where *Pi* refers to the delay probability of device *i*. Furthermore, let us assume that these probability densities are equal and normal distributed with mean $\mu$ and variance $\sigma^2$. Now the computation of the total delay of such a path becomes easy. The delay must be equal to the sum of all delays on the path.

However, the sum of independent normal distributions results in a normal distribution with mean equal to the sum of the individual means and variance equal to the sum of the individual variances.

$$\sum_{i=1}^{n} N(\mu, \sigma^2) = N(\sum_{i=1}^{n} \mu, \sum_{i=1}^{n} \sigma^2) = N(n\mu, n\sigma^2) \tag{2.1}$$

As a result the total coefficient of variation given by the ratio of deviation to mean becomes smaller than the coefficient of variation of a single device on the path.

$$\left(\frac{\sigma}{\mu}\right)_{path} = \frac{\sqrt{n\sigma^2}}{n\mu} = \frac{1}{\sqrt{n}}\left(\frac{\sigma}{\mu}\right)_{gate} \tag{2.2}$$

Assuming independency definitely eases the computational effort, but in many cases this assumption is simply wrong. Therefore, this time the example path from Figure 6 will have devices with again equal normal distributions (mean μ, variance $\sigma^2$), but this time the delay probabilities will be correlated with correlation coefficient $\rho$. Now the task of computing the total delay becomes a little bit more complicated. The mean of the total delay equals the sum of the means of the individual delays on the path. The variance, however, changes drastically. In addition to the sum of the individual variances on the path, a term describing the correlation between each two individual distributions is added.

$$\mu_{path} = n\mu \tag{2.3}$$

$$\sigma_{path}^2 = \sum_{i=1}^{n} \sigma^2 + 2\rho \sum_{i=1}^{n} \sum_{j>i}^{n} \sigma_i \sigma_j = n\sigma^2\,(1 + \rho(n-1)) \tag{2.4}$$

The above results give rise to the following expression describing the coefficient of variation. It now depends on the correlation coefficient.

$$\left(\frac{\sigma}{\mu}\right)_{path} = \frac{\sqrt{n\sigma^2(1+\rho(n-1))}}{n\mu} = \sqrt{\frac{1+\rho(n-1)}{n}}\left(\frac{\sigma}{\mu}\right)_{gate} \tag{2.5}$$

By close observation of Equ. 2.5 it can be seen that a simple substitution of $\rho = 0$ (uncorrelatedness which implies independency) results in equation 2.2. Furthermore, if

$\rho = 1$(fully correlated delays) then the total coefficient of variation equals the coefficient of variation for a single device which is larger than the total one in the independent case. However, the mean is independent of the coefficient $\rho$ and is the same in all cases. This means that the denominator in the fraction describing the total coefficient of variation is constant. Hence, it is the numerator (the standard deviation) that varies with the correlation. Therefore, for both independent and fully correlated cases the resulting density function is around the same mean, only its spread changes. Given that the spread in the correlated case is larger, it can be concluded that this assumption results in overestimation of the total delay.

Another situation where the worst-case delay needs to be determined is the case where multiple paths converge. Here, probability densities function of the maximum needs to be computed. In the following two paths with equal and normal total delay probability densities are considered.

Figure 7 shows the resulting density function of the maximum delay, given that the two paths are independent. It can be seen that the mean of the maximum is larger than any of the original means and the shape closely, but not perfectly resembles a Gaussian density. The increase in the mean is caused by the fact that in three out of four cases the maximum delay is on the right side of the mean of the single path and only in one case on the left.

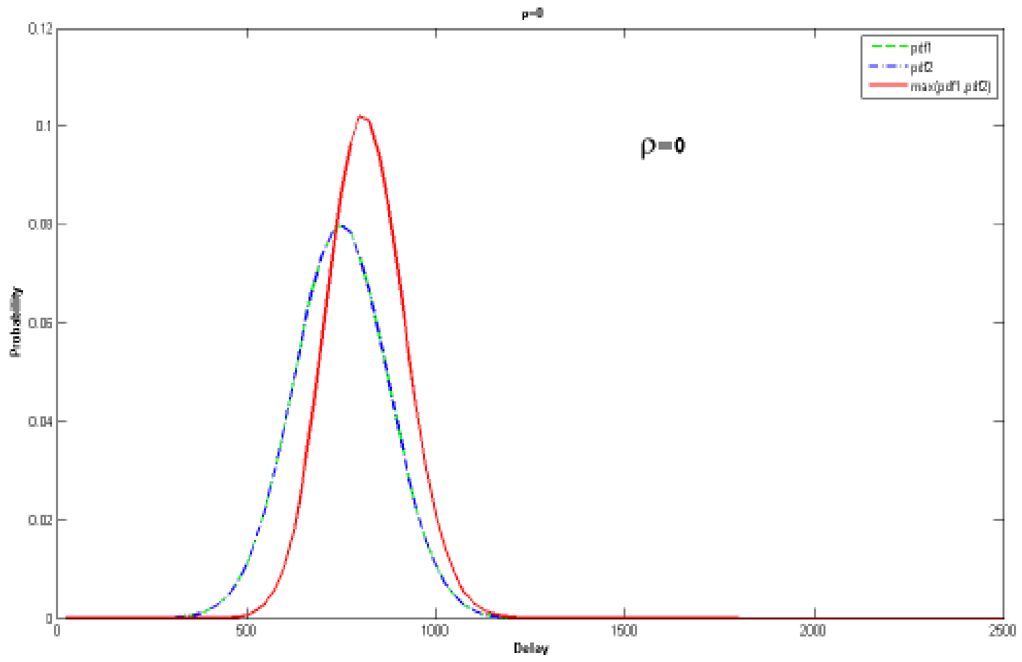


Figure 7: Independent ($\boldsymbol{\rho = 0}$).

As the correlation between the two paths increases, the resulting maximum density shifts to the left and its mean converges more to the mean of the two paths. Figure 8 shows that for ($\rho = 0.5$).

For perfectly correlated delay of the individual paths the result becomes trivial. Having two random variables with equal distributions, which are perfectly correlated, basically means observing one and the same random variable twice at a single instant of time.

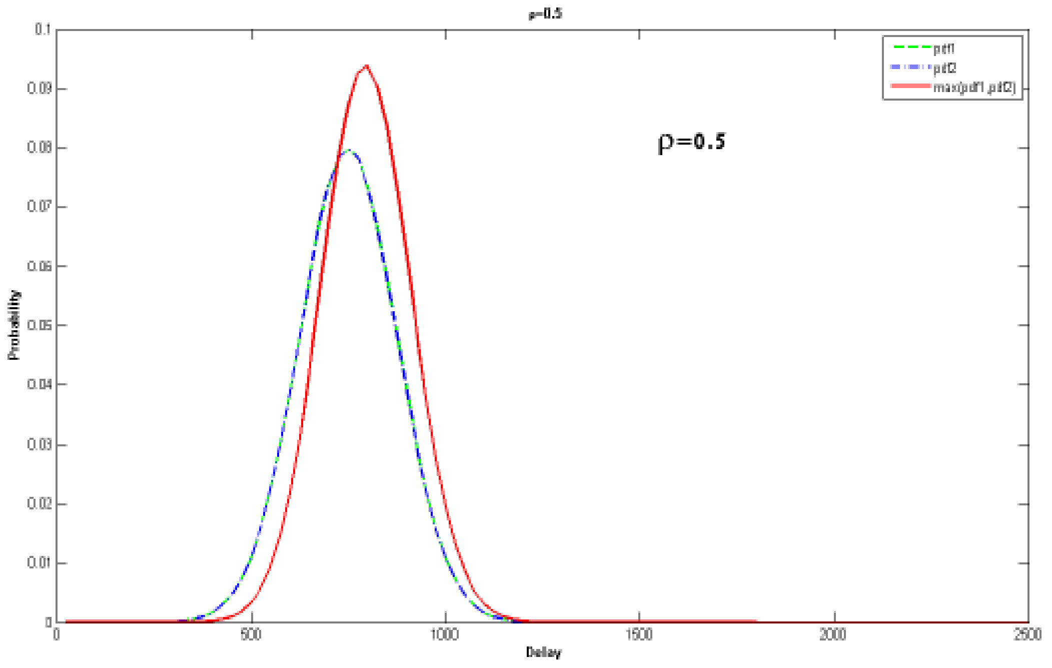


Figure 8: Independent ($\boldsymbol{\rho} = \mathbf{0.5}$).

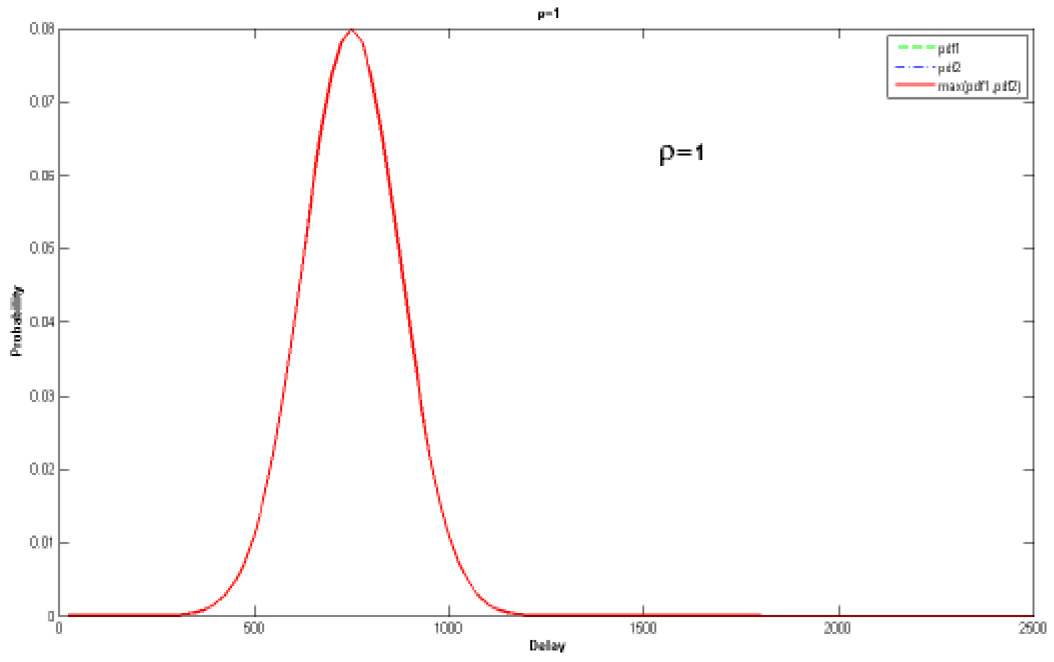


Figure 9: Independent ($\boldsymbol{\rho} = \mathbf{1}$).

Thus, the distribution of the maximum will be equal to either one of the two single path delay distributions (Figure 9). Because of the above results it can be concluded that the independent assumption will overestimate the delay after a maximum operation and the correlated assumption will yield smaller result. The converse is true about the delay for a single path. Therefore, assumptions may be based on circuit topology. For a shallow circuit the maximum operation will dominate the worst-case delay, hence the independent assumption might be used. For a significantly deeper circuit the delay of a single path will be dominant and, thus, the correlated assumption is expected to work.

# CHAPTER 3

# STATISTICAL STATIC TIMING ANALYSIS

In this chapter, we present an efficient statistical timing analysis algorithm that predicts the probability distribution of the circuit delay, considering both inter-die and intra-die variations, while accounting for the effects of spatial correlations in intra-die parameter variations. The procedure uses a first-order Taylor series expansion to approximate the gate and interconnect delays. Next, principal component analysis techniques are employed to transform the set of correlated parameters into an uncorrelated set. The statistical timing computation is then easily performed with a PERT-like circuit graph traversal using statistical *sum* and *max* functions.

## *3.1 Introduction*

As introduced in Chapter 1, conventional static timing analysis techniques handle the problem of variability by analyzing a circuit at multiple process corners. However, it is generally accepted that such an approach is inadequate, since the complexity of the variations in the performance space implies that if a small number of process corners is to be chosen, these corners must be very conservative/pessimistic as well as risky. For true accuracy, this can be overcome by using a larger number of process corners, but then the number of corners that must be considered for an accurate modeling will be too large for computational efficiency, and the method is also over-pessimistic as explained in Chapter 2.

The limitations of traditional static timing analysis techniques lie in their deterministic nature. An alternative approach that overcomes these problems is SSTA, which treats delays not as fixed numbers, but as probability density functions, taking the statistical distribution of parametric variations into consideration while analyzing the circuit.

In the literature, the statistical timing analysis approaches can be classified into continuous and discrete methods. Continuous methods [21, 23, 40, 43] use analytical approaches to find closed-form expressions for the PDF of the circuit delay. For simplicity, these methods often assume a normal distribution for the gate delay, but even so, finding the closed-from expression of the circuit distribution is still not an easy task. Discrete methods [22, 35, 38] are not limited to normal distributions, and can discretize any arbitrary delay distribution as a set of tuples, each corresponding to a discrete delay and its probability. The discrete probabilities are propagated through the circuit to find a discrete PDF for the circuit delay. However, this method is liable to suffer from the problem of having to propagate an exponential number of discrete point probabilities. In [29], an efficient method was proposed by modeling arrival times as cumulative density functions and delays as probability density functions and by defining operations of *sum* and *max* on these functions. Alternatively, instead of finding the distribution of circuit delay directly, several attempts have been made to find upper and lower bounds for the circuit delay distribution [22, 24, 40].

Statistical timing analyzers can also be categorized into path-based and block- based techniques. A path-based SSTA method, such as the works in [21, 30, 36, 40], enumerates all signal propagation paths or selective critical paths, finds the probability distribution of each individual path delay and then computes PDF of circuit delay by integration over all the paths in space. Although the computation of probability distribution for a single path is not difficult for arbitrarily distributed process parameter or arbitrary delay functions, the integration over all paths requires the joint probability density function of all paths and thus the correlation information among all paths must be computed which is of extremely high complexity. In addition, path-based methods suffer from the requirement that they may require the enumeration of paths: the number of paths can be exponential with respect to the circuit size. Therefore, such methods are not realistic for practical usage. A block-based SSTA method, such as [20, 23, 24, 29, 33, 35, 38, 43, 44], models delays of gates (wires) as random variables, and propagates/computes signal arrival times using *sum* and *max* operations similarly to propagating arrival times by a deterministic STA. Since block-based methods have linear run-times with respect to the circuit size and are good for incremental modes of operation, they are of the most interest.

Although many prior works have dealt with inter-die and intra-die variations, most of them have ignored intra-die spatial correlations by simply assuming zero correlations among devices on the chip [22, 23, 24, 25, 28, 29, 7, 34-36, 38]. The difficulty in considering spatial correlations between parameters is that it always results in complicated path correlation structures that are hard to deal with. Prior to our work of this chapter, very few studies have taken spatial correlations into consideration. The authors of [43] consider correlation between delays among the transistors inside a single gate (but not correlations between gates). The work in [36] uses a Monte Carlo sampling-based framework to analyze circuit timing on a set of selected sensitizable true paths. Another method in [40] computes path correlations on the basis of pair-wise gate delay covariances and used an analytic method to derive lower and upper bounds of circuit delay. The statistical timing analyzer in [26] takes into account capacitive coupling and intra-die process variation to estimate the worst case delay of critical path. Two parameter space techniques, namely, the parallelepiped method and the ellipsoid method, and a performance-space procedure, the binding probability method, were proposed in [32] to find either bounds or the exact distribution of the minimum slack of a selected set of paths. The approach in [21] proposes a model for spatial correlation and a method of statistical timing analysis to compute the delay distribution of a specific critical path. However, the probability distribution for a single critical path may not be a good predictor of the distribution of the circuit delay (which is the maximum of all path delays), as will be explained in Section 3.2. Moreover, the method may be computationally expensive when the number of critical paths is too large. In [20], the authors further extend their work in [21, 22] to compute an upper bound on the distribution of exact circuit delay.

In this chapter, we will propose a block-based SSTA method that computes the distribution of circuit delay while considering correlations due to path reconvergence as well as spatial correlations. We will model the circuit delay as a correlated multivariate normal distribution, considering both gate and wire delay variations.

In order to manipulate the complicated correlation structure, the principal component analysis technique is employed to transform the sets of correlated parameters into sets of uncorrelated ones. The statistical timing computation is then performed with a PERT-like circuit graph traversal. The complexity of the algorithm is $O(p \times n \times (N_g + N_I))$, which is linear in the number of gates $N_g$ and interconnects $N_I$ , and also linear in $p$, the number of spatially correlated random variables, and the number of grid squares, $n$, that are used to model variational regions. In other words, the cost is, at worst, $p \times n$ times the cost of a deterministic static timing analysis. We believe that this is the first method that can fully handle spatially correlated distributions under reasonably general assumptions, with a complexity that is comparable to traditional deterministic static timing analysis.

The remainder of this chapter is organized as follows. Section 3.2 formally formulates the problem to be solved in this work. The algorithm is presented in Section 3.3.

## *3.2 Problem Formulation*

Under process variations, parameter values, such as the gate length, the gate width, the metal line width and the metal line height, are random variables. Some of these variations, such as across-chip linewidth variations (ACLV) which are mainly caused by proximity and local effects [5], are deterministic, while others are random: this work will focus on the effect of random variations, and will model these parameters as random variables. The gate and interconnect delays, as functions of these parameters, also become random variables. Given appropriate modeling of process parameters or gate and interconnect delays, the task of SSTA is to find the PDF of the circuit delay.

The static timing works with the usual translation from a combinational circuit to a timing graph [42]. The node in this graph corresponds to the circuit primary inputs/outputs and gate input/output pins. The edges are of two types: one set corresponds to the pin-to-pin delay arcs within a gate, and the other set to interconnections from the drivers to receivers. The edges are weighted by the pin-to-pin gate delay, and interconnect delay, respectively. The primary inputs of the combinational circuit are connected to a virtual source node, and the primary outputs to a virtual sink node with directed virtual edges. In the case that primary inputs arrive at different times, the virtual edges from the virtual source to the primary inputs are assigned weights of the arrival times. Likewise, if the required times at the primary outputs are different, the weights of the edges from the outputs to the virtual sink are appropriately chosen.

For a combinational logic circuit, the problem of static timing analysis is to compute the longest path delay in the circuit from any primary input to any primary output, which corresponds to length of the longest path in the timing graph. In static timing analysis, the technique that is commonly referred to in the literature as PERT is commonly used[1]. This procedure starts from the source node to traverse the graph in a topological order and uses a *sum* operation or *max* operation (at a multi-fanin node) to find the longest path at the sink node.

Since we will employ a PERT-like traversal to analyze the distribution of circuit delay, we define a statistical timing graph of a circuit, as in the case of deterministic STA.

**Definition 1:**

*Let $G_s = (V, E)$ be a timing graph for a circuit with a single source node and a single sink node, where V is a set of nodes and E a set of directed edges. The graph $G_s$ is called a statistical timing graph if each edge i is assigned a weight $d_i$, where $d_i$, is a random variable, where the random variables may be uncorrelated or correlated. The weight associated with an edge corresponds to gate delay or interconnect delay. For a virtual*

[1] In reality, this is actually the critical path method (CPM) in operations research. However, we will persist with the term "PERT," which is widely used in the static timing analysis literature.

*edge, the weight is random variables with mean of its deterministic value and standard deviation of zero and it is independent from any other edges.*

**Definition 2:**

*Let a path $p_i$, be a set of ordered edges from the source node to the sink node in $G_s$ and $D_i$ be the path length distribution of $p_i$, computed as the sum of the weights $d_k$ for all edges k on the path. Finding the distribution of $D_{max} = max(D_1, \ldots\ldots, D_i, \ldots\ldots. D_{n_{paths}})$ among all paths (indexed from 1 to n paths) in the graph $G_s$ is referred to as the problem of SSTA of a circuit.*

Note that for the same nominal design, the identity of the longest path may change, depending on the random values taken by the process parameters. Therefore, finding the delay distribution of one critical path at a time is not enough, and correlations between paths must be considered in finding the maximum of the PDFs of all paths. Such an analysis is essential for finding the probability of failure of a circuit, which is available from the cumulative density function (CDF) of the circuit delay.

For an edge-triggered sequential circuit, the statistical timing graph can be constructed similarly by breaking the circuit into a set of combinational blocks between latches, and the analysis includes statistical checks on setup and hold time violations. The former requires the computation of the distribution of the maximum arrival time at the latches, which requires the solution of the SSTA problem as defined above. For intra-die variation, we only consider the impact of global and random components. However, the local component can also be accounted for in the proposed method, given, for instance, the chip layout and precharacterized spatial maps of parameters as in [41]. For transistors, we consider the following process parameters [39] as random variables: transistor length

$L_{eff}$ and width $W_g$, gate oxide thickness $T_{ox}$, doping concentration density $N_a$; for interconnect, at each metal layer, we consider the following parameters: metal width $W_{int_l}$, metal thickness $T_{int_l}$ and interlayer dielectric (ILD) thickness $H_{ILD}$, where the subscript $l$ represents that the random variable is of layer $l$, where $l = 1 \ldots\ldots n_{layers}$. However, the SSTA method presented in this chapter is general enough that it can be applied to handle variations in other parameters as well.

For spatial correlation, we use the grid-based model. It is assumed that nonzero correlations may exist only among the same type of process parameters in different grids, and there is no correlation between different types of process parameters. (Note here that we consider interconnect parameters in different layers to be "different types of parameters," e.g., $W_{int_1}$ and $W_{int_2}$ are uncorrelated[2].)

The process parameter values are assumed to be normally distributed random variables. The gate and interconnect delays, being functions of the fundamental process parameters, are approximated using a first-order Taylor series expansion. We will show that as a result of this, all edges in graph $G_s$ are normally distributed random variables. Since we consider spatial correlations of the process parameters, it turns out that some of the delays are correlated random variables. Furthermore, the circuit delay $D_{max}$ is modeled as a multivariate normal distribution. Although the closed form of circuit delay distribution is not normal, we show that the loss of accuracy is not significant under this approximation.

[2] This assumption is not critical to the correctness of our procedure, but is used in our experiment results. Out method is general enough to handle corrections between parameters of different types.

## *3.3 SSTA Algorithm*

The core SSTA method is described in this section, and its description is organized as follows. At first, in Section 3.3.1, we will describe how we model the distributions of gate and interconnect delays as normal distributions, given the PDFs that describe the variations of various parameters. In general, these PDFs will be correlated with each other. In Section 3.3.2, we will show how we can simplify the complicated correlated structure of parameters by orthogonal transformations. Section 3.3.3 will describe the PERT-like traversal algorithm on the statistical timing graph by demonstrating the procedure for the computation of *max* and *sum* functions.

### *3.3.1 Modeling Gate/Interconnect Delay PDFs*

In this section, we will show how the variations in the process parameters are translated into PDFs that describe the variations in the gate and interconnect delays that correspond to the weights on edges of the statistical timing graph.

Before we introduce how the distributions of gate and interconnect delays will be modeled, let us first consider an arbitrary function $d = f(\vec{P})$ that is assumed to be a function on a set of process parameters $\vec{P}$ , where each $p_i \in \vec{P}$ is a random variable with a normal distribution given by $p_i \sim N(\mu_{p_i}, \sigma_{p_i})$.

We can approximate the function *d* linearly using a first order Taylor expansion:

$$d = d_0 + \sum_{\forall\, parameters\, p_i} \left[\frac{\partial f}{\partial p_i}\right] \Delta p_i \qquad (3.1)$$

where $d_0$ is the nominal value of $d$, calculated at the nominal values of process parameters in the set $\vec{P}, \frac{\partial f}{\partial p_i}$ is computed at the nominal values $p_i, \Delta p_i = p_i - \mu_{p_i}$ is a normally distributed random variable and $\Delta p_i \sim N(0, \sigma_{p_i})$.

In this approximation, $d$ is modeled as a normal distribution, since it is a linear combination of normally distributed random variables. Its mean $\mu_d$, and variance $\sigma_d^2$ are:

$$\mu_d = d_0 \tag{3.2}$$

$$\sigma_d^2 = \sum_{\forall i} \left[\frac{\partial f}{\partial p_i}\right]_0^2 \sigma_{p_i}^2 + 2\sum_{\forall i \neq j} \left[\frac{\partial f}{\partial p_i}\right]_0 \left[\frac{\partial f}{\partial p_i}\right]_0 cov(p_i, p_j) \tag{3.3}$$

Where $cov(p_i, p_j)$ is the covariance of $p_i$ and $p_j$.

It is reasonable to ask whether the approximation of $d$ as a normal distribution is valid, since the distribution of $d$ may, strictly speaking, not be Gaussian. We can say that when $\Delta p_i$has relatively small variations, the first order Taylor expansion is adequate and the approximation is acceptable with little loss of accuracy. This is generally true of intra-die variations, where the process parameter variations are relatively small in comparison with the nominal values. For this reason, as functions of process parameters, the gate and interconnect delays can be approximated as a sum of normal distributions (which is also normal) applying the Equation (3.1).

***Computing the PDF of interconnect delay***

In this work, we use the Elmore delay model [42] for simplicity to calculate the interconnect delays[3]. Under the Elmore model, the interconnect delay is a function of the resistances $\vec{R}_w$ and capacitances $\vec{C}_w$ of all wire segments in the interconnect tree and input load capacitances $\vec{C}_g$ of the fanout gates, or receivers.

$$d_{int} = d(\vec{R}_w, \vec{C}_w, \vec{C}_g) \tag{3.4}$$

Since the resistances and capacitances above are furthermore decided by the process parameters $\vec{P}$ of the interconnect and the receivers, such as $W_{int_1}$, $T_{int_1}$, $H_{ILD}$,

[3] However, it should be emphasized that any delay model may be used, and all that is required is the sensitivity of the delay to the process parameters. For example, through a full circuit simulation, the sensitivity may be computed by performing ad joint sensitivity analysis.

$W_g$, $L_{eff}$ and $T_{ox}$, the sensitivities of the interconnect delay to a process parameter $p_i$ can be found by using the chain rule:

$$\frac{\partial d_{int}}{\partial p_i} = \sum_{\forall R_{w_k} \in \vec{R}_w} \frac{\partial d}{\partial R_{w_k}} \frac{\partial R_{w_k}}{\partial p_i} + \sum_{\forall C_{w_k} \in \vec{C}_w} \frac{\partial d}{\partial C_{w_k}} \frac{\partial C_{w_k}}{\partial p_i} + \sum_{\forall C_{g_k} \in \vec{C}_g} \frac{\partial d}{\partial C_{g_k}} \frac{\partial C_{g_k}}{\partial p_i} \qquad (3.5)$$

The distribution of interconnect delay can then be approximated on the computed sensitivities.

We will now specifically consider the factors that affect the interconnect delay associated with edges in the statistical timing graph. Recall that under our model, we divide the chip area into grids so that the process parameter variations within a grid are identical, but those in different grids exhibit spatial correlations. Now consider an interconnect tree with several different segments that reside in different grids. The delay variations in the tree are affected by the process parameter variations of wires in all grids that the tree traverses.

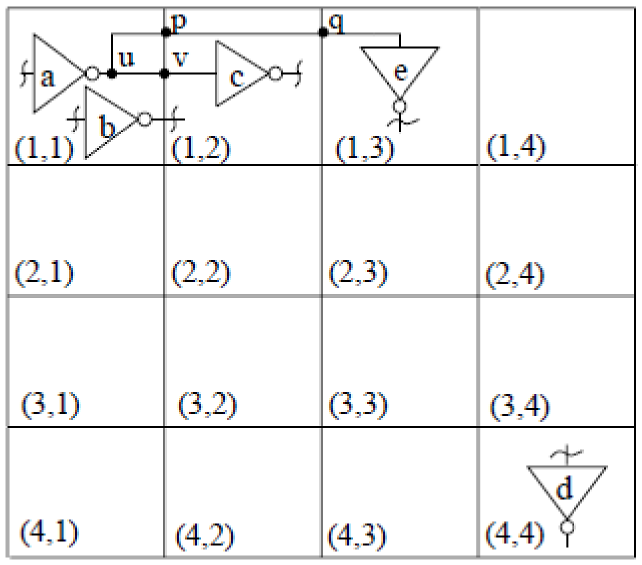


Figure 10: Grid model for spatial correlations.

For example, in Figure 10, consider the two segments *uv* and *pq* in the interconnect tree driven by gate *a*. Segment *uv* passes through the grid (1, 1) and *pq* through the grid (1, 2). Then the resistance and capacitance of segment *uv* should be calculated based on the process parameters of grid (1, 1), while the resistance and capacitance of segment *pq* should be based on those of grid (1, 2). Hence, the distribution of the interconnect tree delay is actually a function of random variables of interconnect parameters in both grid (1, 1) and grid (1, 2), and should incorporate any correlations between these random variables. Similarly, if the gates that the interconnect tree drives reside in different grid locations, the interconnect delay to any sink is also a function of random variables of gate process parameters of all grids in which the receivers are located.

In summary, the distribution of interconnect delay function can be approximated

$$d_{int} = d_{int}^0 + \sum_{i\subset\Gamma_g}\left[\frac{\partial d}{\partial L_{eff}^i}\right]_0 \Delta L_{eff}^i + \sum_{i\subset\Gamma_g}\left[\frac{\partial d}{\partial W_g^i}\right]_0 \Delta W_g^i + \sum_{i\subset\Gamma_g}\left[\frac{\partial d}{\partial T_{ox}^i}\right]_0 \Delta T_{ox}^i + \sum_{l=1}^{n_{layer}}\left\{\sum_{i\subset\Gamma_{int}}\left[\frac{\partial d}{\partial W_{int_1}^i}\right]_0 \Delta W_{int_l}^i + \sum_{i\subset\Gamma_{int}}\left[\frac{\partial d}{\partial T_{int_l}^i}\right]_0 \Delta T_{int_l}^i + \sum_{i\subset\Gamma_{int}}\left[\frac{\partial d}{\partial H_{ILD_l}^i}\right]_0 \Delta H_{int_l}^i\right\} \tag{3.6}$$

where $d_{int}^0$ is the interconnect delay calculated with nominal values of process parameters, $\Gamma_g$is the set of indices of grids that all the receivers reside in, $\Gamma_{int}$ is the set of indices of grids that the interconnect tree traverses, and $\Delta L_{eff}^i = L_{eff}^i - \mu_{L_{eff}^i}$ where $L_{eff}^i$ is the random variable representing transistor length in the $i^{th}$grid. The parameters $\Delta w_g^i$, $\Delta T_{ox}^i$, $\Delta w_{int_l}^i$, $\Delta T_{int_l}^i$ and $\Delta H_{ILD_l}^i$ are similarly defined . As before, the subscript "o" next to each sensitivity represents the fact that it is evaluated at the nominal point.

***Computing the PDF of gate delay and output signal transition time***

The distribution of gate delay and output signal transition time at the gate output can be approximated in a similar manner as described above, given the sensitivities of the gate delay to the process parameters.

Consider a multiple-input gate, let $d_{gate}^{pin_i}$ be the gate delay from the $i^{th}$ to the output and $S_{out}^{pin_i}$ be the corresponding output signal transition time. In general, both $d_{gate}^{pin_i}$ and $S_{out}^{pin_i}$ can be written as a function of the process parameters $\vec{P}$ of the gate, the loading capacitance of the driving interconnect tree $\overrightarrow{C_w}$ and the succeeding gates that it drives $\overrightarrow{C_g}$ , and the input signal transition time $S_{in}^{pin_i}$ at this input pin of the gate

$$d_{gate}^{pin_i}=D_{gate}(\vec{P},\vec{C}_w,\vec{C}_g,S_{in}^{pin_i}) \tag{3.7}$$

$$S_{out}^{pin_i}=S_{gate}(\vec{P},\vec{C}_w,\vec{C}_g,S_{in}^{pin_i}) \tag{3.8}$$

The distributions of $d_{gate}^{pin_i}$ and $S_{in}^{pin_i}$ can be approximated as Gaussians using linear expressions of parameters, where the mean values of $d_{gate}^{pin_i}$ and $S_{in}^{pin_i}$ can be found by using the mean values of $\vec{P}$, $\vec{C}_w$, $\overrightarrow{C_g}$ and $S_{in}^{pin_i}$ functions $D_{gate}$ or $S_{gate}$ , and the sensitivities of either $d_{gate}^{pin_i}$ or $S_{in}^{pin_i}$ to process parameters can be computed applying the chain rule. The derivatives of $\vec{C}_w$ and $\vec{C}_g$ to the process parameters can be easily computed, as $\vec{C}_w$ and $\vec{C}_g$ are functions of process parameters. The input signal transition time, $S_{in}$, is a function of the output transition time of the preceding gate and the delay of the interconnect connecting the preceding gates and this gate, where both interconnect delay (as discussed earlier) and output transition time of the preceding gate (as will be shown in the next paragraph) are Gaussian random variables that can be expressed as a linear function of parameter variations. Therefore, at a gate input, the input signal transition time $S_{in}$ is always given as a normally distributed random variable with mean and first-order sensitivities to the parameter variations.

To consider the effect of the transition time of an input signal on the gate delay, the output signal transition time $S_{out}$ at each gate output must be computed in addition to pin-to-pin delay of the gate. In conventional static timing analysis, $S_{out}$ is set to $S_{out}^{pin_i}$ if the path ending at the output of the gate traversing the $i^{th}$ input pin has the longest path delay $d_{path_i}$. In SSTA, each of the paths through different gate input pins has a certain probability to be the longest path. Therefore, $S_{out}$ should be computed as a weighted sum of the distributions of $S_{out}^{pin_i}$, where the weight equals the probability that the path through the $i^{th}$ pin is the longest among all others:

$$S_{out} = \sum_{\forall\, input\, pin\, i} \left\{ Prob\left[d_{path_i} > max_{\forall i \neq j}(d_{path_i})\right] \times S_{out}^{pin_i} \right\} \tag{3.9}$$

where $d_{path_i}$ is the random path delay variable at the gate output through the $i^{th}$ input pin. The result of $max_{\forall i \neq j}(d_{path_i})$ is a random variable representing for the distribution of maximum of multiple paths. As will be discussed later in Section 3.3.3, $d_{path_i}$ and $max_{\forall i \neq j}(d_{path_i})$ can be approximated as Gaussian using *sum* and *max* operators, and their correlation can easily be computed. Therefore, finding the value of $Prob\left[d_{path_i} > max_{\forall i \neq j}(d_{path_i})\right]$, i.e., $Prob\left[d_{path_i} > max_{\forall i \neq j}(d_{path_i} > 0)\right]$ becomes computing the probability of a Gaussian random variable greater than zero, which can easily be found from a look-up table. As each $S_{out}^{pin_i}$ is a Gaussian random variable in linear combination of the process parameter variations, $S_{out}$ is therefore also a Gaussian-distributed random variable and its sensitivities to all process parameters $\frac{\partial S_{out}}{\partial p_i}$ can easily be found from its linear expression.

### 3.3.2 *Orthogonal Transformation of Correlated Variables*

In statistical timing analysis without spatial correlations, correlations due to reconvergent paths have long been an obstacle. When the spatial correlation of process parameters is also taken into consideration, the correlation structure becomes even more complicated. To make the problem tractable, we use the Principal Component Analysis (PCA) technique [7] to transform the set of correlated parameters into an uncorrelated set. PCA is a method that can be employed to examine the relationship among a set of correlated variables. Given a set of correlated random variables $\vec{X}$ with a covariance matrix *R*, PCA can transform the set $\vec{X}$ into a set of mutually orthogonal random variables,$\overrightarrow{X'}$, such that each member of $\overrightarrow{X'}$ has zero mean and unit variance.

The elements of the set $\overrightarrow{X'}$ _ are called principal components in PCA, and the size of $\overrightarrow{X'}$ is no larger than the size of $\overrightarrow{X'}$. Any variable $x_i \in \overrightarrow{X'}$ can then be expressed in terms of the principal components $\overrightarrow{X'}$ as follows:

$$x_i = \left(\sum_j \sqrt{\lambda_j} \,.\, v_{ij} . x'_j\right)\sigma_i + \mu_i \tag{3.10}$$

where $x'_j$ is a principal component in set $\overrightarrow{X'}$, $\lambda_j$ is the $j^{th}$ eigenvalue of the covariance matrix *R*, $v_{ij}$ is the $i^{th}$ element of the $j^{th}$ eigenvector of *R*, and $\sigma_i$ and $\mu_i$ are respectively, the mean and standard deviation of $x_i$.

Since we assume that different types of parameters are uncorrelated, we can group the random variables of parameters by types and perform principal component analysis in each group separately, i.e., we compute the principal components for

$\vec{L}_{eff}$ , $\vec{W_g}$ , $\vec{T}_{ox}$ , $\vec{N}_a$ , $\vec{W}_{int_l}$ and $\vec{T}_{int_l}$ individually. Clearly, not only are the principal components of the same type of parameters independent, but so are the principal components of different type of parameters.

For instance, let $\vec{L}_{eff}$ be a random vector representing transistor gate length variations in all grids and it is of multivariate normal distribution with covariance matrix $R_{L_{eff}}$. Let $\overrightarrow{L'}_{eff}$ be the set of principal components computed by PCA. Then any $L^i_{eff} \in \overrightarrow{L'}_{eff}$ representing the variation of transistor gate length of the $i^{th}$ grid can then be expressed as a linear function of the principal components

$$L^i_{eff} = \mu_{L^i_{eff}} + a_{i1} \times L'^1_{eff} + \cdots + a_{it} \times L'^t_{eff} \tag{3.11}$$

where $\mu_{L^i_{eff}}$ is the mean of $L^i_{eff}$, $L'^1_{eff}$ is a principal component in $\overrightarrow{L'}_{eff}$ all $L'^1_{eff}$ are independent with zero means and unit variances, and *t* is the total number of principal components in $\overrightarrow{L'}_{eff}$.

In this way, any random variable in $\vec{W_g}$,$\vec{T}_{ox}$,$\vec{N}_a$ ,$\vec{W}_{int_l}$, $\vec{T}_{int_l}$ and $\vec{H}_{ILD_l}$ can be expressed as a linear function of the corresponding principal components in $\overrightarrow{W'}_g$,$\overrightarrow{T'}_{ox}$,$\overrightarrow{N'}_a$ ,$\overrightarrow{W'}_{int_l}$, $\overrightarrow{T'}_{int_l}$ and $\overrightarrow{H'}_{ILD_l}$.Superposing the set of rotated random variables of parameters on the original random variables in gate or interconnect delay in Equation (3.6), the expression of gate or interconnect delay is then changed to the linear combination of principal components of all parameters

$$d = d_0 + k_1 \times p'_1 + \cdots + k_m \times p'_m \tag{3.12}$$

Where $p'_i \in \overrightarrow{P'}_i$ and $\overrightarrow{P'} = \overrightarrow{L'}_{eff} \cup \overrightarrow{W'}_g \cup \overrightarrow{T'}_{ox} \cup \overrightarrow{N'}_a \cup \overrightarrow{W'}_{int_l} \cup \overrightarrow{T'}_{int_l} \cup \overrightarrow{H'}_{ILD_l}$ and $m$ is the size of $\overrightarrow{P'}$. Note that all of the principal components $p'_i$ that appear in Equation (3.12) are independent. Equation (3.12) has the following properties:

**Property 1** Since all $p'_i$ are orthogonal, the variance of *d* can be simply computed as

$$\sigma_d^2 = \sum_{i=1}^{m} k_i^2 \tag{3.13}$$

**Property 2** The covariance between *d* and any principal component $p'_i$ is given by

$$cov(d, p'_i) = k_i \sigma_{p'_i}^2 = k_i \tag{3.14}$$

In other words, the coefficient of $p'_i$ is exactly the covariance between *d* and $p'_i$

**Property 3** Let $d_i$and $d_j$be two random variables:

$$d_i = d_i^0 + k_{i1} \times p'_i + \cdots . + k_{im} p'_m \tag{3.15}$$

$$d_j = d_j^0 + k_{j1} \times p'_j + \cdots . + k_{jm} p'_m \tag{3.16}$$

The covariance of $d_i$ and $d_j$,$con$ ($d_i$,$d_j$) can be computed by

$$cov(d_i, d_j) = \sum_{r=1}^{m} k_{ir}, k_{jr} \tag{3.17}$$

### *3.3.3 PERT-like Traversal of SSTA*

Using the techniques discussed up to this point, all edges of the statistical timing graph may be modeled as normally distributed random variables. In this section, we will describe a procedure for finding the distribution of the statistical longest path in the graph.

In conventional deterministic STA, the PERT algorithm can be used to find the longest path in a graph by traversing it in topological order using two types of functions:

• The *sum* function, and

• The *max* function.

In our statistical timing analysis, a PERT-like traversal is employed to find the distribution of circuit delay. However, unlike deterministic STA, the *sum* and *max* operations here are functions of a set of correlated multivariate Gaussian random variables instead of fixed values:

1) $d_{sum} = \sum_{i=1}^{l} d_i$, and
2) $d_{max} = max(d_1, d_2)$.

where $d_i$ is a Gaussian random variable representing either gate delay or wire delay expressed as linear functions of principal components in the form of Equation (3.15), and $l$ is the number of random variables that *sum* or *max* function is operating on.

***Computing the distribution of the sum function***

The computation of the distribution of sum function is simple. Since the $d_{sum} = \sum_{i=1}^{l} d_i$ is a linear combination of normally distributed random variables, $d_{sum}$ is a normal distribution. The mean $\mu_{dsum}$ and variance $\sigma_{dsum}^2$ of the sum are given by

$$\mu_{dsum} = \sum_{i=1}^{l} d_i^0 \tag{3.18}$$

$$\sigma_{dsum}^2 = \sum_{j=1}^{m} \left( \sum_{i=1}^{l} k_{ij} \right)^2 \tag{3.19}$$

***Computing the distribution of the max function***

The *max* function of $l$ normally distributed random variables $d_{max} = max(d_1, \dots, d_l)$) is, strictly speaking, not Gaussian. However, we have found that, in practice, it can be approximated closely by a Gaussian. This idea is similar in spirit to Berkelaar's approach in [23, 31], although it is more general since Berkelaar's work restricted its attention to delay random variables that were uncorrelated[4]. In this work, we use the Gaussian distribution to approximate the result of a *max* function, so that

[4] Many researchers in the community were well aware of Berkelaar's results as early as 1997, though his work did not appear as an archival publication.

$d_{max} \sim N(\mu_{d_{max}}, \sigma_{d_{max}})$. We also approximate $d_{max}$ as a linear function of all the principal components, $p'_1 \dots p'_m$.

$$d_{max} = \mu_{d_{max}} + a_1 p'_1 + \cdots + a_m p'_m \tag{3.20}$$

Therefore, determining this approximation for $d_{max}$ is equivalent to finding the values of $\mu_{d_{max}}$ and all $a_i$'s.

From Property 2 of Section 3.3.2, we know that the coefficient $a_r$ equals $cov(d_{max}, p'_r)$. Then the variance of the expression on the right hand side of Equation (3.20) is computed as $s_0^2 = \sum_{r=1}^{m} cov^2(d_{max}, p'_r)$. Since this is merely an approximation, there may be a difference the value $s_0^2$ and the actual variance $\sigma_{d_{max}}^2$ of $d_{max}$. To diminish the difference, we can normanizes the value of $a_r$ by setting it as

$$a_r = cov(d_{max}, p'_r) . \frac{\sigma_{d_{max}}}{s_0} \tag{3.21}$$

We can see now that to find the linear approximation for $d_{max}$., The values of $\mu_{d_{max}}, \sigma_{d_{max}}$ and $cov(d_{max}, p_i)$ are required. In the work of [43], similar inputs were required in their algorithm and the results from [27] were applied and seen to provide good results. In this work, we have borrowed the same analytical formula from [27] for the computation of the *max* function.

According to [27], if $\xi$ and η are two random variables, $\xi \sim N(\mu_1, \sigma_1), \eta \sim N(\mu_2, \sigma_2)$, with a correlation coefficient of $r(\xi, \eta) = \rho$, then the mean $\mu_t$ and the variance $\sigma_t^2$ of $t = max(\xi, \eta)$ can be approximated by

$$\mu_t = \mu_1 . \phi(\beta) + 2 . \phi(-\beta) + \alpha . \varphi(\beta) \tag{3.22}$$

$$\sigma_t^2 = (\mu_1^2 + \sigma_1^2) . \phi(\beta) + (\mu_2^2 + \sigma_2^2) . \phi(-\beta) + (\mu_1 + \mu_2) . \alpha . \varphi(\beta) - \mu_t^2 \tag{3.23}$$

Where $\alpha = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2\rho}$ (3.24)

$$\beta = \frac{(\mu_1 - \mu_2)}{\alpha} \tag{3.25}$$

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} exp\left[-\frac{x^2}{2}\right] \tag{3.26}$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} exp\left[-\frac{y^2}{2}\right] dy \tag{3.27}$$

The formula will not apply if $\sigma_1 = \sigma_2$ and $\rho = 1$. However, in this case, the *max* function is simply identical to the random variable with largest mean value.

Moreover, from [27], if $\gamma$ is another normally distributed random variable and the correlation coeeficients $r(\xi, \gamma) = \rho_1, r(\eta, \gamma) = \rho_2$ then the correlation between $\gamma$ and $t = max(\xi, \eta)$ can be obtained by

$$r(\xi, \gamma) = \frac{\sigma_1.\rho_1.\phi(\beta) + \sigma_2.\rho_2.\phi(-\beta)}{\sigma_t} \tag{3.28}$$

Using the formula above, we can find all the values required. As an example, let us see how this can be done by first starting with a two-variable *max* function, $d_{max} = \max(d_i, d_j)$. Let $d_{max}$ be of the form of Equation (3.20). We can find the approximation of $d_{max}$ as follows:

1. Given the expression of $d_i$ and $d_j$ , each is linear combinations of the principal components, compute their mean and standard deviation values $\mu_{d_i}, \sigma_{d_i}$ and $\mu_{d_j}, \sigma_{d_j}$, respectively, as described in *Property 1* of section 3.3.2.
2. Find the correlation coefficient between $d_i$and $d_j$. The covariance of $d_i$ and $d_j$ $cov(d_i, d_j)$ , can be computed using *Property 3* in Section 3.3.2. Now if $r(d_i, d_j) = 1$ and $\sigma_{d_i} = \sigma_{d_j}$, set $d_{max}$ to be identical to $d_i$ or $d_j$, whichever has larger mean value and we can stop here; otherwise, we will continue to the next step.

3. Calculate the mean $\mu_{d_{max}}$ and variance $\sigma^2_{d_{max}}$ of $d_{max}$ using Equations (3.22) and (3.23).
4. Find all coefficients $a_r$ of $p'_r$. According to *Property 2*, $a_r = cov(d_{max}, p'_r)$, also $cov(d_i, p'_r) = k_{ir}$ and $cov(d_j, p'_r) = k_{jr}$. Applying Equation (3.28), the values of $cov(d_{max}, p'_r)$ and thus $a_r$ can be calculated.
5. After all of the $a_r$'s have been calculated, determine $s_0 = \sqrt{\sum_{r=1}^{m} a_r^2}$. Normalize the coefficient by resetting each $a_r = a_r \frac{\sigma_{d_{max}}}{s_0}$.

The calculation of the two-variable *max* function can easily be extended to a multi-variable *max* function by repeating the steps of the two-variable case recursively.

As mentioned at the beginning of this section, *max* of two Gaussian random variables is not strictly Gaussian. This approximation can sometimes introduce serious error, e.g., when the two Gaussian random variables have the same mean and standard deviation and correlation value of -1, and the distribution of the maximum is a half Gaussian. During the computation of multi-variable max function, some inaccuracy could be introduced since we approximate the *max* function as normal even though it is not really normal, and proceed with further recursive calculations.

To the best of our knowledge, there is no theoretical analysis available in the literature that quantifies the inaccuracies when a normal distribution is used to approximate the maximum of a set of Gaussian random variables. However, a numerically based analysis was provided in [27], which suggests that in some situations the errors can be great, but for many applications this approximate is quite satisfactory. Moreover, recall that we have a "normalization" step to diminish the difference between the variance computed from the linear form of *max* approximation and the real variance of the *max* function. As in the case of approximating the *max* as normal distribution, there is no theoretical proof about how this "normalization" step can affect the accuracy of the approximation. Another option to diminish the difference is to move it into an independent random Gaussian component, and it is difficult to state definitively which of these options is better. In our work, we choose the former option and find that it provides excellent accuracy, where the statistics of the "normalization" ratio for several test circuits are provided.

At this point, not only the edges, but also the results of *sum* and *max* functions are expressed as linear functions of the principal components. Therefore, using a PERT traversal by incorporating the computation of sum and max functions described above, the distribution of arrival time at any node in the timing graph becomes a linear function of principal components, and so the distribution of circuit delay can be computed at the virtual sink node.

The overall flow of our algorithm is shown in Figure 11. It is noticed that this work is in some sense parallel to the work of [32]: in [32], delays are represented as linear combinations of global random variables, while in our work, they are linear functions of principal components; in [32], the max of delays are reexpressed as linear functions using binding probabilities, while in our work, the linear functions are found by an analytical method from [27].

To further speed up the process, the following technique may be used: During the max operation of SSTA, if the value of $\mu + 3.\sigma$ of one path has a lower delay than the value of $\mu - 3.\sigma$ of another path, we can simply calculate the max function ignoring the former path.

*Input: Process parameter variations*

*Output: Distribution of circuit delay*

*1. According to the size of the chip, partition the chip region into n = nrow× ncol grids.*

*2. For each type of parameter, determine the n jointly normally distributed random variables and the corresponding covariance matrix.*

*3. Perform an orthogonal transformation to represent each random variable with a set of principal components.*

*4. For each gate and net connection, model their delays as linear combinations of the principal components generated in step 3.*

*5. Map the circuit into a statistical timing graph by adding one virtual-source node, one virtual-sink node and corresponding edges.*

*6. Using sum and max functions on Gaussian random variables, perform a PERT-like traversal on the graph to find the distribution of the statistical longest path. This distribution achieved is the circuit delay distribution*

Figure 11: Overall flow of our statistical timing analysis.

# CHAPTER 4

# INCORPORATING NON-GAUSSIAN DISTRIBUTED PROCESS PARAMETERS AND NONLINEAR DELAY FUNCTIONS

In this chapter, we present a general framework and an efficient method of block-based SSTA that can deal with process variations with non-Gaussian distributions, and/or delay functions with nonlinear dependencies on process parameter variations. We extend techniques for evaluating the s*um* and *max* functions in SSTA from the linear, Gaussian case of Chapter 3, to the nonlinear, non-Gaussian case. The proposed approach is shown to be accurate and efficient in predicting timing characteristics and yield of circuit.

## *4.1 Introduction*

In Chapter 3, we proposed an efficient method for timing analysis under process variations, under the assumption that where all process variations have or can be approximated by Gaussian distributions and all delays have linear sensitivities to the process parameters. But there are two limitations to this approach. First, although some types of distributions can be approximated by a Gaussian, others may display asymmetric types of distributions (e.g., lognormal distributions), or symmetric types of non-Gaussian distributions (e.g., uniform distributions) that cannot be well-approximated by a Gaussian. For example, via resistance is known to have an asymmetric probability distribution. A second issue is related to the use of a first-order Taylor expansion to approximate a delay function as a linear function of the variations of process parameters. The linear approximation can only be justified under the assumption that variations are small. With technology scaling, as the percentage change in process variations becomes larger, delays may show nonlinear dependencies on some sources of variations.

Therefore, it is desirable to develop SSTA techniques that can deal with non-Gaussian-distributed process parameters and/or nonlinear effects on gate [wire] delays[5], in order to obtain sufficiently accurate results for analyzing the timing yield.

### *4.2 Framework for Handling Non-Gaussian and/or Non-linear Function Parameters*

As we know that SSTA approach is parameterized block-based method. Any gate or wire delay is presented as a linear function of process variations, and this representation is referred to as a first-order canonical form in [44]:

$$A = a_0 + \sum_{i=1}^{n} a_i . \Delta X_i + a_{n+1} . \Delta R_a \tag{4.1}$$

Here, $a_0$ is the mean or nominal delay, and $\Delta X_i = X_i - \widehat{X_i}$ is variation of process parameter $X_i$, centralized by subtracting its mean value $\widehat{X_i}$. Each $\Delta X_i$ represents for a global source of variation that has a global effect on all delays, and is modeled as a Gaussian random variable $N(0, \sigma_{X_i})$; all $\Delta X_i$ variables are mutually independent. The coefficient $a_i$ is the sensitivity of delay to $X_i$, and $\Delta R_a$ is the variation of local uncertainty that only affects the delay locally, and is modeled as a normalized Gaussian random variable that is independent of all other sources of variations. The sensitivity of the delay to $R_a$ is given by $a_{n+1}$.

#### *4.2.1 A Generalized Canonical Form for the Delay*

A generalized canonical form of gate or wire delay is defined by extending the form of (4.1) as follows:

$$A = a_0 + \sum_{i=1}^{n_{LG}} a_{LG,i} . \Delta X_{LG,i} + f_A(\Delta X_N) + a_{n+1} . \Delta R_a \tag{4.2}$$

[5] For conciseness, in the remainder of the thesis, we will use the term “non-Gaussian parameter” to refer to a non-Gaussian-distributed process parameter, and “nonlinear function parameter” to a process parameter whose variation has nonlinear effects on delays.

Here $a_0$ is the mean value of the delay, $\Delta X_{LG} = \{X_{LG,1}, X_{LG,2}, \dots X_{LG,n_{NLG}}\}$ is the set of random variables for the global sources of variation that are both Gaussian-distributed and have linear effects on delay, and $n_{LG}$ is number of such types of variations. The sensitivity of the delay to $\Delta X_{LG,i}$ is given by $a_{LG,i}$. We also define a set of random variables, of cardinality $n_{NLG}, \Delta X_N = \{\Delta X_{N,1}, \Delta X_{N,2}, \dots \Delta X_{N,n_{NLG}}\}$. The elements of this set correspond to the global sources of variations that are non-Gaussian-distributed or have nonlinear effects on the delay, and $f_A$ is a function describing the dependence of the delay on non-Gaussian and nonlinear function parameters, with a mean value that is normalized to zero. Finally, $\Delta R_a$ is a normalized Gaussian parameter that represents local sources of variations, and $a_{n+1}$ is its sensitivity to the delay.

The generalized canonical form differs from the original first-order canonical form of delay only in the term $f_A(\Delta X_N)$ that describes dependencies of A on non-Gaussian and nonlinear function parameters. For convenience, this term is referred to as a non-Gaussian nonlinear term in this chapter. Note that $f_A$ can be either a nonlinear function of non-Gaussian-distributed process parameters, or a linear function of non-Gaussian process parameters, or a nonlinear function of Gaussian process parameters. The function $f_A$ can be a function of arbitrary type, and the non-Gaussian parameters can have any arbitrary probability density function. For numerical computations, nonlinear functions and non-Gaussian distributions can be specified by tables.

### *4.2.2 The Computation of the sum Function*

As in the case for first-order canonical forms, it is straightforward to compute the *sum* function for two random variables, each specified in generalized canonical form.

If C = A + B, where A and B are both in generalized canonical form, then C can also be expressed in a generalized canonical form, with its coefficients specified by:

$$c_0 = a_0 + b_0 \tag{4.3}$$

$$c_{LG,i} = a_{LG,i} + b_{LG,i} \quad (1 < i < n_{LG}) \tag{4.4}$$

$$f_c(\Delta X_N) = f_A(\Delta X_N) + f_B(\Delta X_N) \tag{4.5}$$

The computation of $c_0$ and each $c_{LG,i}$ is simple. The term $f_c(\Delta X_N)$ is obtained by computing the sum of the non-Gaussian nonlinear terms of A and B. In practice, this can be computed by numerically summing the tables describing $f_A(\Delta X_N)$ and $f_B(\Delta X_N)$.

### *4.2.3 The Computation of the max Function*

It is necessary to use an approximation in computing the max of two random variables, each specified in generalized canonical form. In order to preserve the correlations of delays, a random variable $C_{app}$ in generalized canonical form is used to approximate $C = max(A, B)$. The framework for computing $C_{app}$ can be applied here, by using the concept of tightness probability:

$$c_0 = E[max(A, B)] \tag{4.6}$$

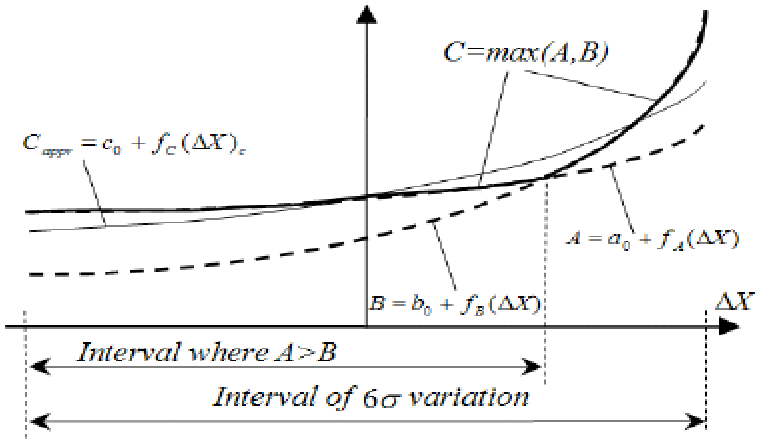
$$c_{LG,i} = T_A a_{LG,i} + (1 - T_A) b_{LG,i}\ , \qquad \text{for} \qquad (1 < i < n_{LG}) \tag{4.7}$$

$$f_c(\Delta X_N) = T_A f_A(\Delta X_N) + (1 - T_A) f_B(\Delta X_N) \tag{4.8}$$

As in the case for first-order canonical form, this approximation for the maximum of two generalized canonical forms is a linear approximation: $c_0$ is matched with the exact mean value of $C = max(A, B)$ ; $C_{app}$ is a linear combination of A and B using the tightness probabilities, where the coefficient $c_{LG,i}$ is computed as a linear combination of coefficients $a_{LG,i}$ and $b_{LG,i}$ and the non-Gaussian nonlinear term $f_C$ as a linear combination of functions $f_A$ and $f_B$, weighted by the corresponding tightness probabilities $T_A$ and $T_B$, respectively. The sensitivity coefficient $c_{n+1}$ for the local independent source of variations is computed so as to make the variance of $C_{app}$ equal to the variance of the exact maximum $C = max(A, B)$, where the exact variance $\sigma_C^2$ is expressed through the mean and the second moment as:

$$\sigma_C^2 = E[max(A, B)^2] - (E[max(A, B)^2])^2 \tag{4.9}$$

Figure 12 graphically shows the interpretation of a linear approximation for the maximum of generalized canonical forms that depend only on one nonlinear function parameter. The canonical forms for A and B are shown using thick dashed curves in the figure, and the exact maximum $C = max(A, B)$ is shown using a bold solid curve. The approximation of the maximum, $C_{app}$ , is represented by a solid thin curve: here, the curve of $C_{app}$ is closer to curve A, because, as can be observed in Figure 12, $max(A, B)$ is more often equal to A than to B; in other words, A has a higher probability of being the maximum.



$C=max(A,B)$
$C_{appr} = c_0 + f_C(\Delta X)_c$
$A = a_0 + f_A(\Delta X)$
$B = b_0 + f_B(\Delta X)$
$\Delta X$
Interval where A>B
Interval of $6\sigma$ variation

Figure 12: Approximation of the maximum of two generalized canonical forms A and B.

Finding the approximation for the maximum of two generalized canonical forms requires the computation of the tightness probability $T_A$, the mean $E[max(A,B)^2]$ and the second moment $(E[max(A,B)^2])^2$ of max$(A,B)$ that are defined as follows:

$$T_A = Prob(A > B)$$

$$=\int_{A>B} p(\Delta X_N, \Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_N d\Delta X_{LG} d\Delta X_a d\Delta X_b \tag{4.10}$$

$$E[max(A,B)] = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} max(A,B)\, p(\Delta X_N, \Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_N d\Delta X_{LG} d\Delta X_a d\Delta X_b \tag{4.11}$$

$$E[max(A,B))^2]$$
$$= \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} (max(A,B))^2\, p(\Delta X_N, \Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_N d\Delta X_{LG} d\Delta X_a d\Delta X_b \tag{4.12}$$

where $p(\Delta X_N, \Delta X_{LG}, \Delta X_a, \Delta X_b)$ is the joint probability density function of all process parameter variations.

If the vector of $\Delta X_N$ is empty, then the computations regress to the maximum of two first-order canonical forms, which can be computed analytically in a very efficient way. However, when there are non-Gaussian probability distributed or nonlinear function parameters, simple analytical formulas may not exist for the maximum of two generalized canonical forms. In the remainder of this section, we will focus mainly on the computation of tightness probability, the mean and the second moment for the *max* function.

***Computations of Tightness Probability, Mean and Second Moment***

The computations of tightness probability, mean and second moment for the *max* function involve the evaluations of the integrals in (4.10), (4.11) and (4.12) which can be very hard to compute analytically for arbitrary non-Gaussian process parameter PDFs and arbitrary nonlinear functions, $f_A$. The obvious way to solve this problem is to apply a numerical technique, but this results in losing the desired computational efficiency. In this section, we present a combined approach that processes Gaussian and linear function parameters analytically, and uses a numerical technique only for non-Gaussian or nonlinear function parameters. The method is efficient for realistic cases where most sources of variations can be captured accurately enough by Gaussian distributions and linear delay functions, and only a few of them demonstrate significant nonlinear behavior or non-Gaussian distribution. Therefore, as will be illustrated in the experimental results section, the proposed technique does not reduce the efficiency of dealing with Gaussian and linear function parameters, and can handle additionally up to 7 to 8 non-Gaussian and/or nonlinear function process parameters with reasonable run-times.

There are two equivalent ways of presenting the technique for computing the tightness probability, mean and the second moment. One is based on conditional probability and conditional moments, while the other uses transformation of the integrals defining the tightness probability, mean and the second moment. We begin with a presentation of the first approach.

The generalized canonical form in expression (4.2) can be reorganized by combining the non-Gaussian nonlinear term and the mean value $a_0$:

$$A = \left(a_0 + f_A(X_N)\right) + \sum_{i=1}^{n_{LG}} a_{LG,i}.\Delta X_{LG,i} + a_{n+1}.\Delta R_a \tag{4.13}$$

Then, for the fixed values of the non-Gaussian and nonlinear function parameters $\Delta X_N$, A can be regarded a first-order canonical form, $A_{Cond}$, with only Gaussian and linear function parameters and its mean value is $a_0 + f_A(X_N)$. Now, consider two generalized canonical forms A and B represented in the form of Equation (4.13).When all $\Delta X_N$ are at fixed values, the conditional tightness probability $T_{A,cond}$, conditional mean $c_{0,cond}$ and conditional second moments $m_{2,cond}$ of $\max(A, B)$ become functions of non-Gaussian and nonlinear function parameters $\Delta X_N$:

$$T_{A,cond}(\Delta X_N) = P(A > B|\Delta X_N)$$

$$c_{0,cond}(\Delta X_N) = E[max(A,B)|\Delta X_N]$$

$$m_{2,cond}(\Delta X_N) = E[(max(A,B))^2|\Delta X_N] \tag{4.14}$$

Here, we assume that non-Gaussian and nonlinear function parameters $\Delta X_N$ are independent of all of the Gaussian and linear function parameters $\Delta X_{LG}$. In fact, this is a rather valid assumption: correlated random variables tend to have similar distributions, and if a linear parameter is correlated with a nonlinear one, independence can be achieved by orthogonal transformation techniques, such as principal component analysis or independent component analysis. Therefore, the joint conditional probability density function of $\Delta X_{LG}$, under the condition of frozen values of $\Delta X_N$, is simply the joint probability density function of the $\Delta X_{LG}$:

$$\mathrm{p}(\Delta \mathrm{X}_{\mathrm{LG}}|\Delta \mathrm{X}_{\mathrm{N}}) = \mathrm{p}(\Delta \mathrm{X}_{\mathrm{LG}}) \tag{4.15}$$

Thus, we can use analytical Clark's formulas in [27] for computing the conditional tightness probability, mean and second moments for the maximum of two generalized canonical forms, under the condition that the values of all non-Gaussian and nonlinear function parameters are frozen; however, $a_0$ and $b_0$ should be substituted by $a_0 + f_A(X_N)$ and $b_0 + f_B(X_N)$. Since this method uses only analytical formulas, the required values can be computed efficiently. The actual values of tightness probability, mean, and second moment of $max(A, B)$ can be computed by integrating the conditional tightness probability, mean and second moment over the space of non-Gaussian and nonlinear function parameters with their joint probability density function:

$$T_A = \int_{-\infty}^{\infty} T_{A,cond}(\Delta X_N) p(\Delta X_N) d\Delta X_N \tag{4.16}$$

$$E[max(A, B)] = \int_{-\infty}^{\infty} c_{0,cond}(\Delta X_N) p(\Delta X_N) d\Delta X_N \tag{4.17}$$

$$E[(max(A, B))^2] = \int_{-\infty}^{\infty} m_{2,cond}(\Delta X_N) p(\Delta X_N) d\Delta X_N \tag{4.18}$$

The integrations in Equations (4.16), (4.17) and (4.18) can be evaluated numerically. In the simplest case, it is performed by integrating numerically in $m$ orthogonal discretized regions of non-Gaussian and nonlinear function parameters. We compute the conditional tightness probability, conditional mean and conditional second moment by formulas (4.14). Then the integrals of Equation (4.16), (4.17) and (4.18) can be computed approximately as sums of corresponding values over all the discretization grids. For example, the numerical formula for tightness probability is as follows:

$$T_A = \sum_{k=1}^{m} T_{A,cond,k}(\Delta X_N). p_k(\Delta X_N). V_k \tag{4.19}$$

where $T_{A,cond,k}(\Delta X_N)$ is the conditional tightness probability that A>B under the condition that non-Gaussian and nonlinear function parameters have fixed values inside the $k^{th}$ grid of integration; $p_k(\Delta X_N)$ is the value of the joint probability density function of the non-Gaussian and nonlinear function parameters in $k^{th}$ grid; $V_k$ is volume of the $k^{th}$ grid. The computational complexity of numerical integration, performed by discretizing the integration region, is exponential with respect to the number of nonlinear and non-Gaussian parameters. Our experiments show that for reasonable accuracy it is enough to have as little as 5 to 7 discrete points for each variable. This approach is applicable for cases with up to 7 to 8 nonlinear and non-Gaussian variables. For higher dimensions the integrals can be computed by a Monte Carlo integration technique.

To better understand the technique for computing the required values of tightness probability, mean and standard deviations of $max(A,B)$, we now provide an alternative explanation for an equivalent derivation by a transformation of the integrals. Let us start with the evaluation of tightness probability in Equation (4.10).

Given the condition that the $\Delta X_N$ variables are independent of the $\Delta X_{LG}$ variables, the joint probability density function of all sources of variations can be decomposed into:

$$p(\Delta X_N, \Delta X_{LG}, \Delta X_a, \Delta X_b) = p(\Delta X_N).p(\Delta X_{LG}, \Delta X_a, \Delta X_b) \quad (4.20)$$

$$T_A = \int_{A>B} p(\Delta X_{LG}).p(\Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_{LG} d\Delta X_a d\Delta X_b d\Delta X_N \quad (4.21)$$

For fixed values of $\Delta X_N$, the region A>B, where A and B are in generalized canonical forms, can be regarded as comparing two Gaussian random variables $A_G(\Delta X_N)$ and $B_G(\Delta X_N)$, where

$$A_G = \left(a_0 + f_A(\Delta X_N)\right) + \sum_{i=1}^{n_{LG}} a_{LG,i}\Delta X_{LG,i} + a_{n+1}\Delta R_a \tag{4.22}$$

$$B_G = \left(b_0 + f_B(\Delta X_N)\right) + \sum_{i=1}^{n_{LG}} b_{LG,i}\Delta X_{LG,i} + b_{n+1}\Delta R_b \tag{4.23}$$

If we set

$$Q_0(\Delta X_N) = \int_{A_G(\Delta X_N) > B_G(\Delta X_N)} p(\Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_{LG} d\Delta X_a d\Delta X_b \tag{4.24}$$

Then the tightness probability can be computed as:

$$T_A = \int_{A>B} p(X_N).\, p(\Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_{LG} d\Delta X_a d\Delta X_b dX_N$$

$$= \int_{-\infty}^{\infty} p(X_N) Q_0(X_N) dX_N \tag{4.25}$$

Note that $Q_0(X_N)$ for fixed values of $\Delta X_N$ is in fact the tightness probability of $A_G(X_N)$ in $max(\mathrm{A_G(X_N)}, \mathrm{B_G(X_N)})$, where $A_G(X_N)$ and $B_G(X_N)$ are both Gaussians for fixed $\Delta X_N$. Since there is an analytical formula [27] for the tightness probability for Gaussian random variables, for fixed values of $(X_N)$, $Q_0(X_N)$ can be computed efficiently. The tightness probability $T_A$ in (4.25) can then be obtained by numerical integration over the space of non-Gaussian and/or nonlinear process parameters $X_N$.

Similarly, using the independence between $\Delta X_N$ and $\Delta X_{LG}$, the mean and second moment of $max(A, B)$ can be computed as:

$$E[max(A, B)] =$$

$$\int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} max(A, B)\,.\, p(\Delta X_N).\, p(\Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_{LG} d\Delta X_a d\Delta X_b dX_N$$

$$= \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} p(\Delta X_N)\, Q_1(\Delta X_N) d\Delta X_N \quad (4.26)$$

$E[max(A,B)^2] =$

$$\int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} max(A,B)^2 \,.\, p(\Delta X_N).\, p(\Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_{LG} d\Delta X_a d\Delta X_b dX_N$$

$$= \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} p(\Delta X_N)\, Q_1(\Delta X_N) d\Delta X_N \quad (4.27)$$

$Q_1(\Delta X_N)$

$$= \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} max\big(A_G(\Delta X_N), B_G(\Delta X_N)\big) \,.\, p(\Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_{LG} d\Delta X_a d\Delta X_b$$

(4.28)

$Q_2(\Delta X_N)$

$$= \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} (max\big(A_G(\Delta X_N), B_G(\Delta X_N)\big))^2 .\, p(\Delta X_{LG}, \Delta X_a, \Delta X_b) d\Delta X_{LG} d\Delta X_a d\Delta X_b$$

(4.29)

For fixed values of $\Delta X_N$ , $Q_1(\Delta X_N)$ and $Q_1(\Delta X_N)$ are the mean and second moment, respectively, for the maximum of two Gaussian random variables and these can be found using analytical formulas. The mean and second moment of $max(A,B)$ can then be computed by numerical integration over the space of non-Gaussian and/or nonlinear process parameter $X_N$.

## *4.3 Implementation and Results*

The proposed approach was implemented on top of EinsStat [44], an industrial statistical timing analysis tool. In the implementation, a process variation can have a non-Gaussian distribution and the delay dependence on a process parameter can be a nonlinear function. These are both specified by tables using an appropriately chosen discretization. The integrals for the mean, second moment and tightness probability are computed by numerical integration.

We first tested our implementation on computing $max(A, B)$ of two first-order canonical forms A and B with non-Gaussian parameters:

$$A = 10 + 0.5. \Delta X_1 + \Delta X_2 + 0.5. \Delta R_a \tag{4.30}$$

$$B = 10 + \Delta X_1 + 0.5. \Delta X_2 + 0.5. \Delta R_b \tag{4.31}$$

where $\Delta X_1$ and $\Delta X_2$ are random variables with lognormal probability distributions, and $\Delta R_a$ and $\Delta R_b$ are Gaussian random variables for the locally independent randomness. Figure 13(a) shows the probability density function of $max(A, B)$ computed by the proposed technique, by the original parameterized SSTA technique for linear Gaussian process parameters (where non-Gaussian distributions are approximated with Gaussians having the same mean and standard deviation), and by Monte Carlo simulation. The PDF computed by the proposed technique matches the Monte Carlo results much closer than the PDF computed by the original technique. The proposed technique and Monte Carlo simulation both predict asymmetric PDFs with similar trends especially at the tails of PDFs. The PDF computed by the original technique has a symmetric shape and substantially underestimates the worst-case value.

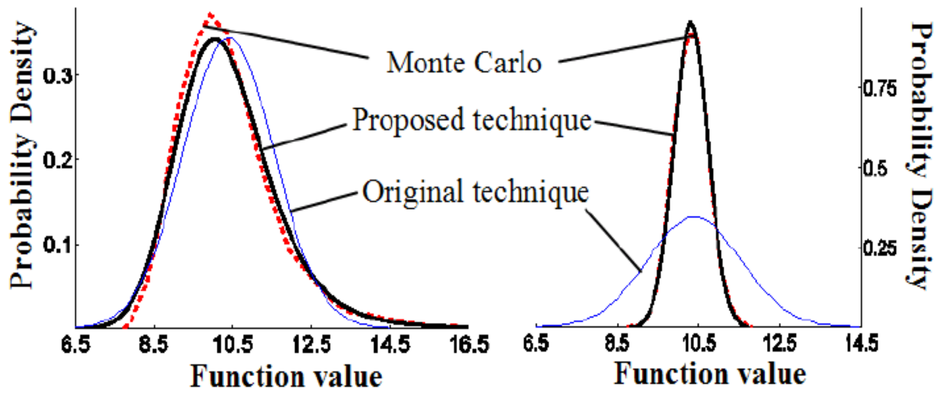
Next, we tested our technique on $max(A, B)$ with nonlinear (cubic) functions of Gaussian parameters:

$$A = 10 + \frac{(\Delta X_1)^3}{18} + \frac{(\Delta X_2)^3}{9} + 0.5. \Delta R_a \tag{4.32}$$

$$B = 10 + \frac{(\Delta X_1)^3}{9} + \frac{(\Delta X_2)^3}{18} + 0.5.\Delta R_b \tag{4.33}$$

Figure 13(b) compares the PDFs computed by the original technique, by the proposed technique and by Monte Carlo simulation. The original technique uses linear approximation of nonlinear functions that passes through the same -3σ and +3σ points. The proposed technique predicts virtually the same result as Monte Carlo, while the original technique significantly over-estimates the standard deviation.

To choose the number of discretization points that provides a good tradeoff between accuracy and run-time, we ran tests on a small industrial design A (3,042 gates and 17,579 timing arcs). Table 2 shows the CPU-time of our technique for different numbers of non-Gaussian parameters, for 5 and 10 discretization points.



(a) Test for non-Gaussian, (b) Test for non-Linear

Figure 13: Comparison of PDFs for maximum of two generalized canonical forms A and B. (a) shows the results on a non-Gaussian distribution (b) shows results on a nonlinear delay function.

The run time was measured on a single processor IBM RISC System 6000 model 43P-681. It is observed that processing three non-Gaussian parameters with 10 discretized points takes about 40 times longer than handling all three parameters as Gaussians, but for 5 discretization points, the run-time is only about 3 times longer. The PDF plots for design A are provided in Figure 14 for when 5, 10 and 20 discretized points are used. We observe that as the difference between PDF curves for 10 and 20 points is almost undistinguishable, the curve with 5 points also gives a result that is accurate enough. For nonlinear functions, we saw a similar dependence of run-time on the number of discretization points. Therefore, for our other experiments, we have used only 5 discretized points.

TABLE 2: COMPARISON OF THE RUN-TIME AS THE NUMBER OF NON-GAUSSIAN DISTRIBUTED SOURCES, AND THE NUMBER DISCRETIZATION POINTS, ARE VARIED

| Number of Non-Gaussians | | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| CPU-times (s) | 10 points | 69.17 | 7.53 | 2.14 | 1.38 |
| | 5 points | 3.82 | 1.54 | 1.40 | 1.38 |

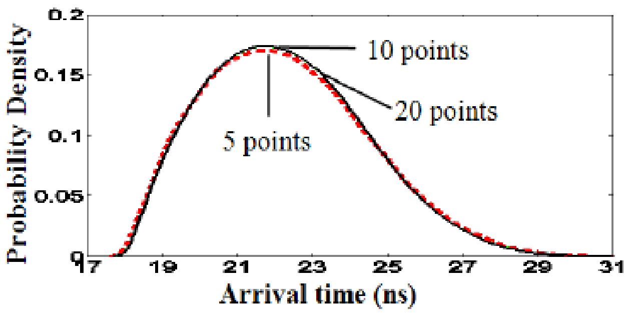


Figure 14: Comparison of accuracy versus run-time for Design A.

We performed statistical timing analysis of the same design A with linear delay functions of three lognormally distributed global sources of variations and a Gaussian uncorrelated local variation. The average values of delay sensitivities to each global and local variation were set to 2% and 6% of the corresponding nominal delay values, respectively. Figure 15 shows the probability density functions of the latest arrival time computed by three different techniques. The proposed technique gives a close match to the Monte Carlo result. In contrast, the PDF computed by the original SSTA technique for linear, Gaussian case deviates substantially from the Monte Carlo result. The PDF computed by Monte Carlo simulation is not Gaussian, but closer to lognormal because all three global sources of variation have lognormal distributions.

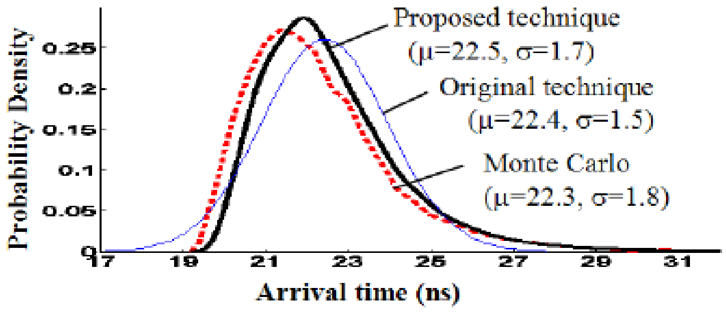
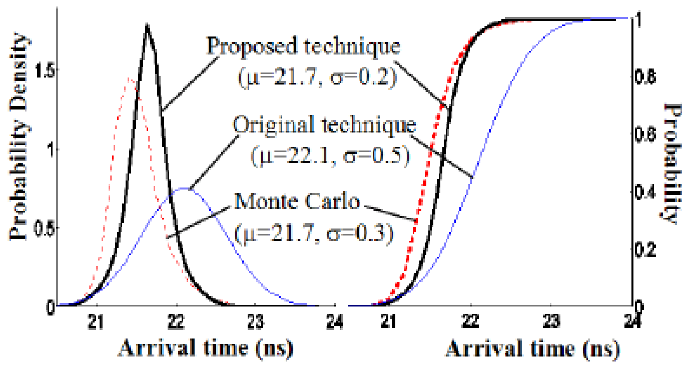


Figure 15: Comparison of PDFs of arrival time at a timing point for design A when different approaches are applied.

Unlike the proposed method, the original SSTA technique for the linear, Gaussian case approximates all delays with a Gaussian distribution, and therefore, it is hard for it to estimate the PDF well. The Monte Carlo predicts the 0.1% and 99.9% confidence points of path delays as 19.4 ns and 32 .0 ns, respectively. The proposed algorithm estimates similar values of 19.6 ns and 31 .5 ns, respectively, while the original technique computes these values as 17.8 ns and 27.0 ns, respectively.

In the second set of experiments, the three global sources of variation had Gaussian distributions but the delays of circuit gates and wires were cubic functions of these variations. The values of delay sensitivities to each global source of variation and uncorrelated local variation were set to 2% and 6% of the corresponding nominal delay values, respectively. Figure 16 shows PDFs and CDFs of the circuit delay computed by three different techniques. The proposed technique computes the same mean value as Monte Carlo, while the original technique overestimates it.



(a) PDF Curves (b) CDF Curves

Figure 16: Comparison of PDFs of arrival time at a timing point for design A when different approaches are applied. The delay functions at all circuit nodes are nonlinear (cubic) function of the variational sources in the experiments.

The original technique computes the 99.9% confidence point as 22.7 ns, as against 22.9 ns from Monte Carlo, while the original technique over-estimates it as 23.7 ns. Thus, we can conclude that when parameter variations have non-Gaussian distributions, or gate and wire delay depends on parameters nonlinearly, the proposed technique is essential to correctly predict circuit delay distribution and manufacturing yield.

Table 3 shows the run time of statistical timing analysis for five industrial designs when different numbers of non-Gaussian parameters are used in the analysis. In the set of tests, there are three global variational process parameters. In the case when the number of non-Gaussians is zero, the three global sources are set as Gaussian random variables, and in general, when the number of non-Gaussians is set to $k (0 \leq k \leq 3)$, the remaining $3 - k$ sources remain Gaussians. We see that, as the number of non-Gaussian parameters increases to 3, the run-time is only about 3 to 5 times longer compared to the case without any non-Gaussian parameters.

TABLE 3: COMPARISON OF RUN-TIME VERSUS THE NUMBERS OF NON-GAUSSIAN PROCESS PARAMETERS FOR VARIOUS SIZES OF INDUSTRIAL DESIGNS

| Ckt Name | Nuber of Gates | Timing Arcs | Number of Non-Gaussians | | | |
|---|---|---|---|---|---|---|
| | | | 3 | 2 | 1 | 0 |
| A | 3,042 | 17,579 | 3.8 s | 1.5 s | 1.4 s | 1.4 s |
| B | 11,937 | 57,151 | 12.3 s | 5.53 s | 4.3 s | 3.07 s |
| C | 53,317 | 292,097 | 79.1 s | 8 s35. | 27.3 s | 18.7 s |
| D | 70,216 | 363,537 | 93.3 s | 41.3 s | 30.5 s | 19.7 s |
| E | 1,085,034 | 5,799,545 | 2,083.1 s | 982.0 s | 788.5 s | 703.6 s |

The size of the designs for tests varies from 3,042 up to 1,085,034 gates. For the largest design E, the run-time is only about 35 minutes. In contrast, for the smallest design A, the run-time of Monte Carlo simulation is about 5 hours. However, due to the large size of designs, Monte Carlo simulations cannot be completed in a realistic amount of time, and thus the run-times are not provided in the table. Statistical timing analysis with nonlinear parameters has approximately the same run time.

## *4.4 Summary*

In this chapter, we have presented a novel and efficient technique for handling arbitrary non-Gaussian and nonlinear function parameters in parameterized block-based SSTA. Our approach is based on an extension of the first-order canonical form for representing delay and arrival time variations. Therefore this technique is fully compatible with the parameterized SSTA approach for Gaussian and linear function parameters presented in Chapter 3, and preserves its computational efficiency in processing such types of process parameter variations. The experimental results showed that the probability distributions of circuit delays computed by the new technique are closer to the results of Monte Carlo simulations than the original parameterized SSTA which approximates non-Gaussian distributions with Gaussians and nonlinear functions with linear functions, especially at the 99.9% confidence level. It should be also noted that in many cases non-Gaussian distributions of parameter variations can be approximated with Gaussians with reasonable accuracy, and only significantly asymmetric distributions requires handling as non-Gaussians. This conclusion is very important in practice because it justifies approximating most parameter distributions by Gaussians.

The limitation of the algorithm is that its run-time is exponential to the number of non-Gaussian and/or nonlinear function parameters. To further improve the efficiency, it is possible to develop techniques that can compute the max function analytically. In practice, as the number of non-Gaussian and/or nonlinear function parameters is not large, the algorithm is very efficient and provides a general framework for SSTA handling non-Gaussian parameters and nonlinear functions of delays. The method can be used to validate the approximation of process parameters as Gaussians and usage of linear delay functions, and then selectively apply crucial process parameters as non-Gaussian distributed or with nonlinear functions. The method is also important for sign-off timing analysis.

# CHAPTER 5

# GATE-LEVEL STATISTICAL STATIC TIMING ANALYSIS

## *5.1 Introduction*

The minimum feature size of CMOS technology continues to scale down, which enables higher density and lower chip cost. Additionally, the move to deep submicron process technologies has caused process variation to become a major issue that must be dealt with during the design of circuits. In older process technologies, the issue of process variation was mitigated through the use of a guardband. Designers would design their circuits under a tighter timing constraint and a smaller power budget than they were actually trying to meet. In the course of timing and power analysis, corner analysis would be used to determine the worst-case power and timing that the circuit would encounter with the target manufacturing process. Therefore, when the circuit was actually manufactured, it would still function according to the original specifications. The problem with this approach is that in deep submicron processes, the size of the guardband is prohibitively large if the circuit is constructed so that every chip meets the timing and power requirements. In place of guardbanding, and guaranteeing that every manufactured die functions, designers have moved toward meeting a performance yield requirement, where performance yield is defined as the percentage of manufactured die that will function within the timing constraints.

Process variation is the deviation of a parameter from its intended value. Parameters that are normally considered to be affected in deep submicron processes include gate length, device width, oxide thickness, and doping density. These parameters each have an effect on the resulting device characteristics. Now, the goal of Statistical

Static Timing Analysis (SSTA) is to perform a timing analysis of a circuit for the purpose of deciding whether or not the circuit meets the design requirements. The result from SSTA is a probability density function (pdf) of the circuit delay. This pdf can then be used to find the performance yield of a circuit, where the performance yield is defined to be the percentage of manufactured die that will function at a specific clock period.

High-level synthesis (HLS), also known as behavioral synthesis, is a synthesis technique that allows designers to move up the design chain to a higher level of abstraction. This means that instead of designing at the register transfer level (RTL), where a designer must specify all the timing of the circuit, the designer can work at a behavioral level, where only the data flow of the required circuit has to be specified. This frees the designer from the burden of many low-level details of circuit design, allowing for productivity increases of up to 10 times and code reductions of up to 100 times [45]. As manufacturing technologies continue to shrink, HLS is becoming a powerful technique to decrease the amount of time required to design a chip. In this chapter, we apply statistical timing analysis to high-level synthesis, and develop yield driven synthesis framework so that the impact of process variations is taken into account during high-level synthesis. The rest of chapter is organized as follows. Section 5.2 describes the background and previous work. Section 5.3 shows the proposed timing analysis model. Section 5.4 presents the experimental results. Finally, section 5.5 concludes this paper.

## *5.2 Background and Previous Work*

### *5.2.1 Timing in High Level Synthesis*

High-level synthesis (HLS) is the process of translating a behavioral description into a hardware implementation at register-transfer level. The design specification is usually written as a behavioral description, in languages such as System C. The behavioral description is first compiled into an internal representation (such as control and/or data flow graphs (CDFGs)), which are then mapped to the functional units that are selected from the resource library to meet design goals. The synthesis process usually consists of scheduling, module selection, and resource sharing [46].

High-level synthesis usually consists of several steps: scheduling, module selection, and resource sharing. Scheduling assigns each operation (such as add and multiply) in a CDFG to one or more clock cycles (or control steps). Scheduling techniques in HLS are usually classified as time-constrained scheduling or resource-constrained scheduling. Module selection decides the type of functional units to perform the operation in CDFG. Resource sharing uses the same resource (functional units or registers) to perform multiple operations or store more than one variable. These steps can interact with each other and affect the final synthesis results. In this paper, we focus on *data-flow intensive* applications (represented by a DFG), in which most of the computations performed in the design are arithmetic operations (such as addition and multiplication).

### *5.2.2 Statistical Delay Model*

The delay of each device by a linear combination of independent random variables leads to the creation of the canonical form.

$$d_a = \mu_a + \sum_i^n a_i z_i + a_{n+1} R \tag{5.1}$$

where $\mu_a$ is the mean delay, $z_i$ represents the $n$ independent RVs used to express the spatially correlated device-parameter variations, $R$ represents the residual independent variation, and coefficients $a_i$*'s* represent the sensitivity of delay to each of the RVs.

Because the canonical delay model of a functional module can be obtained from the result of the gate-level SSTA, the delay model of the high-level design can be also defined as (5.1). Thus, this model is possible to statistically operate among delay models in the same manner as an analysis at gate-level design.

It will be convenient to express both the sum and the maximum of such canonical forms in a canonical form. This will preserve the same approach throughout the computation of the delay for the whole circuit. Expressing the sum (*C*) of two canonical delays (*A* and *B*) is almost a straightforward task. The only unintuitive part is the coefficient of residual independent variation $c_{n+1}$. As the two coefficients, of which it is composed, correspond to independent (orthogonal) RVs, the new coefficient must be equal to the combined magnitude of the two.

$$C = A + B \tag{5.2}$$

$$\mu_c = \mu_a + \mu_b \tag{5.3}$$

$$c_i = a_i + b_i \text{ for } 1 \leq i \leq n \tag{5.4}$$

$$c_{n+1} = \sqrt{a_{n+1}^2 + b_{n+1}^2} \tag{5.5}$$

Computation of the maximum is a significantly more complex. As the maximum operation is nonlinear, but the canonical form is only an approximation of the maximum can be computed. The following is an algorithm proposed for solving this problem [27].

*1) Compute variances and covariance of A and B*

First of all the variance and covariance of the canonical forms *A* and *B* need to be calculated.

$$\sigma_a^2 = \sum_i^n a_i^2 \qquad \sigma_b^2 = \sum_i^n b_i^2 \qquad r = \sum_i^n a_i b_i \tag{5.6}$$

*2) Compute tightness probability TA = P (A > B) (the probability that arrival time A is larger than B) as presented in [47]*

$$T_A = \phi\left(\frac{\mu_a - \mu_b}{\theta}\right) \tag{5.7}$$

$$\phi(\acute{x}) \ = \int_{-\infty}^{\acute{x}} \phi(x)dx \tag{5.8}$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} exp^{-\frac{x^2}{2}} \tag{5.9}$$

$$\theta = \sqrt{\sigma_a^2 + \sigma_b^2 - 2r} \tag{5.10}$$

*3) Compute mean and variance of C = maximum (A,B)*

The new mean and variance of the new canonical form *C* = *max* (*A, B*) have to be expressed.

$$\mu_c = \mu_a T_A + \mu_b(1 - T_A) + \theta_\phi(\frac{\mu_a - \mu_b}{\theta}) \tag{5.11}$$

$$\sigma_c^2 = (\mu_a + \sigma_a^2)T_A + (\mu_a + \sigma_a^2)(1 - T_A) + (\mu_a + \mu_b)\theta_\phi(\frac{\mu_a - \mu_b}{\theta})\text{-}\mu_c^2 \tag{5.12}$$

*4) Compute sensitivity coefficient $c_i$ using the tightness probability*

Then the weighting coefficients for the maximum.

$$c_i = a_i T_A + b_i(1 - T_A) \text{ for } 1 \leq i \leq n \tag{5.13}$$

*5) Compute sensitivity coefficient $c_{n+1}$ of canonical form $c_{approx}$ to make the variance of $c_{approx}$ equal to the variance of C = maximum (A,B).*

It was shown in [48] that a valid coefficient $c_{n+1}$ always exists as the residue $(\sigma_c^2 - \sum_i^n c_i^2)$ is always greater than or equal to zero. Unfortunately, this approach only computes an estimate, which by no means guarantees conservative results. Another way of coping with the problem is the use of the following relation.

$$max(\sum_i^n a_i, \sum_i^n b_i) \leq \sum_i^n max(a_i, b_i) \tag{5.14}$$

Consider the very simple canonical form for two delays $d_a = \mu_a + a\Delta X$ and $d_b = \mu_b + b\Delta X,$ where $\mu_a$ and $\mu_b$ are the mean delays of $d_a$ and $d_b$ respectively, and $a$ and $b$ are their sensitivities to the common RV $\Delta X$. The maximum of $d_a$ and $d_b$ is the upper envelope of these two intersecting lines, which is a nonlinear function and cannot be expressed exactly by the canonical form. Hence, to represent this maximum, a linear function of $\Delta X$ must be constructed that approximates this nonlinear function [20]. This result guarantees that if the higher of the coefficients corresponding to a particular independent RV is selected, then the result will be conservative. Therefore, a bounding canonical $C_{bound}$ form of the delay can be constructed by selecting the higher mean and the largest coefficients.

$$\mu_c = max(\mu_a, \mu_b) \tag{5.15}$$

$$C_{bound_{i}=} max(a_i, b_i) \tag{5.16}$$

## *5.3 Proposed Timing Model*

### *5.3.1 Nonlinear and Nonnormal Approaches*

Statistical STA is a very complex solution. A parameter has the same probability distribution for all the delays, but different delays may depend on the same parameter differently, which means different nonlinear functions. In order to extend parameterized statistical STA to non-Gaussian and nonlinear parameters, we generalize the first-order Canonical form (Equation 5.1) to non-Gaussian and nonlinear parameters. Then we construct a statistical approximation for the maximum of two generalized canonical forms by applying the same ideas as in the linear Gaussian case. Because of the existence of non-normal distributions and nonlinear dependencies, special canonical forms have been developed to cope with these challenges [27]. All of these are handled by numerical computations and tightness probabilities. In order to include the effect of nonlinear dependencies additional term is included in the form.

$$d_a = \mu_a + \sum_i^n a_i z_i + \sum_{i=1}^n \sum_{j=1}^n b_{ij} z_i z_j + a_{n+1} R \tag{5.17}$$

Where $z_1$ to $z_n$ represent sources of normal variations, and $z_{n+1}$ to $z_{n+m}$ are RVs with non-normal variations.

For the non-normal distributions the same approach is used. The delay terms for both the normally distributed contributions and the non-normal ones.

$$d_a = \mu_a + \sum_i^n a_i z_i + \sum_j^m a_{n+j} z_{n+j} + a_{n+m+1} R \tag{5.18}$$

Equations 5.17 and 5.18 can be aggregated in the following common form.

$$d_a = \mu_a + \sum_i^n a_i z_i + f(z_{n+1}, \dots \dots .., z_{n+m}) + a_{n+1} \tag{5.19}$$

where $f$ represents the nonlinear function and is described as a table for computational purposes, and RVs $z_{n+1}$ to $z_{n+m}$ represent sources of normal variations with nonlinear dependences or non-normal variations.

### *5.3.2 Timing Graph Mapping*

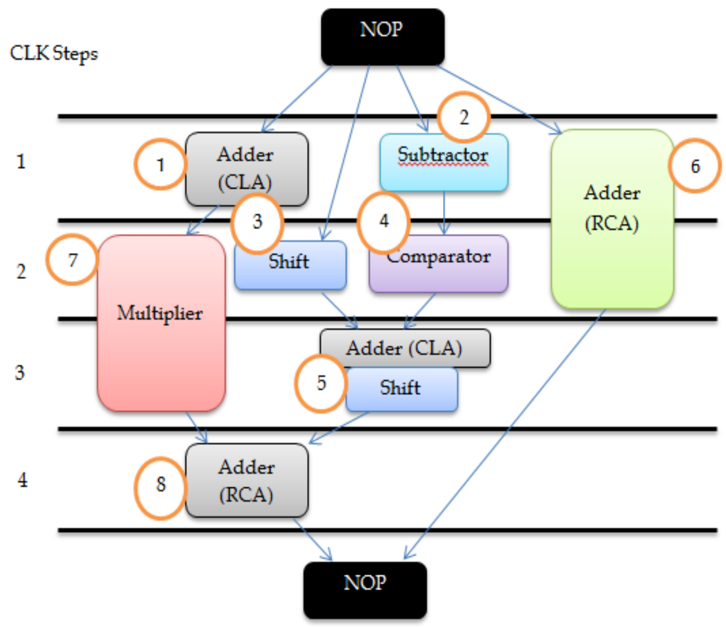
In high-level design, accurate timing operation means that each functional module should satisfy its governing timing constraints. To meet these timing requirements, accurate timing analysis and prediction are necessary. Thus, we propose a new method for timing analysis, in which the above-defined delay models of the functional modules are mapped to the timing graph. The data flow graph (DFG), which represents high-level design, is converted into a timing graph, as shown in the example (Figure 17). For the convenience of timing analysis, the inputs and outputs of modules are connected to virtual source and virtual sink nodes, respectively. That is, a virtual source node is a departure point of all circuit signals, and a virtual sink node is its final destination. Here, timing quantities expressed as equation 5.19 are propagated from the virtual source node to the virtual sink node. Additionally, the proposed method can map to the timing graph for various high-level operations as described in Figure 17.

### *5.3.3 Module of two (or more) operation cycles*

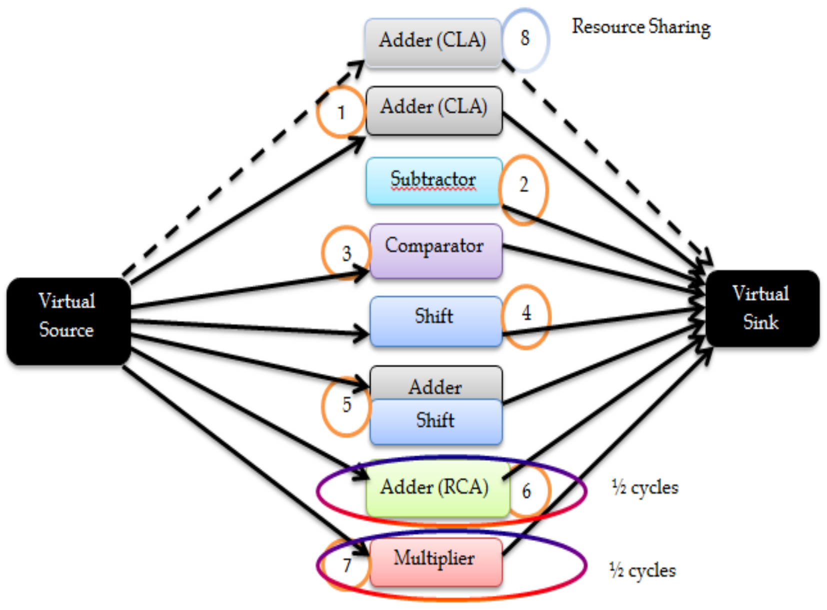
The module of more than two operation cycles is often used. In the mapping procedure, a delay of the module is normalized to one clock period. (i.e., the delay of more than two cycle operation is normalized to the one cycle operation delay.) For example, Adder (Ripple Carry Adder/Carry Look ahead Adder) and Multiplier modules use two operation cycles (Figure 17a). In the conversion procedure into a timing graph, they are used as the delay normalized to a clock period; 1/2 prefix means that half-value of delay time is applied to a timing analysis (Figure 17b).

### *5.3.4 Resource Chaining*

If a delay sum of modules satisfies the timing constraints, the circuit can use the chained resource. The resource chaining means to operate more than two functions in one cycle, e.g., Module 5 (Figure 17). In this case, the proposed method does timing analysis by using the statistical add operation.



(a)



(b)

Figure 17: Conversion of a data-flow graph (DFG) (a) into a timing graph (b) for an example circuit.

### *5.3.5 Resource Sharing*

Two (or more) operations may be bound to the same resource if they do not concurrently operate and they can be implemented by resources of the same type. Generally, the primary goal of resource sharing is to reduce the area of a circuit. In the proposed timing graph, the resource sharing can be considered. For example, module 1 and module 8 can be shared because they are same functions and are independent of timing (Figure 17). If a circuit is implemented by sharing these operations, resources for module 1 and module 8 are combined in the timing graph, and then the seven operations of the timing graph, module 1 to module 7, are just used for a timing analysis. On the other hand, if the circuit is implemented without the resource sharing, the timing graph has the eight resources as module 1 to module 8, and then the timing analysis is performed including module 8. If a module of the largest timing quantity satisfies one cycle operation, the whole circuit may operate on timing. Therefore, the latest arrival time among all fan-in edges of virtual sink should be calculated by the statistical operations.

### *5.3.6 Statistical Operations*

For an accurate statistical timing analysis, two major statistical operations between delay models are necessary. First one is the statistical add operation. Two or more modules can be chained in one clock-cycle, as module 5 (Figure 17). These chained modules are sequentially represented in the timing graph and total sum of delays of chained modules should be less than timing constraint, i.e., clock period. The add operation evaluates the distribution of sum of two distributions and each distribution can be expressed as equation 5.19. Then, the second operation is the statistical max operation.

The max operation is used for finding the latest arrival time among all fan-in edges of the virtual sink, which directly influence on the timing yield of a circuit. The max operation is defined as equation 5.16.

*5.4 Experimental Results*

We perform high-level statistical static timing analysis (SSTA) for arithmetic operations (Figure 17). The parameters which vary during SSTA are the gate length, the gate-oxide, and the doping concentration dependent threshold voltage variation. Table 4 shows the specifics of our process variation modeling. In this experiment, the predictive technology model (PTM) 45 nm technology was used to extract the necessary gate delay data [49]. We have assumed that the process parameters are non-normal distributed and that the ratio of die-to-die (D2D) and within-die (WID) variations of each process parameter is 1:1 [50]. Under these experimental environments, the value of mean (μ) and standard deviation (σ) for the delay distribution are obtained by block-based SSTA result for each module.

TABLE 4: PROCESS VARIATION PARAMETERS

| **Parameter** | **μ** | **3σ % Deviation from Mean** | **Correlation Distance (μm)** |
|---|---|---|---|
| $L_g$ | 45 nm | 15% | 1.0 |
| $W_g$ | 9 5nm | 12% | 1.0 |
| $N_a$ | $2 \times 10^{20}$ | 6% | 0.0 |
| $t_{ox}$ | 1.75 nm | 6% | 0.0005 |

The accuracy of the analysis has been assessed by comparing the performances of the proposed method and the Monte-Carlo (MC) simulation which is known to be the most accurate method. We also quantified the effectiveness of resource sharing, which has been observed to affect the timing yield. However, the effect of resource sharing for the timing yield decreased as TYC increased (Table 6).

TABLE 5: HIGH-LEVEL SSTA RESULTS OF EACH FUNCTIONAL MODULE AND TIMING YIELDS OF THE TOTAL SYSTEM

| **Operation** | **Timing Yield Constraints (TYC) [%]** | | |
|---|---|---|---|
| | ***85*** | ***95*** | ***99*** |
| Adder (CLA) | 96.23 | 99.20 | 99.89 |
| Subtractor | 98.61 | 99.73 | 99.97 |
| Comparator | 99.96 | 100 | 100 |
| Arithmetic Shifter | 100 | 100 | 100 |
| Adder2 (RCA)/2 | 85 | 95 | 99 |
| Multiplier/2 | 99.73 | 99.97 | 100 |
| Adder1 + Arithmetic Shifter | 85.565 | 95.62 | 99.13 |
| Adder1 (CLA) | 96.23 | 99.20 | 99.89 |
| ***Total system w/RS (1-8)*** | ***84.02*** | ***94.53*** | ***98.87*** |
| ***Total system w/o RS (1-8)*** | ***84.02*** | ***94.53*** | ***98.87*** |

TABLE 6: COMPARISONS RESULTS BETWEEN THE PROPOSED METHOD AND MC SIMULATION, VALUES IN PARENTHESIS ARE DIFFERENCES COMPARED TO MC SIMULATION

| **Timing Yield Constraints (TYC) [%]** | **Timing yield with resource sharing [%]** | | **Timing yield without resource sharing [%]** | |
|---|---|---|---|---|
| | Proposed | MC | Proposed | MC |
| 85 | 84.02 (1.67) | 85.425 | 84.02(1.6) | 85.37 |
| 95 | 94.53 (0.84) | 95.33 | 94.53 (0.87) | 95.36 |
| 99 | 98.87 (0.26) | 99.13 | 98.87(0.26) | 99.13 |

## *5.5 Summary*

We formulated the timing yield constraint high level synthesis problem for arithmetic operation modules. To solve this problem, we proposed a promising timing analysis method which considers process variations in high-level synthesis. In preliminary experiments, the proposed timing analysis method was comparably accurate when compared with the Monte-Carlo simulation. Specifically, our method showed very slight differences of 1.67% at 85% TYC and of 0.26% at 99% ($3\sigma$) TYC.

# CHAPTER 6

# ARCHITECTURAL-LEVEL STATISTICAL STATIC TIMING ANALYSIS

## *6.1 Introduction*

STA has been one of the most ubiquitous and popular analysis engines in the design of digital circuits for the last 30 years. However, in recent years, the increased loss of predictability in semiconductor devices has raised concern over the ability of STA to effectively model statistical variations. This has resulted in all-encompassing research [51], [7] in the so-called SSTA, which marks a significant departure from the traditional STA framework. The fundamental paleness of STA is that while global shifts in the process (referred to as die-to-die variations) can be approximated by creating multiple corner files, there is no statistically rigorous method for modeling variations across a die (referred to as within-die variations). However, with process scaling progressing well into the nanometer regime, process variations have become significantly more pronounced and within-die variations have become a non-negligible component of the total variation. It is shown that the incapability of STA to model within-die variation can result in either an over- or underestimate of the circuit delay, depending on the circuit topology [25]. Hence, STA's desirable property of being conservative may no longer hold for certain circuit topologies while, at the same time, STA may be overly pessimistic for other circuit topologies. This accuracy problem of STA can be even more pronounced in advanced processes. Consequently, the need for an effective modeling of process variations in timing analysis has led to extensive research in statistical STA.

SSTA algorithms can be broadly categorized into path-based and block-based. The path based SSTA seeks to estimate timing statistically on selected critical paths.

However, the task of selecting a subset of paths whose time constraints are statistically critical has a worst-case computation complexity that grows exponentially with respect to the circuit size. Hence the path based SSTA is not easily scalable to handle realistic circuits. On the other hand, the block based SSTA champions the notion of progressive computation. Specifically, by treating every gate/wire as a timing block, the SSTA is performed block by block in the forward direction in the circuit timing graph without looking back to the path history. As such, the computation complexity of block based SSTA would grow linearly with respect to the circuit size. However, to realize the full benefit of block based SSTA, we have to address a challenging issue that timing variables in a circuit could be correlated due to either global variations(20, 52, 53) or path reconvergence (54, 55). Global correlation refers to the statistical correlation among timing variables in the circuit due to global variations such as inter or intra-die spatial correlations, same gate type correlations, temperature or supply voltage fluctuations, etc. Path correlation, on the other hand, is caused by the phenomenon of path reconvergence, that is, timing variables in the circuit can share a common subset of gate/wire blocks along their path histories. Several solutions have been proposed to deal with either of these two types of correlations. In [20], [52], [53], the dependence on global variations is explicitly represented using a canonical timing model. However, these approaches did not take into account the path correlations. In [55], a method based on common block detection is introduced to deal with the path correlations. However, this method does not address the issue of dependence on global variations. To the best of our knowledge, there is no existing method that has dealt with both types of correlations simultaneously. We present a novel block based SSTA modeling in this chapter that is designed to consider both global correlations and path correlations:

- We develop a model encompassed with numerical computations and tightness probabilities to conditionally approximate the *MAX/MIN* operator by a linear mixing operator.
- We extend the commonly used canonical timing model to be able to represent all possible correlations, including the path correlations, between timing variables in the circuit.

The remainder of this chapter is organized as follows. SSTA problem formulation has been described in Section 6.2. Section 6.3 details the solution approaches. Section 6.4 details our architectural simulation and also present results from experiments which were conducted in order to benchmark our approach. We conclude in Section 6.5.

## *6.2 SSTA Problem Formulation*

In this section, we will formally define the problem to be solved.

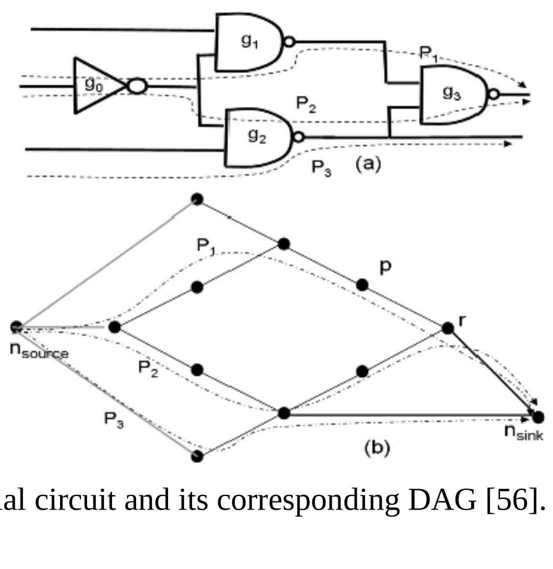


Figure 18: Combinatorial circuit and its corresponding DAG [56].

**Definition:**

A combinational circuit can be described using a Directed Acyclic Graph (DAG) $G$ given as $G = \{N, E, n_s, n_f\}$, where $N$ is the set of nodes corresponding to the input/output pins of the devices in the circuit, $E$ is the set of edges connecting these nodes, each with weight $d_i$, and $n_s$, $n_f$ are respectively source and sink of the graph. Figure 18(a) shows a digital circuit and its corresponding DAG is shown in Figure 18(b).

***Problem Formulation:***

*Let $p_i$ be a path of ordered edges from a source to a sink in G. Let $D_i = \sum d_{ij}$ be the path length of $p_i$. Then Dmax = max(D1, . . . ,Di, . . . ,Dn) is referred as the SSTA problem of the circuit.*

There are two main challenges in SSTA. The Topological Correlation which emerges from reconvergent paths, these are the ones which originate from a common node and then converge again at another node (reconvergent node). Such correlation complicates the maximum operation as it now has to be computed over correlated RVs. In a circuit example shown in Figure 19, one can see that the two red paths reconverge at the rightmost gate ($g_3$).

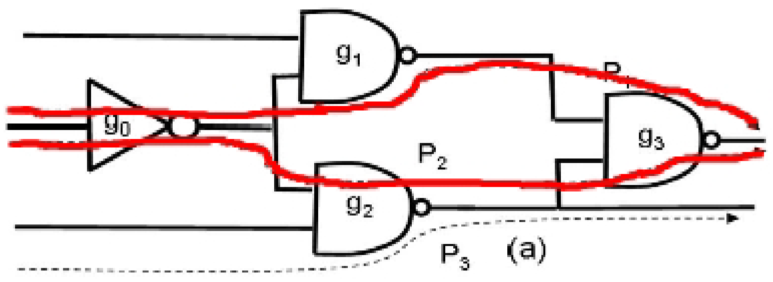


Figure 19: Topological Correlation [56].

The second challenge is the Spatial Correlation. It arises due to device proximity on the die and gives raise to the problems of modeling delay and arrival time so that correlations are included, as well as preserving and propagating these correlations. Figure 20 shows such two paths correlated by two closely placed gates ($g_1$ and $g_2$).

## *6.3 Solution Approaches*

The most general and brute force method of solving the above mentioned problem is to use numerical integration [54]. Although exact and applicable, this method is highly computationally expensive and thus, undesired. This leads to another approach, namely, the use of Monte Carlo methods [55]. The exact structure of these methods varies with the problem at hand. However, in general they all follow a common pattern: perform a statistical sampling of the sample space, perform deterministic computation for each sample, and aggregate the results into one final. In order to decrease the error, a lot of samples need to be taken, which, on the other hand, increases the computation effort. Therefore, probabilistic analysis methods are highly desired. Two such exist, one is the *Path-based* approach and the other is the *Block-based* approach.

The Path-based approach constructs a set of nodes that are likely to form the critical paths. The delay for each of these paths is then computed and a statistical maximum is performed over these results to yield the worst case delay.

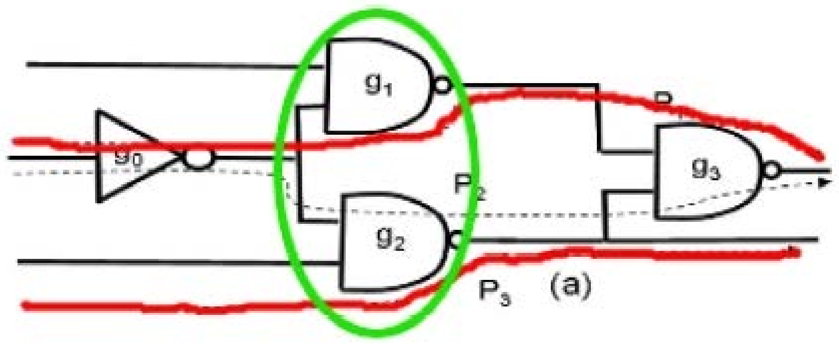


Figure 20: Spatial Correlation [56].

However, there are several problems associated with this approach. Sometimes it is hard to construct a set of likely critical paths. Therefore, the worst case scenario can be unintentionally omitted. This significantly increases the number of computations needed. Therefore, it is desired to use the Block-based approach. There instead of constructing critical paths the whole graph is traversed node by node. For all fan-in edges to a node the associated delay is added to the arrival time at the source node (the node upstream of the current one). The final arrival time at the node is computed using a maximum operation over the previous results. This approach has the advantage of propagating only two times, the rise and the fall time.

### *6.3.1 Distribution Propagation Approaches*

Analytical handling of distributions would be a good and computationally inexpensive approach. However, due to the nonlinearities and nonnormalities that are to occur in the dependencies and distributions used, it becomes a task close to impossible. There exist ways of handling this problem analytically, but assumptions are inevitable part of them. Therefore, another way is to discretize the distributions and normalize them so that the discreet values sum up to 1. In this way new set of probability mass functions is constructed, which closely approximates the real densities.

Now summation is an easy task to do. The result of such an operation is just a sum of shifted and scaled values of the delay. The shifts and the magnitude of the scaling is determined by the distribution of the arrival time.

$$z = x + y \tag{6.1}$$

$$f_z\,(t) = f_x(1)f_y(t-1) + f_x(2)f_y(t-2) + \ - - - + f_z(n)f_y(t-n) \tag{6.2}$$

$$f_z(t) = \textstyle\sum_{i=-\infty}^{\infty} f_x(i)f_y(t-i) = f_x(t) * f_y(t) \tag{6.3}$$

Where x, y are the delays of two devices which are connected in series. z is the resulting delay. $f_x, f_y, f_z$ are the delay functions of the device x, y and convolution of x &y function respectively.

Performing discrete time convolution is enough to compute the resulting delay from two devices in series. In order to compute the maximum delay between two paths (*x* and *y*) two cases have to be considered. Either one of the path *y* has a particular delay and path *x* has a delay less than or equal to the one of *x* or vice versa (equation 6.5). In order to obtain a density function this must be computed for all possible values of the delay *t*.

$$z = max(x, y) \tag{6.4}$$

$$f_z(t) = F_x(\tau < t)f_y(t) + F_y(\tau < t)f_x(t) \tag{6.5}$$

### *6.3.2 Propagation of Delay Dependences*

Expressing the delay of each device by a linear combination of independent random variables leads to the creation of the canonical form.

$$d_a = \mu_a + \sum_i^n a_i z_i + a_{n+1} R \tag{6.6}$$

where $\mu_a$ is the mean delay, $z_i$ represents the $n$ independent RVs used to express the spatially correlated device-parameter variations, $R$ represents the residual independent variation, and coefficients $a_i$*'s* represent the sensitivity of delay to each of the RVs.

It will be convenient to express both the sum and the maximum of such canonical forms in a canonical form. This will preserve the same approach throughout the computation of the delay for the whole circuit. Expressing the sum (*C*) of two canonical delays (*A* and *B*) is almost a straightforward task. The only unintuitive part is the coefficient of residual independent variation $c_{n+1}$. As the two coefficients, of which it is composed, correspond to independent (orthogonal) RVs, the new coefficient must be equal to the combined magnitude of the two.

$$C = A + B \tag{6.7}$$

$$\mu_c = \mu_a + \mu_b \tag{6.8}$$

$$c_i = a_i + b_i \text{ for } 1 \leq i \leq n \tag{6.9}$$

$$c_{n+1} = \sqrt{a_{n+1}^2 + b_{n+1}^2} \tag{6.10}$$

Computation of the maximum is a significantly more involved. As the maximum operation is nonlinear, but the canonical form is, only an approximation of the maximum can be computed. The following is an algorithm proposed for solving this problem [53].

*1) Compute variances and covariance of A and B*

First of all the variance and covariance of the canonical forms *A* and *B* need to be calculated.

$$\sigma_a^2 = \sum_i^n a_i^2 \qquad \sigma_b^2 = \sum_i^n b_i^2 \qquad r = \sum_i^n a_i b_i \tag{6.11}$$

*2) Compute tightness probability TA = P(A > B) (the probability that arrival time A is larger than B) as presented in [27]*

$$T_A = \phi\left(\frac{\mu_a - \mu_b}{\theta}\right) \tag{6.12}$$

$$\phi(\acute{x}) = \int_{-\infty}^{\acute{x}} \phi(x)dx \tag{6.13}$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} exp^{-\frac{x^2}{2}} \tag{6.14}$$

$$\theta = \sqrt{\sigma_a^2 + \sigma_b^2 - 2r} \tag{6.15}$$

*3) Compute mean and variance of C = maximum(A,B)*

The new mean and variance of the new canonical form *C* = *max*(*A*,*B*) have to be expressed.

$$\mu_c = \mu_a T_A + \mu_b(1 - T_A) + \theta_\phi\left(\frac{\mu_a - \mu_b}{\theta}\right) \tag{6.16}$$

$$\sigma_c^2 = (\mu_a + \sigma_a^2)T_A + (\mu_a + \sigma_a^2)(1 - T_A) + (\mu_a + \mu_b)\theta_\phi\left(\frac{\mu_a - \mu_b}{\theta}\right) - \mu_c^2 \tag{6.17}$$

*4) Compute sensitivity coefficient $c_i$ using the tightness probability*

Then the weighting coefficients for the maximum.

$$c_i = a_i T_A + b_i(1 - T_A) \text{ for } 1 \le i \le n \tag{6.18}$$

*5) Compute sensitivity coefficient $c_{n+1}$ of canonical form $c_{approx}$ to make the variance of $c_{approx}$ equal to the variance of C = maximum(A,B).*

It was shown in [47] that a valid coefficient $c_{n+1}$ always exists as the residue $(\sigma_c^2 - \sum_i^n c_i^2)$ is always greater than or equal to zero.

Unfortunately, this approach only computes an estimate, which by no means guarantees conservative results. Therefore, it is not suitable as it might underestimate the delay on some occasions. Another way of coping with the problem is the use of the following relation.

$$max(\sum_i^n a_i, \sum_i^n b_i) \leq \sum_i^n max(a_i, b_i) \tag{6.19}$$

Consider the very simple canonical form for two delays $d_a = \mu_a + a\Delta$ and $d_b = \mu_b + b\Delta X,$ where $\mu_a$ and $\mu_b$ are the mean delays of $d_a$ and $d_b$ respectively, and $a$ and $b$ are their sensitivities to the common RV $\Delta X$. In [48] an example of $d_a$ and $d_b$ is shown as a function of $\Delta X$. The maximum of $d_a$ and $d_b$ is the upper envelope of these two intersecting lines, which is a nonlinear function and cannot be expressed exactly by the canonical form. Hence, to represent this maximum, a linear function of $\Delta X$ must be constructed that approximates this nonlinear function.

Note that $c_{approx}$ will at times underestimate and at times overestimate the actual result. On the other hand, the method proposed in [20] constructs a bound $d_{c_{bound}} = \mu_{c_{bound}} + c_{bound}\Delta X,$ where $\mu_{c_{bound}} = max(\mu_a + \mu_b)$ and $c_{bound} = max(a, b).$ As can be seen, the error of $c_{approx}$will be smaller than that of $c_{bound}$, where as $c_{bound}$, will be guaranteed conservative.

This result guarantees that if the higher of the coefficients corresponding to a particular independent RV is selected, then the result will be conservative. Therefore, a bounding canonical $C_{bound}$ form of the delay can be constructed by selecting the higher mean and the largest coefficients.

$$\mu_c = max(\mu_a, \mu_b) \tag{6.20}$$

$$C_{bound_{i=}} max(a_i, b_i) \tag{6.21}$$

### 6.3.3 Nonlinear and Nonnormal Approaches

Because of the existence of nonnormal distributions and nonlinear dependencies, special canonical forms have been developed to cope with these challenges [27]. All of these are handled by numerical computations and tightness probabilities. In order to include the effect of nonlinear dependencies additional term is included in the form.

$$d_a = \mu_a + \sum_i^n a_i z_i + \sum_{i=1}^n \sum_{j=1}^n b_{ij} z_i z_j + a_{n+1} R \quad (6.22)$$

Where $z_1$ to $z_n$ represent sources of normal variations, and $z_{n+1}$ to $z_{n+m}$ are RVs with nonnormal variations.

For the nonnormal distributions the same approach is used. The delay terms for both the normally distributed contributions and the nonnormal ones.

$$d_a = \mu_a + \sum_i^n a_i z_i + \sum_j^m a_{n+j} z_{n+j} + a_{n+m+1} R \quad (6.23)$$

Equations 6.22 and 6.23 can be aggregated in the following common form.

$$d_a = \mu_a + \sum_i^n a_i z_i + f(z_{n+1}, \dots \dots .., z_{n+m}) + a_{n+1} R \quad (6.24)$$

Where $f$ represents the nonlinear function and is described as a table for computational purposes, and RVs $z_{n+1}$ to $z_{n+m}$ represent sources of normal variations with nonlinear dependences or nonnormal variations.

### 6.4 Architectural Simulations

The vector-thread (VT) architectural paradigm describes a class of architectures that unify the vector and multithreaded execution models. In other words, VT architectures compactly encode large amounts of structured parallelism in a form that lets simple microarchitectures attain high-performance at low power by avoiding complex control and datapath structures, and by reducing activity on large wires. Moreover, VT exploits fine-grained parallelism locality more effectively that traditional superscalar, VLIW, or multithreaded architectures. In this thesis, we feed our statistical model to the specified Scaled Vector Thread Architecture as well as Graphic Processing Unit (GPU) GeForce 8800 GTX architecture with some parameters shown in Table 7 which includes a MIPS-RISC control processor or Single instruction, multiple data (SIMD) processor, 32 Kbytes of cache, and a four-lane vector-thread unit or multiprocessor that can execute 16 operations per cycle and support up to 128 simultaneously active virtual processor threads.

TABLE 7: PROCESSOR PARAMETERS

| **Clock Rate** | **Vector Thread Units/ # of Multiprocessors** | **# of Clusters/# of Processors** | **# of Registers (per cluster)/# of Registers (per processor)** |
|---|---|---|---|
| 400 MHz | 4 | 16 | 8192 |

Our SSTA delay model has been implemented in C/C++ and tested by benchmark circuits. It is noted that before testing all benchmark circuits are re-mapped into a library which has gates of *not, nand2,nand3, nor2, nor3 and xor/xnor*. Table 8 summarizes the performance comparison and runtime estimations. We ran 60 large IWLS, ITC and ISCAS benchmark designs to compute the per-circuit speed of our tightness probability based SSTA engine implemented on vector thread architecture. This tightness probability based analysis was performed with 4 vector thread units. Columns 1 list the name of the circuit. Columns 2, 3 and 4 list the number of primary inputs, primary outputs and gates in the circuit. Columns 5 and 7 list the GPU and CPU runtime, respectively. The time taken to transfer data between the CPU and GPU was accounted for in the GPU runtimes listed. In particular, the data transferred from the CPU to the GPU is the arrival times at each primary input, and the $\mu$ and σ information for all pin-to-output delays of all gates.

Column 8 reports the speedup obtained by using a single GPU card. Our results indicate that our approach can obtain an average speed up of about 282 times as compared to a serial CPU implementation and is faster than GeForce 8800 GTX.

TABLE 8: SSTA RESULTS USING TIGHTNESS PROBABILITY

| **Circuit** | **# Inputs** | **#Outputs** | **#Gates** | **Single GPU runtimes (s)** | **Scaled VT Processor runtimes (s)** | **CPU runtimes** | **Speedup For Single GPU** | **Speedup For Scaled VT** |
|---|---|---|---|---|---|---|---|---|
| b14 | 276 | 299 | 9496 | 4.734 | 4.201 | 1303.63 | 275.394x | 310.314x |
| b15_1 | 483 | 518 | 13781 | 6.952 | 6.521 | 1891.884 | 272.116 | 290.121x |
| b17 | 1450 | 1511 | 41174 | 20.736 | 19.311 | 5652.45 | 272.589x | 292.706x |
| b18 | 3305 | 3293 | 6599 | 6.326 | 5.977 | 905.924 | 143.197 | 151.568x |
| b21 | 521 | 512 | 20977 | 10.311 | 10.101 | 2879.765 | 279.298x | 285.097x |
| b22_1 | 734 | 725 | 25253 | 12.519 | 12.210 | 3466.783 | 276.913x | 283.929x |
| s832 | 23 | 24 | 587 | 0.298 | 0.248 | 80.585 | 270.376x | 324.939x |
| s8381 | 66 | 33 | 562 | 0.295 | 0.278 | 77.153 | 261.341x | 277.528x |
| s1238 | 32 | 32 | 857 | 0.432 | 0.419 | 117.651 | 272.248x | 280.789x |
| s1196 | 32 | 32 | 762 | 0.388 | 0.359 | 104.609 | 269.796x | 291.389x |
| s1423 | 91 | 79 | 949 | 0.521 | 0.497 | 130.281 | 249.858x | 262.134x |
| s1494 | 14 | 25 | 1033 | 0.508 | 0.489 | 141.812 | 279.414x | 290.004x |
| s1488 | 14 | 25 | 1016 | 0.5 | 0.481 | 139.479 | 279.187x | 289.977x |
| s5378 | 199 | 213 | 2033 | 1.16 | 0.979 | 279.094 | 240.58x | 285.080x |
| s92341 | 247 | 250 | 3642 | 1.949 | 1.766 | 499.981 | 256.57x | 283.114x |
| s13207 | 700 | 790 | 5849 | 3.512 | 3.271 | 802.963 | 228.633x | 245.479x |
| s15850 | 611 | 684 | 6421 | 3.675 | 3.347 | 881.488 | 239.855x | 263.366x |
| s35932 | 1763 | 2048 | 19898 | 11.318 | 11.008 | 2731.638 | 241.349x | 248.150x |
| s38584 | 1464 | 1730 | 21051 | 11.544 | 11.104 | 2889.924 | 250.335x | 260.259x |
| s38417 | 1664 | 1742 | 18451 | 10.341 | 9.97 | 2532.991 | 244.958x | 254.061x |
| C1355 | 41 | 32 | 715 | 0.366 | 0.309 | 98.157 | 268.363x | 317.660x |
| C1908 | 33 | 25 | 902 | 0.446 | 0.393 | 123.828 | 277.46x | 315.083x |
| C2670 | 233 | 140 | 1411 | 0.797 | 0.689 | 193.705 | 242.906x | 281.139x |
| C3540 | 50 | 22 | 1755 | 0.842 | 0.803 | 240.93 | 286.1x | 300.037x |
| C432 | 36 | 7 | 317 | 0.155 | 0.139 | 43.518 | 280.605x | 313.079x |
| C499 | 41 | 32 | 675 | 0.347 | 0.317 | 92.665 | 267x | 292.318x |
| C5315 | 178 | 123 | 2867 | 1.461 | 1.379 | 393.588 | 269.323x | 285.415x |
| C6288 | 32 | 32 | 2494 | 1.197 | 1.139 | 342.381 | 285.927x | 300.597x |
| C7552 | 207 | 108 | 3835 | 1.899 | 1.810 | 526.477 | 277.214x | 290.871x |
| C880 | 60 | 26 | 486 | 0.253 | 0.224 | 66.719 | 263.923x | 297.852x |
| Avg | | | | | | | 258.994x | 282.1352x |

## *6.5 Summary*

We have presented the implementation of tightness probability based SSTA on Vector Thread Architecture as well as a GPU GeForce 8800 GTX architecture. Tightness probability based SSTA is computationally expensive, but crucial in design timing closure since it enables an accurate analysis of the delay variations. Our implementation computes multiple timing analysis evaluations for a single gate in parallel. Threads which execute in parallel do not have data or control dependencies on each other. All threads execute identical instructions, but on different data. Our results indicate that our approach can provide 282 times speedup when compared to a conventional CPU implementation.

# CHAPTER 7

# DESIGN METHODOLOGY

## *7.1 Introduction*

Technology scaling has brought the rapid increase in process variability [1]. Its effects on device performance have compelled the industry to transition to statistical techniques for timing sign-off. Traditional corner case analysis (CCA) [1, 56] constrains the design and often sets stringent, unrealistic timing specifications. Moreover, for technology nodes smaller than 65 *nm*, these overestimated timing bounds compensate the performance improvement due to device scaling. SSTA [56] is used in practice to analyze the impact of process variations on timing. It handles the random parts of the process variations as probability distributions to calculate the delay statistically. SSTA has gained widespread acceptance for standard cell based designs, as it removes a significant portion of pessimism introduced by conventional approaches like CCA while accounting for global (inter-chip) and local (intra-chip) process variations [57]. The application of both path based and block based SSTA have been shown to be advantageous [56] for cell based ASICs for which reusable timing models could be easily characterized. The method of SSTA for microprocessors is proposed in [57, 58] which is applicable only for standard cell based blocks. But, for example, cache blocks in microprocessors are not made of standard cells. More than 50% of a multi-core processor and more than 30% of each core are occupied by cache arrays and custom, transistor level blocks both of which are not standard-cell based. For custom macros, there are significantly more transistor level options to improve performance with less overhead than in case of gate level circuits. Moreover, such macros occur in portions of the processor which are extremely timing critical where variations could adversely affect the final performance.

The methodology proposed in [58] for generating statistical models for the large IP macros can be used in SSTA flows allowing fast analysis. While this method is shown to be accurate, it works only for macros with gates as basic units and cannot be easily adapted for transistor level macros. A method of variation aware transistor level timing analysis for macros is described in [59]. Statistical models are built for macros at a chip level of hierarchy. These approaches introduce some inaccuracy in predicting chip level performance degradation due to variations. To overcome these problems and to perform accurate variation analysis of transistor level macros, rigorous, but time consuming MC SPICE simulations of selected paths are currently used. The simulation run time is of the order of hours/path. It is impractical to perform such MC simulations on all paths in the macros and is therefore required to have a prior knowledge of the top paths that could potentially become critical. Hence it becomes necessary to have a fast statistical timing analysis flow for transistor level macros that can compute the delay distributions due to process variations of all paths in the macros with accuracy close to MC simulations. The proposed methodology finds a solution to this problem. It first groups the macro transistors into logic gates called xcells by applying special grouping technique which does not approximate any transistor or wire information. It is vital in preventing any accuracy loss. For all extracted xcells timing library considering both inter-chip and intra-chip process variations using a SPICE circuit simulator is built. The library is later used by an industrial-standard timing engine to perform block based SSTA of the macro.

## *7.2 Global and Local Process Variations*

Threshold-voltage ($V_{th}$), effective channel length ($L_{eff}$), oxide thickness ($T_{ox}$), mobility (µ), and dopant concentration (C) are the main variation parameters that significantly affect performance. Their variations result in designs with a wide spread of critical path delay distributions that may degrade the timing yield, i.e. decrease the fraction of manufactured chips that meet the timing constraints. For analysis purposes, parameter variations are usually classified into two categories: the inter-chip or global and the intra-chip or local variations. In case of globally varying parameters, their values are the same for all devices on the chip.

Variation parameters may depend on each other. For instance, an increase in $T_{ox}$ also increases $V_{th}$. Principal component analysis is used to convert the dependent variation parameters into independent principal components (PCs). In general, the delay of a path $D$ due to variation is given by [60]:

$$D = D_0 + \sum_{i=1}^{n} \sigma_{p_i} . Z(Y_i) + \sum_{i=1}^{n} \sum_{k=1}^{j} \sigma_{m_{ik}} . Z(L_{ik}) \qquad (7.1)$$

Where $D$ is the path delay; $D_0$ is the nominal delay (without variation); $\sigma_{p_i}$ is the standard deviation of the delay distribution due to the global random variable $Z(Y_i)$; $i$ varies from 1 to n number of principal components; $\sigma_{m_{ik}}$ is the standard deviation of the delay distribution due to the local random variable $Z(L_{ik})$; $k$ varies from 1 to j number of transistors.

In equation (7.1) the local delay component is dependent on the number of transistors. The fact that for global variations all transistors within a macro are completely correlated and for local variations they are completely uncorrelated (statistically independent) helps re-write equation (7.1) as follows [60]:

$$D = D_0 + \sum_{i=1}^{n} \sigma_{p_i} . Z(Y_i) + \sum_{i=1}^{n} \sigma_{m_i} . Z(L_i) \tag{7.2}$$

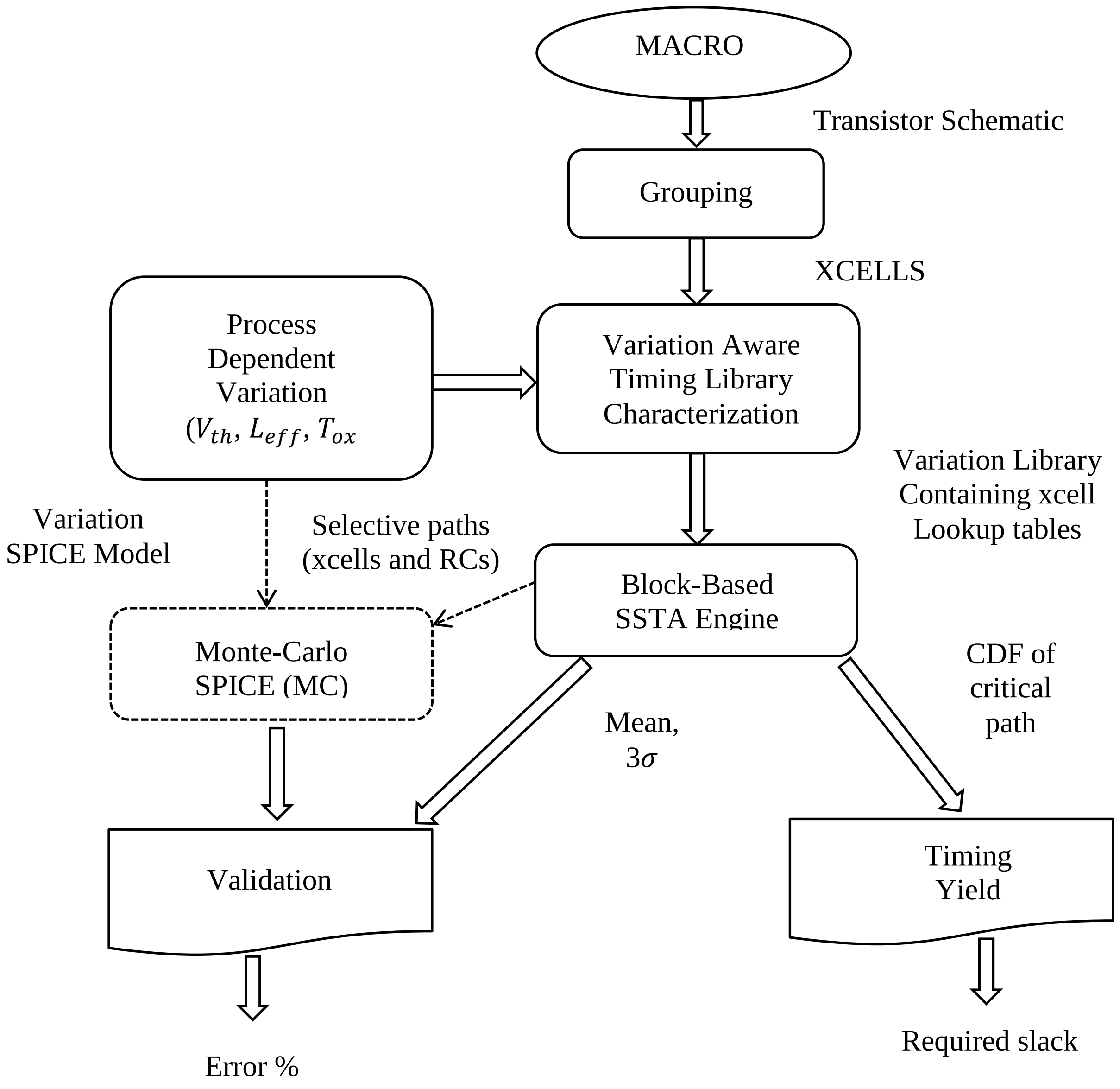
In equation (7.2) the number of local random variables $Z(L)$ is reduced from nj to just n showing that $Z(L)$ does not depend on the number of transistors in the macro. This is a useful result because in a macro, the number of transistors j could be in millions.

## *7.3 Our Approach*

The proposed SSTA flow developed for transistor macros is shown in Figure 7.1. It consists of two major steps.

1. Transistor level macro is converted to gate level blocks called xcells using the Xblock procedure [65].
2. Variation aware library is characterized for these xcells using the variation aware SPICE models.

SSTA engine determines delay distributions for all paths in the macro using the variation libraries. The validation step compares the SSTA results with MCS results. The timing yield step estimates the required arrival time based on the most critical path due to variation and reverse PCA step provides information on the variation sensitivities of each path that can be used for design optimization.



MACRO
Transistor Schematic
Grouping
XCELLS
Process Dependent Variation ($V_{th}$, $L_{eff}$, $T_{ox}$
Variation Aware Timing Library Characterization
Variation Library Containing xcell Lookup tables
Variation SPICE Model
Selective paths (xcells and RCs)
Block-Based SSTA Engine
Monte-Carlo SPICE (MC)
CDF of critical path
Mean, $3\sigma$
Validation
Timing Yield
Error %
Required slack

Figure 21: Proposed SSTA flow.

The method used by block-based SSTA engine in Figure 21 is described in [58]. It is based on simultaneous application of the usual static as well as statistical static timing analysis. At the first stage usual static timing analysis (STA) is applied and at the second stage - SSTA. The offered method of the analysis allows reaching acceptable analysis results from the practical point of view of accuracy at rather small expenses of machine runtime. SSTA engine determines delay distributions for all paths in the macro using the variation libraries considering equation 7.2. The validation step compares the SSTA results with MC results. The timing yield step estimates the required arrival time based on the most critical path due to variation.

## *7.4 Convert a Macro to the xcells*

The conversion of the macro's transistor level netlist into a netlist of xcells is performed by an internal tool Xblock [65]. Xblock was developed to facilitate hierarchical, transistor level static timing analysis using industrial block-based timing analyzers. It takes as input a transistor level GDSII layout of a macro and obtains a logic (verilog format) and parasitic netlists (spef format) as outputs that can be used by a static timing engine. The logic netlist consists of xcells each of which contains transistors that are source/drain connected to its output node.

The xcells are inferred by a rule-based recognition process that can recognize static CMOS, transmission gates, cross-coupled domino gates, latches, and flops. Using the inherent hierarchy in memory blocks like cache, specialized xcells are formed by grouping a number of SRAM bit cells (~5000 bit-cells per xcell) that are referred as bit-columns. The parasitic netlist contains the interconnect and device internal parasitics. The latter include the transistor parasitics that are pushed to the output node of each xcell. In order to reduce the number of inferred xcells that must be characterized, the xcells that have the same topology and whose internal parasitics are within a small range are folded to form a single xcell. Xblock also automatically generates control files for all the inferred xcells to drive the characterization engine for both setup and hold analysis. For certain special xcells (like bit-column) the control file is manually generated to handle complex constraints (like bitline pre-charge in a memory cell). An average xcell other than the bit-column typically consists of 10-15 transistors.

Xblock currently facilitates fast and accurate timing analysis for large industrial macros including memories through a block-based STA engine, providing visibility within the macro while performing chip level STA. Our proposed flow extends the usage of Xblock to generate xcells from transistor level macros that are suitable for SSTA library characterization.

## *7.5 Variation Aware Device Models*

In order to characterize variation libraries we first need SPICE device models that are variation aware. Transistor models corresponding to the typical (TT) corner case and the *3σ* variation ranges of different parameters are provided by the foundry. The variation parameters (like $V_{th}$, $L_{eff}$, $T_{ox}$, $\mu$) are dependent on each other. We perform Principal Component Analysis [9] [11] and convert these parameters into uncorrelated Principal Component (PCs). The foundry provides a correlation matrix $C_x$ that specifies the correlations between various interdependent input variables $X_m$ . Here, $X_1 = V_{th}$ , $X_2 = T_{ox}$, $X_3 = \mu$, …….. $X_m = L_{eff}$. A linear Eigen value decomposition produces a diagonal Eigen value matrix λ and Eigen vector matrix $P$ that satisfy the equation:

$$P \quad C_x \quad P^T = \lambda \tag{7.3}$$

$C_x$ is an $m \times m$, symmetric correlation matrix given by

$$C_x = \begin{bmatrix} 1 & \cdots & C_{x1,xm} \\ \vdots & \ddots & \vdots \\ C_{xm,x1} & \cdots & 1 \end{bmatrix} \tag{7.4}$$

The Eigen value matrix λ is an $n \times n$ symmetric matrix with all other than the diagonal elements equal to 0.

$$\lambda = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \tag{7.5}$$

Each column in the $m \times n$ Eigen vector matrix $P$ is a principal component vector $PC_j = \left[P_{1j}, P_{2j}, P_{3j}, \ldots . P_{mj}\right]^T$. Apart from the principal components being uncorrelated, PCA reduces [64] the number of dependent input variables $X_i$ ($i = 1$ to $m$) to a much smaller number of principal components $PC_j$, ($j = 1$ to $n$). This significantly reduces the number of times each xcell has to be simulated while creating the variation library. The linear relation between the correlated input variables $X_i$ and the uncorrelated principal components $PC_j$ is given by

$$X_i = \sum_{j=1}^{n} P_{ij}\ R(\lambda_j) \tag{7.6}$$

where, $R(\lambda_j) = N(\mu = 0, \sigma^2 = 1) * \lambda_j$

$N(\mu = 0, \sigma^2 = 1)$ represents a normal probability distribution with zero mean and variance = 1. The local variation parameters of the transistors within a macro can be spatially correlated. Our method can be modified to handle such a case by using a correlation matrix $C_{xx}$ instead of the $C_x$ and applying PCA on it. $C_{xx}$ characterizes the parameter correlations of transistors placed in one grid to the parameters of transistors in other grids of a macro [63, 58]. If there are $G$ grids and $m$ dependent input variables, $C_{xx}$ will be of a size $Gm \times Gm$ instead of $m \times m$.

Each variation parameter used in our device model file is a function of *5* PCs obtained by solving equation (7.3) for a correlation matrix of size 15 x 15. From our experiments, we find that using 5 PCs yields good results with reasonable runtime overhead. In our model we have a total of 10 PCs, 5 PCs for global and 5 PCs for local variations.

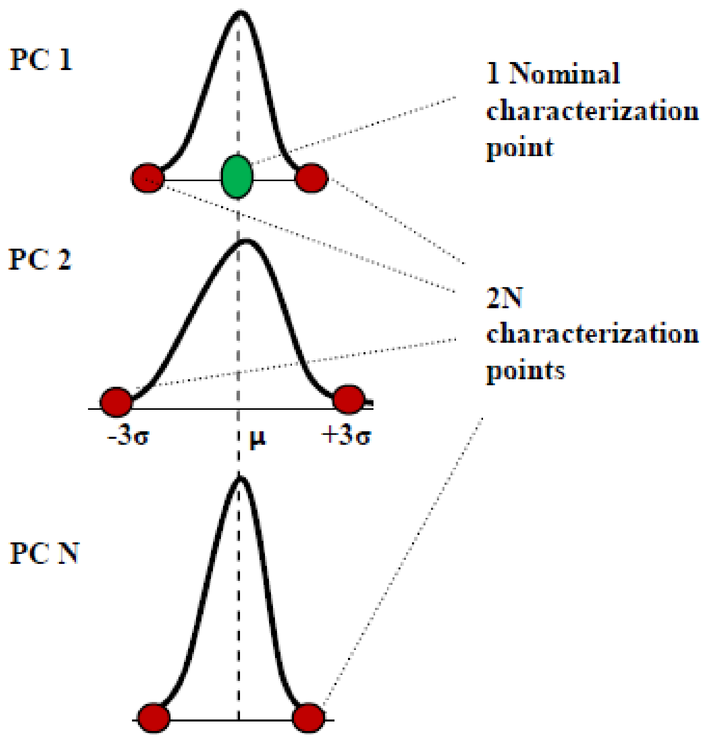


Figure 22: 2N values of PCs for which each xcell in the variation library is characterized.

## *7.6 Variation Library Characterization*

After converting a macro to gate level xcell netlist using Xblock, a timing library is generated. It contains delay/output slew look-up tables for each pin in the xcell and for all PCs. This is accomplished using an automated characterization engine that performs SPICE simulations to obtain delays for a wide range of input and output conditions (slew/load).

Each xcell in the library is characterized at 2☐+1 different values of the PCs stored as 2☐+1 look-up tables; ☐ is the number of PCs. Figure 22 illustrates this process. In our case with 10 PCs, each xcell has 21 tables in the library. One table corresponds to the nominal case, with all 10 PCs set to their mean (nominal) values. The other 20 tables are generated for xcells characterized at the *+3σ* and *-3σ* values for the 10 PCs. Using the delay values from the nominal, *+3σ,* and *-3σ* look-up tables for each PC in the library, the delay sensitivity of each path to different PC variation is computed by the statistical timer [66].

## *7.7 Results*

We used an industrial 45nm design macro for experiments. It contains 100 unique xcells and bit-columns. The total number of transistors in the macro is of the order of a few millions. The delay values shown in the figures and throughout the rest of the chapter are normalized to 1GHz for proprietary reasons, but that scaling in no way affects our message.

## *7.8 Monte-Carlo Vs. SSTA*

The macro studied in this chapter has the critical path (read access line through the SRAM bit-column) that requires at least 2 hours to complete MCS. This makes it impractical to perform MCS using variation device models for all top paths in the design. SSTA allows us to see these distributions and hence analyze the effects of variation on all paths of the design which is the most important goal achieved in this work. Figure 23 shows Cumulative Density Functions (CDF) of the top critical path slacks in the design.

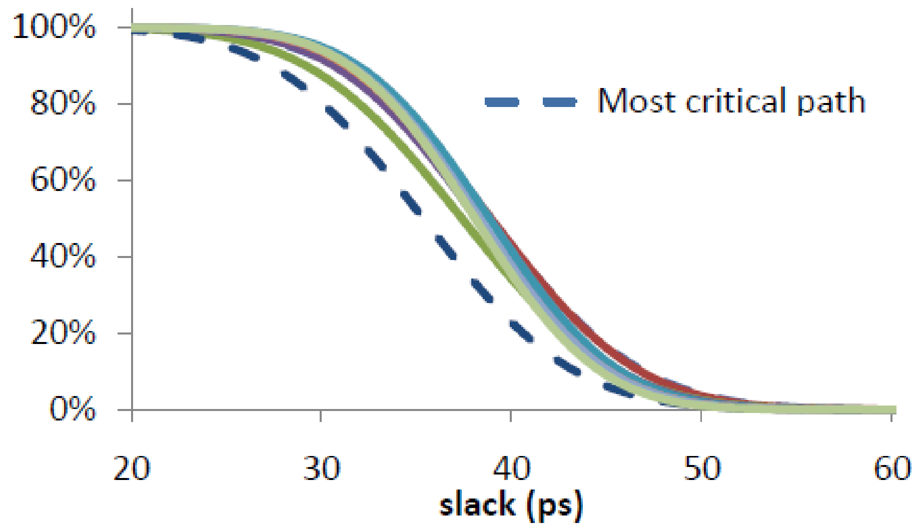


Figure 23: CDF of the slack values of some of the top critical paths.

In order to run MCS to validate SSTA, 100 paths of different lengths in terms of xcell number are pruned out from the macro netlist. A few representative paths are listed in Table 9. The extracted layout parasitics are also included during MCS simulations. Table 9 compares the mean and the standard deviation (*1σ*) of the endpoint delays (arrival time) between SSTA distributions and distributions obtained after 1000 runs of MCS simulations. Figure 24 compares the delay distributions of the most critical path in the macro obtained by MCS simulations and our SSTA flow.

TABLE 9: COMPARISON OF MCS AND SSTA PATH DELAYS

| | **Monte-Carlo SPICE (MCS) Simulation** | | | | | **SSTA** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Xcells/ path | Runtime | Mean (µ1) Delay (ps) | Local (L) Delay (ps) | Global (G) Delay (ps) | Total (T) Delay (ps) | Mean (µ1) Delay (ps) | Local (L) Delay (ps) | Global (G) Delay (ps) | Total (T) Delay (ps) | Error (%) |
| 10 | 2 hrs | 212 | 5.3 | 7.7 | 9.4 | 208 | 6.6 | 7.6 | 8.1 | 3.30% |
| 1 | 15 min | 70.7 | 1.5 | 2.2 | 2.6 | 77 | 0.2 | 0.1 | 0.3 | 5.10% |
| 2 | 30 min | 16 | 0.65 | 0.56 | 1 | 17 | 0.6 | 0.4 | 1.5 | 6% |
| 11 | 2.5 hrs | 298 | 5 | 11 | 12 | 302 | 4.5 | 9 | 10 | 0.60% |
| 1 | 20 min | 46 | 4.2 | 1.7 | 4.4 | 44.5 | 3 | 1.1 | 3.5 | 4% |

The maximum error percentage of the total variation (L+G = Local + Global) of SSTA reported delay is ~6%. The table also shows the runtime for MCS simulations. The runtime for the entire SSTA, which computes the distributions for all paths in the macro, is almost negligible, less than 3 minutes for a macro of ~*600,000* transistors.

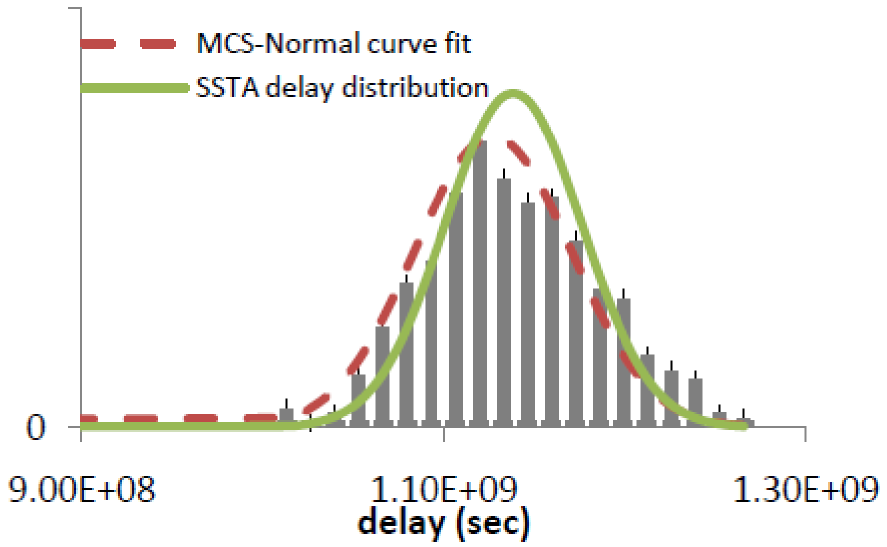


Figure 24: Comparison of the critical paths delay distributions obtained by MCS simulations and SSTA flow.

TABLE 10: DELAY SENSITIVIES WITH RESPECT TO THE DEPENDENT VARIABLES (ACTUAL VARIATION PARAMETERS) OBTAINED USING DELAY SENSITIVITES TO PCS

| Matrix | $V_{th}$ | $L_{eff}$ | $T_{ox}$ | $\mu$ |
|---|---|---|---|---|
| Xcell_1 | 4.0 ps | 1 ps | 0.5 ps | 0.5 ps |
| Xcell_1 | 4.5 ps | 1.2 ps | 2.0 ps | 0.5 ps |
| Xcell_1 | 5.5 ps | 1.8 ps | 2.3 ps | 0.6 ps |
| Xcell_1 | 6.0 ps | 1.4 ps | 2.1 ps | 0.7 ps |

In Table 9, we show the global and local components of the delays. For long paths (10 xcells), the global component dominates the local component due to the cancellation of device mismatches along the path. For short paths, the local variation is close to global variation and in some cases, the local component is dominant. Table 10 shows the sensitivities of the xcells in each path to the original variation parameters like $V_{th}$, $L_{eff}$, $T_{ox}$ and µ. Thus, our flow gives the designer a tool to identify variation sensitive areas in the design, even if they lie within a macro, and fix or optimize them if possible.

### *7.9 Statistical Vs Conventional Corner Case*

Figure 25 compares the delay results for the critical path in the macro obtained both statistically and using conventional (non-statistical) corner case analysis (CCA).

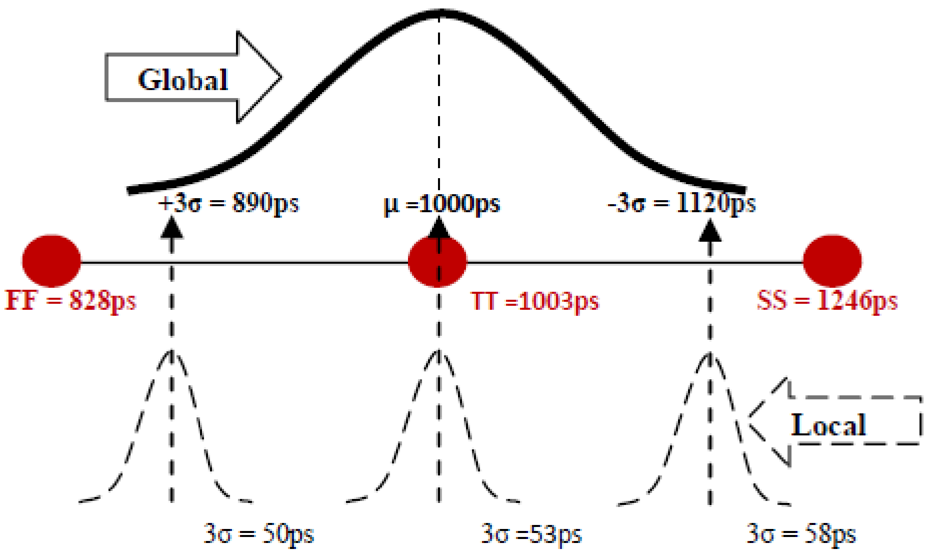


Figure 25: Comparison of statistical and non-statistical analysis: All the normal curves are delay distributions obtained after 1000 runs of MCS simulations. The circles on the median line are deterministic delays obtained using CCA.

The great normal curve above the line (marked *Global*) is the delay distribution obtained for 1000 MCS simulation runs only considering global variations and setting the local PCs to zero. The three small normal curves below the line (marked *Local*) are delay distributions obtained for 1000 MCS simulation runs by setting the global components to be equal to *+3σ,* mean, and -*3σ* and randomly varying only the local PCs. For each MCS run, each PC takes different values for each device in the path. Note that *+3σ,* mean and -*3σ* are the variation points for which the corner case SPICE models are usually designed (commonly referred to FF (Fast), TT (Typical) and SS (Slow). The three circles on the line are endpoint delay values of the same path obtained by using non-statistical, CCA SPICE models.

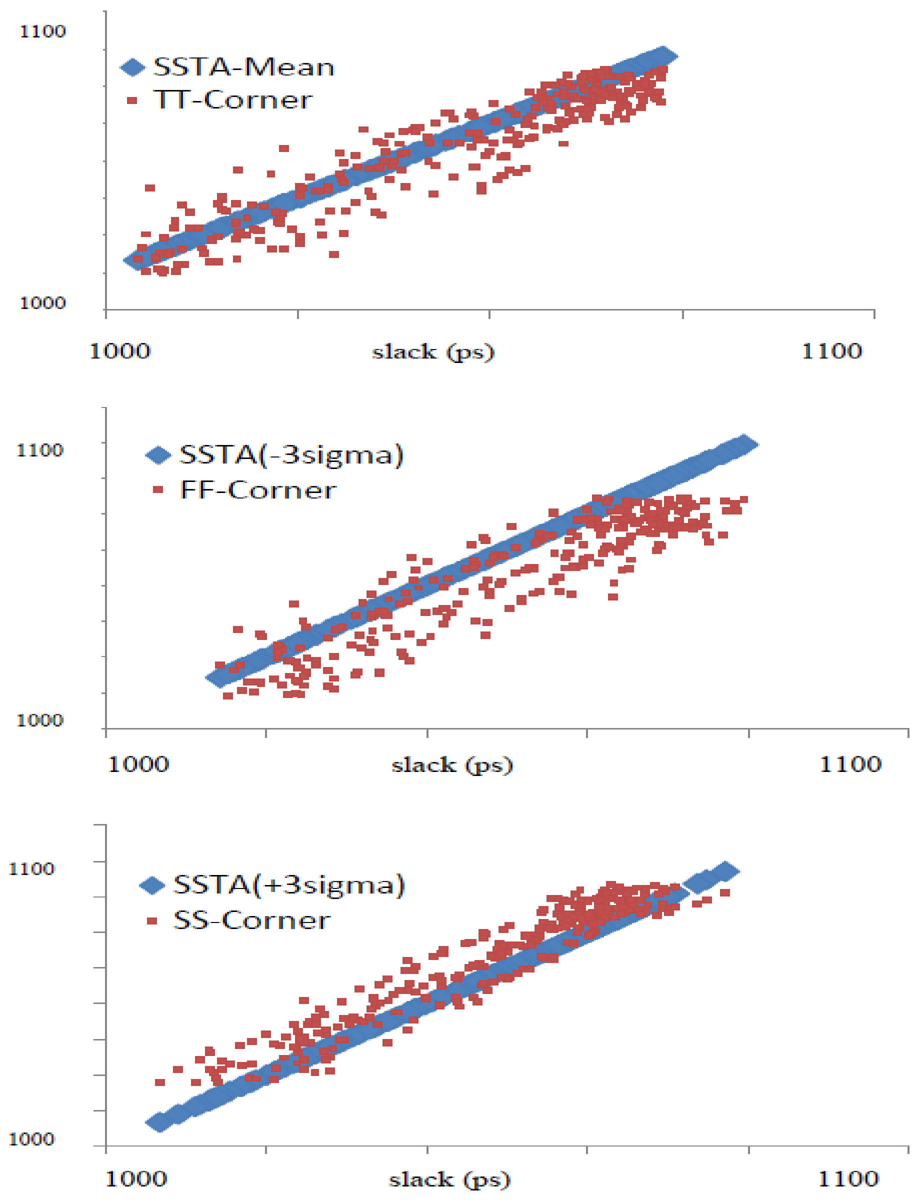


Figure 26: The median line - 45 degree (x = y). x and y-axis represent time delays. Points in the graph above the median line represent greater delays and points below the line represent smaller delays compared to the SSTA values.

It can be seen from Figure 25 that the corner case models are over margined. For instance, the SS corner delay is 1246ps while the 3σ (worst-case) point of the global distribution is only 1120ps. Corner models are typically constructed by reusing the parameters generated from one circuit to another [39]. In order to make sure that the models are valid for a wide range of circuits and also to account for the error % in calculating the truly worst case corner in the presence of several varying parameters, some margin is intentionally forced which makes the SS too pessimistic.

Hence, the worst case analysis performed using the slow model (SS), would give pessimistic results. Also note that the variance (3σ – mean) for local variation is much less compared to global (~60ps compared to ~120ps). It is expected to be much more significant for short paths where the cancellation effect of mismatched devices is less prominent.

With block-based SSTA, we could get the endpoint delays of all paths in the design with almost no overhead in run-time. For all three plots in Figure 26, the SSTA results correspond to -3$\sigma$, mean and +3$\sigma$ delay values obtained from the delay distributions of each path in the macro. Each point on the $x = y$ line represents the delay of a path obtained by running conventional STA individually for the corresponding deterministic corner cases. A point above the line indicates that the particular path has a statistical delay that is slower than that obtained by its equivalent corner model. The majority of path delays obtained by deterministic models are either too slow (compared to -3$\sigma$ SSTA) or too fast (compared to +3$\sigma$ SSTA). This again confirms that FF is too fast and SS is too slow not just for the critical path shown in Figure 25, but for almost all top paths in the design (Figure 26).

## *7.10 Difference in path sensitivity due to variation*

Using SSTA for this macro reveals paths that are not too critical at typical operating condition (mean) but become very critical at the extremes of variation (3$\sigma$). Without SSTA, designers would use the deterministic corner model to obtain delay/slack values which is close to the mean of the delay/slack distributions obtained using SSTA. This could sometimes be misleading as the designer is not aware of the real situation where new paths that are not the most critical could become critical when variations are considered. Figure 27 shows slack distributions obtained for the two top critical paths of the macro using our SSTA. Consider the two paths marked by pointers. It can be seen that the path1 has a smaller mean slack than path2 and is hence less critical from a designer's perspective who will only see these values using a deterministic approach. However the criticality of the two paths change with respect to -3$\sigma$ (worst case) suggesting that path1 is more sensitive to variation than path2.

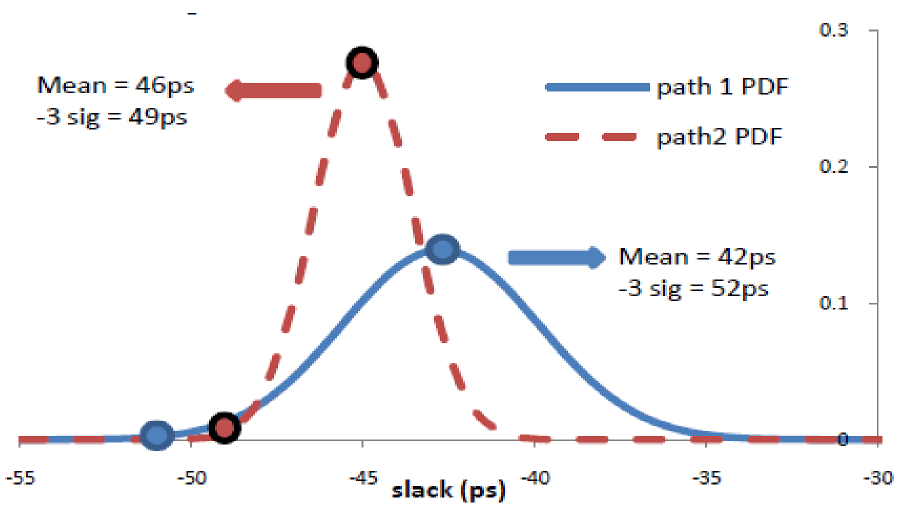


Figure 27: Slack values for paths in the macro that change criticality due to difference in variation sensitivities.

## *7.11 Timing Yield*

Without SSTA, designers would fix the critical paths to meet a frequency that is much greater than the target frequency needed for a particular yield. Figure 28 shows the CDF of the most critical path whose period defines the frequency of the entire macro. 50%

yield point corresponds to the nominal time period of 1000ps at which the SSTA was performed for this macro. For instance, if we need to achieve a 70% yield at 1000ps, SSTA results suggest a minimum required arrival time (RAT) of 1005ps to be set on the critical path based on the slack difference. This design has a large positive slack of 53ps even for a 99.8 % yield, suggesting that the design has been over-optimized. Figure 29 compares the slack values obtained for all paths by setting the minimum RAT from SSTA at 70% and 99% yield points and a conservative RAT used by designers to fix the design before using our SSTA flow. Figure 29 shows a clearly large margin that is pessimistic even to achieve a 99% yield.

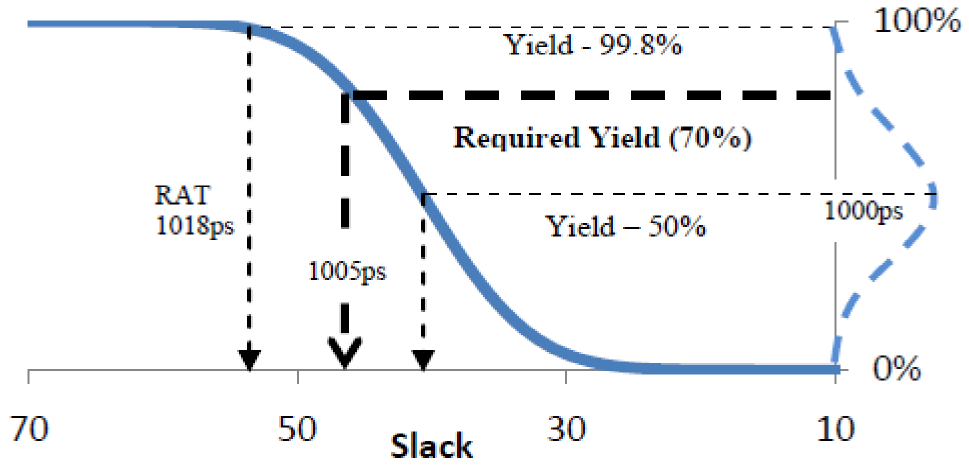


Figure 28: Timing yield plot – CDF of the most critical path of the design obtained using SSTA with RAT = 1000ps.

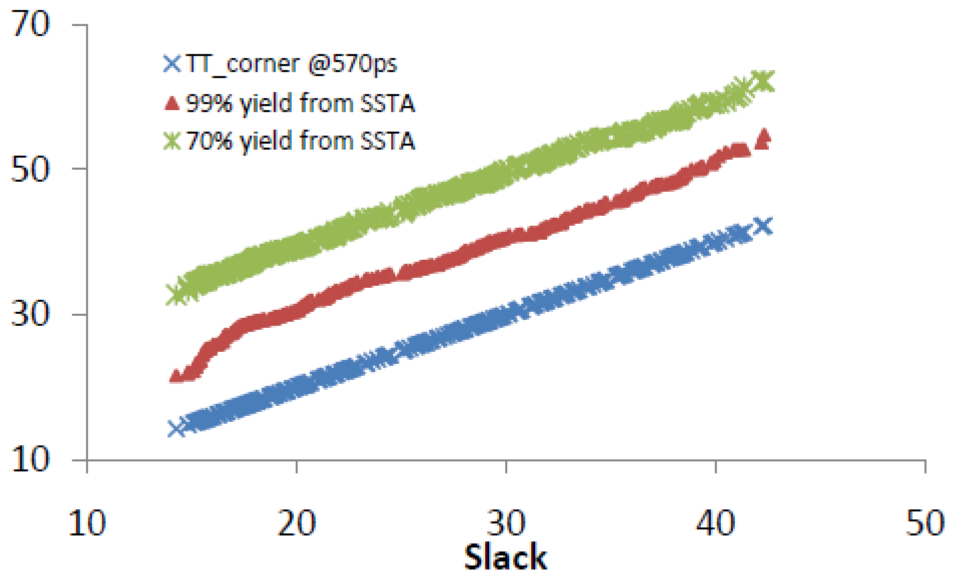


Figure 29: Slack values of all paths of the design obtained by performing SSTA with RAT chosen from 90%, 99% yield points and conventional corner case TT.

## *7.12 Characterization runtime*

Characterizing a variation library at 2N + 1 points as described in section 7.6 for each xcell even though is a one-time effort, is still time consuming. However, libraries generated this way for different PC corners are more accurate since the sensitivities are determined from look-up table delay values obtained by actual circuit simulation rather than analytical formulations. The library generation time linearly increases with the number of points at which each xcell in the library is characterized. For each point of characterization, a look-up table is generated for every xcell in the library. For a 5% compromise in accuracy, the library characterization time can be significantly reduced (Table 11).

TABLE 11: CHARACTERIZATION RUNTIME REDUCTION

| **Number of tables/xcell** | **Accuracy** | **Runtime** |
|---|---|---|
| Char 1:21 tables | 100% (normalized) | ~8hrs |
| Char 2:11 tables | 97% | ~4hrs |
| Char 3:5 tables | 95% | ~2hrs |

Char 1: We consider all 10 PCs.

Char 2: We only consider 5 global PCs and set a correlation of 1 between transistors to represent global variations. We use the same PCs and set a correlation of 0 between transistors to represent local variations.

Char 3: Assuming the delay variance obtained for each PC variation is symmetrical about the mean delay value, we characterize only N+1 tables instead of 2N+1 for the 5 global PCs. The xcells are characterized only at the mean and -3σ points of the PCs instead of the mean, -3σ and +3σ points shown in Figure 22.

## *7.13 Summary*

Macros are custom designed circuit blocks that are usually present in very critical sections of the microprocessor to maximize performance, power and/or yield. Transistor level macros have a very large optimization space that is difficult for designers to manually explore. As a result, custom, transistor macros derive maximum benefit from SSTA. In order to make correct design decisions especially at smaller technology nodes where the effect of variation on performance is large, macro designers currently rely on either non-statistical approaches like CCA which are pessimistic or on extensive circuit level simulations and several runs of MCS analysis, which is extremely time consuming. In this work, we show experimentally that CCA results are indeed pessimistic. While it's almost impossible to do MCS simulations on all paths in a macro or even on a few top critical paths, our SSTA flow provides distributions for all paths in the macro (including SRAM arrays) that are close to SPICE results (~95% accuracy). The flow also helps pin-point the paths and their components that are more sensitive to a particular source of process variation ($V_{th}$,$T_{ox}$, $\mu$, $L_{eff}$) which can be used for design optimization.

While this flow is developed mainly for transistor macros, it can easily be modified to be used for any cell based macro (without applying Xblock). The flow hence allows fast statistical timing analysis of an entire chip without abstracting transistor macros.

# CHAPTER 8

# CONCLUSION

## *8.1 Summary*

In current and future technologies, the increasing number and magnitude of process variations make the prediction of circuit performance an important but very challenging task. As the conventional corner-based technique becomes too pessimistic and slow, statistical circuit performance analysis techniques provide a good alternative.

In this thesis, we have focused on the problem of statistical static timing. The effects of spatial correlations in intra-die variations, which were ignored in most of the previous works, are also considered in our works. We show that spatial correlation is essential in order to correctly predict the probability distributions of circuit timing and leakage power. The statistical timing methods presented in the thesis are shown to be computationally efficient and accurate, and this is demonstrated through comparisons against Monte Carlo simulations. The timing estimation techniques are important, both for yield prediction in the post-layout stage, as well as for supporting circuit design and optimization in all stages of the design flow for shortening the design cycle and saving design costs.

Although in recent years, quite some work has been done in statistical circuit performance analysis for timing, but this area still requires further research. First, statistical performance analysis technique requires proper modeling of process variations including the decomposition and modeling of process variations including spatial correlations. Without an appropriate model, the prediction by statistical analysis could be a “garbage in and garbage out,” the result would not make much sense and cannot guide the circuit optimization in the correct direction. Second, the statistical timing analysis technique depends on correct characterization of gate/interconnect delay with respect to process parameter variations. A library that is characterized with worst-case and best-case corners must be recharacterized, such as characterizing with nominal value and sensitivities to process variations, in order to have accurate statistical timing analyzer. Third, although statistical performance analysis methods are more computationally efficient than corner-based methods and Monte Carlo approaches, they also show a tradeoff between accuracy and run-time. This may not be a problem if this is solely for the purpose of performance analysis. However, in order for the method to be integrated into a framework for circuit performance optimization, a good balance is required between the run-time and the accuracy. Finally, variation-aware circuit optimization techniques [68, 69, 72, 31, 74, 75, 77] that can take into account process variations are active fields for research and development. The technique should be applied across the overall flow of circuit design, including steps such as technology mapping [76], synthesis, buffer insertion [70], clock tree [73], physical design [1, 71], to overcome the limitations of traditional deterministic optimization techniques.

## *8.2 Future Work*

Statistical analysis is generally seen as the next EDA technology for timing and power sign-off. Research into this field has seen significant activity started about five years ago. This dissertation makes contributions to statistical modeling on non-Gaussian process parameter variations and nonlinear delay dependencies, High-level SSTA analysis and Architectural-level SSTA analysis and Design Methodology for ASIC flow. However, there are still lots of research works need to be done in this field.

For future development three major questions shall be stated here. Firstly, the interaction between timing and power especially leakage power is not properly addressed. This yields much optimization potential and should be used. Secondly, the variation aware implementation should be addressed. On one hand the process itself could be optimized but on the other hand the implementation could take the variations into account and thus reduce the overall impact of the variations on the circuit performance. Thirdly, the statistical analysis should be extended to higher levels of the design flow for FPGA, ASIC and NoC. A variation aware high-level synthesis can further optimize the statistical behavior of the final implementation. In a nutshell, the SSTA methods must be capable by proving the timing yield behavior for larger digital blocks as well as analog and mixed-signal circuits. The goal is the analysis of the whole integrated circuit and a sufficient estimation of the yield. With such tools the uprising impact of variation in future process generation can be addressed - but without them it will be impossible to develop complex systems in the future.

# APPENDIX

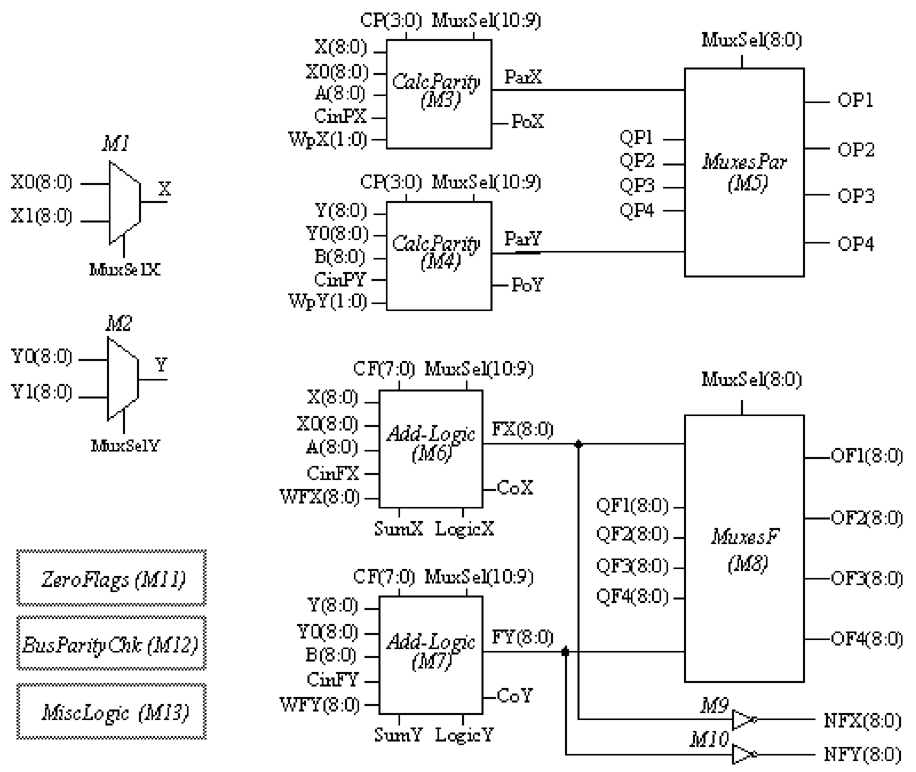
## CASE STUDY OF ISCAS c5315

Statistics: 178 inputs; 123 outputs; 2406 gates

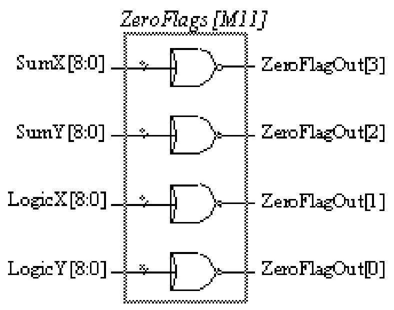
Function: 9-bit ALU

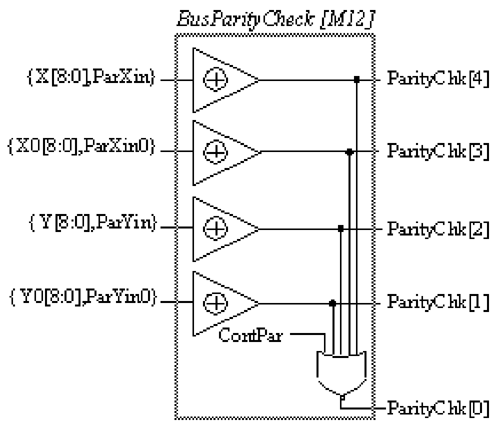
This benchmark is an ALU that performs arithmetic and logic operations simultaneously on two 9-bit input data words, and also computes the parity of the results. Modules M6 and M7 each compute an arithmetic or logic operation specified by the control input bus CF[7:0]. Module M5 consists of multiplexers that route the results of M6 and M7 and four input buses to its four outputs. Output buses OF1 and OF2 can also be set to logic 0 by MuxSel [8]. Modules M3 and M4 compute the parity of the result of the operation given by CP=CF [7:4]. Module M5 contains four multiplexers which direct the parity results and an additional set of four inputs to its outputs. The adders in M6 and M7 as well as the parity logic for the arithmetic operations in M3 and M4 use a carry-select scheme with 4-bit (low-order) and 5-bit (high-order) blocks. The circuit also includes logic for calculating various zero and parity flags of the input buses.

High-Level Model of c5315



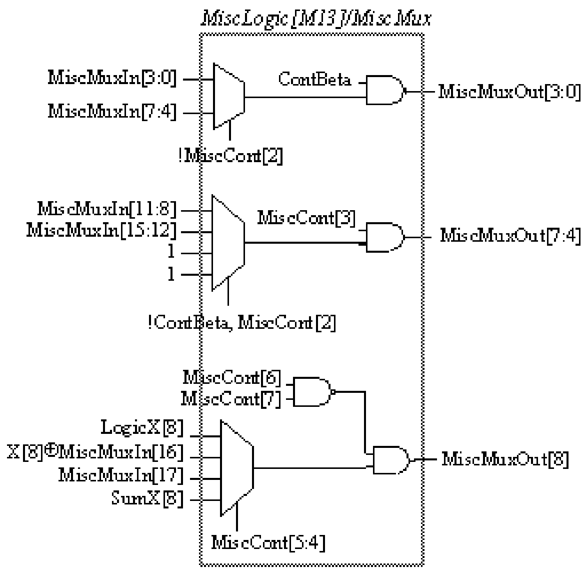
High-Level Model of c5315 [con't]



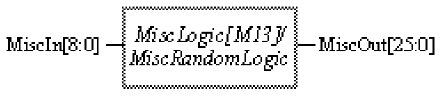


$\overline{\text{ContPar}}$ = ContParChk[0]...ContParChk[5]

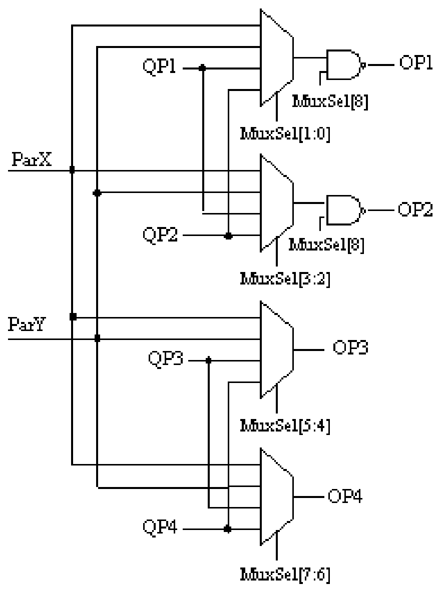
*High-Level Model of c5315 [con't]*



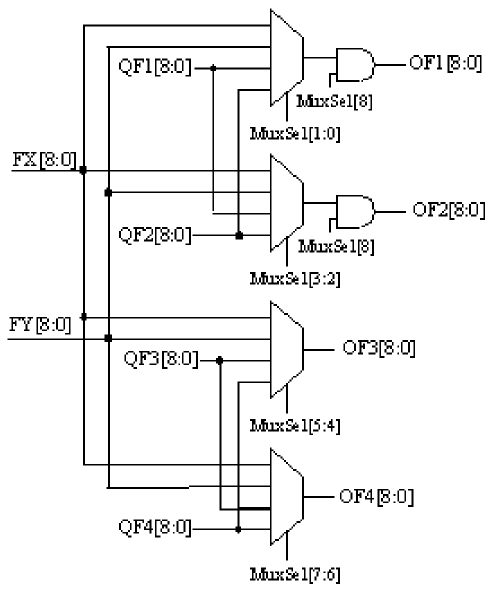
ContBeta = MiscCont[0].MiscCont[1]



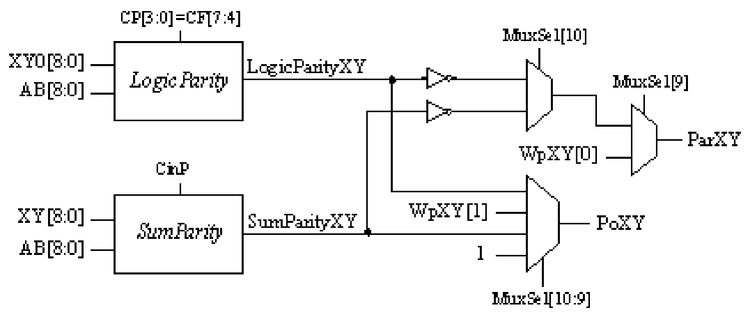
*MuxesPar [M5]*



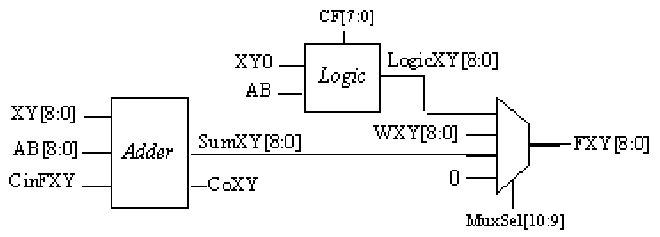
*MuxesF [M8]*



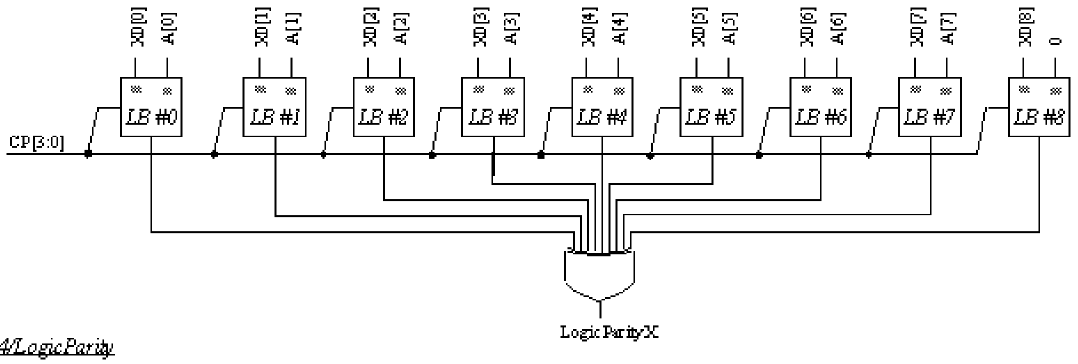
*Parity [M3-M4]*



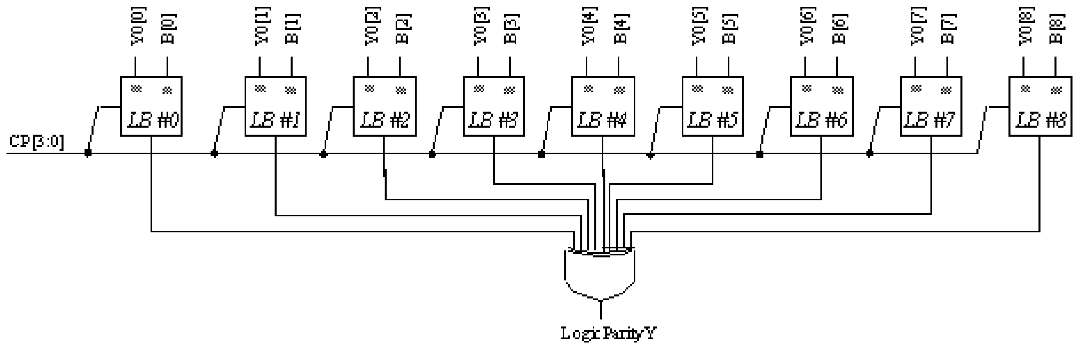
*Add-Logic [M6-M7]*



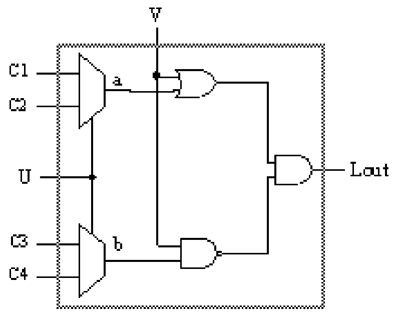
*M3/LogicParity*



*M4/LogicParity*



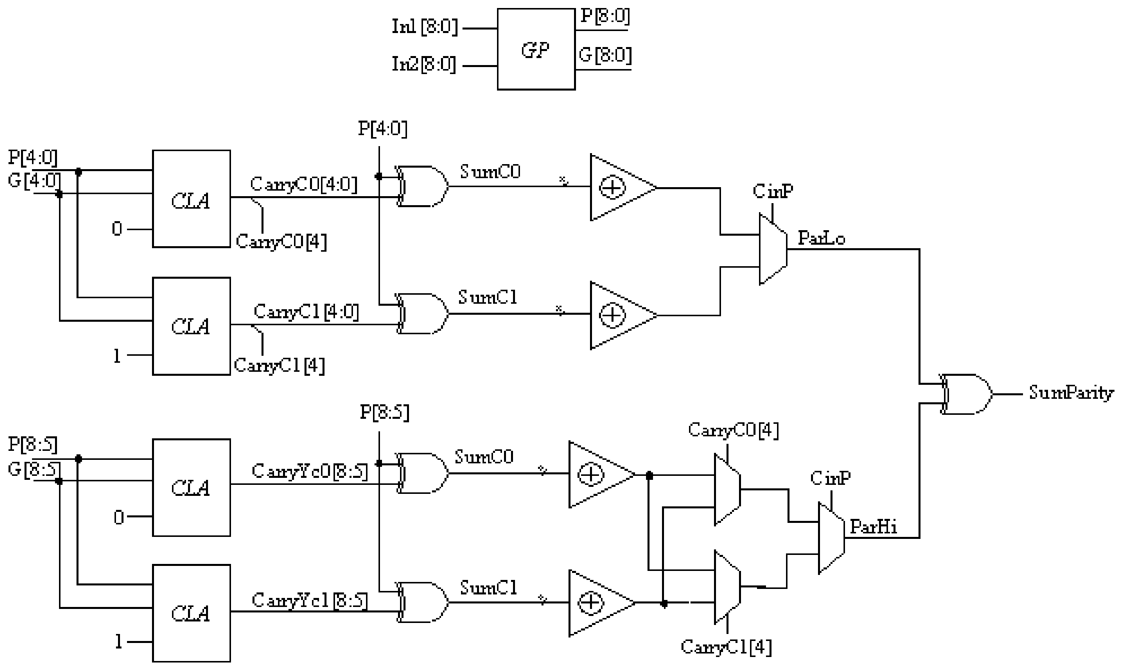
*Logic Block [LB]*



| C1 | C2 | C3 | C4 | a | b | Lout |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | V |
| 0 | 0 | 0 | 1 | 0 | U | U'.V |
| 0 | 0 | 1 | 0 | 0 | U' | U.V |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | U | 0 | U+V |
| 0 | 1 | 0 | 1 | U | U | U⊕V |
| 0 | 1 | 1 | 0 | U | U' | U |
| 0 | 1 | 1 | 1 | U | 1 | U.V' |
| 1 | 0 | 0 | 0 | U' | 0 | U'+V |
| 1 | 0 | 0 | 1 | U' | U | U' |
| 1 | 0 | 1 | 0 | U' | U' | [U⊕V]' |
| 1 | 0 | 1 | 1 | U' | 1 | U'.V' |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | U | U'+V' |
| 1 | 1 | 1 | 0 | 1 | U' | U+V' |
| 1 | 1 | 1 | 1 | 1 | 1 | V' |

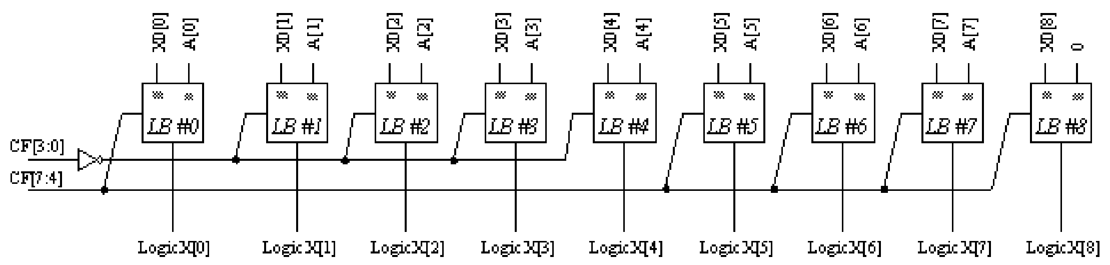
* Lout: all the 16 functions of two variables (U,V)

*SumParity*

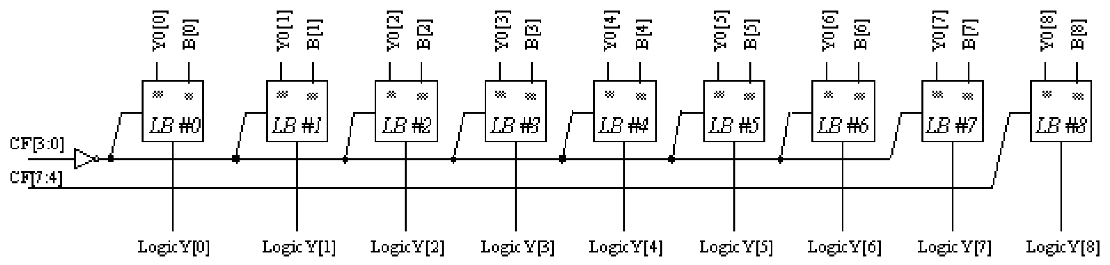


Note: Carry[i] = carry from bit position i to i+1.

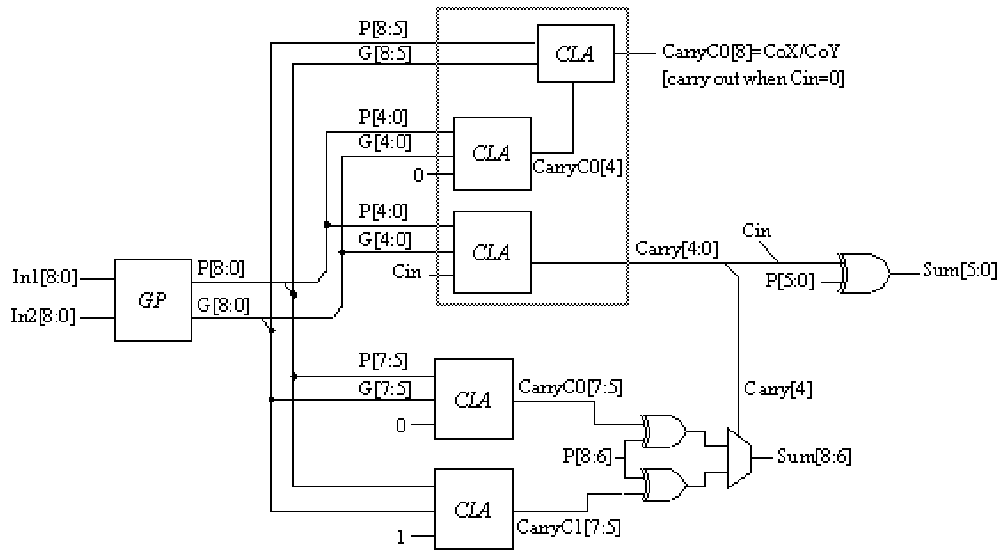
*M6/Logic*



*M7/Logic*



*Adder*



Note: Carry[i] = carry from bit position i to i+1.

| Input | Line number |
|---|---|
| X0[8:0] | 293, 302, 308, 316, 324, **logic 1**, 341, 351, 361 |
| X1[8:0] | 299, 307, 315, 323, 331, 338, 348, 358, 366 |
| MuxSelX | 332 |
| A[8:0] | **logic 1**, **logic 1**, 479, 490, 503, 514, 523, 534, **logic 1** |
| Y0[8:0] | 206, 210, 218, 226, 234, 257, 265, 273, 281 |
| Y1[8:0] | 209, 217, 225, 233, 241, 264, 272, 280, 288 |
| MuxSelY | 335 |
| B[8:0] | 446, 457, 468, 422, 435, 389, 400, 411, 374 |
| CinFX, CinFY | 54, 4 |
| CinPX,CinPY | 2174, 1497 |
| WpX[1:0] | 120, 94 |
| WpY[1:0] | 118, 97 |
| QP1,QP2,QP3,QP4 | 176, 179, 14, 64 |
| Q1[8:0] | 191, 194, 197, 203, 200, 149, 155, 188, 182 |
| Q2[8:0] | 161, 164, 167, 173, 170, 146, 152, 158, 185 |
| Q3[8:0] | 109, 46, 100, 91, 43, 76, 73, 67, 11 |
| Q4[8:0] | 106, 49, 103, 40, 37, 20, 17, 70, 61 |
| WFX[8:0] | 123, 121, 116, 112, 52, 130, 119, 129, 131 |
| WFY[8:0] | 115, 114, 53, 113, 122, 128, 127, 126, 117 |
| MuxSel[10:0] | 4091, 4092, 137, 4090, 4089, 4087, 4088, 1694, 1691, 1690, 1689 |
| CF[7:0] | 248, 251, 242, 254, 3552, 3550, 3546, 3548 |
| CP[3:0]=CF[7:4] | 248, 251, 242, 254 |
| ParYin= MuxSelY ? ParYin0 : ParYin1<br><br>(ParYin0, ParYin1) | 289, 292 |
| ParXin= MuxSelX ? ParXin0 : ParXin1<br><br>(ParXin0, ParXin1) | 369, 372 |

| | |
|---|---|
| ContParChk[5:0] | 562, 245, 552, 556, 559, 386 |
| MiscMuxIn[17:0] | 123 (=WFX[8]), 132, 23, 80, 25, 81, 79, 82, 24, 26, 86, 83, 88, 88, 87, 83, 34, 34 |
| MiscContIn[7:0] | 4115, 135, 3717, 3724, 141, 2358, 31, 27 |
| MiscIn[8:0] | 545, 549, 3173, 136, 1, 373, 145, 2824, 140 |
| Output | Line number |
| OP1,OP2,OP3,OP4 | 658, 690, 767, 807 |
| OF1[8:0] | 654, 642, 651, 648, 645, 670, 667, 664, 661 |
| OF2[8:0] | 688, 676, 685, 682, 679, 702, 699, 696, 693 |
| OF3[8:0] | 727, 747, 732, 737, 742, 752, 757, 762, 722 |
| OF4[8:0] | 712, 787, 772, 777, 782, 792, 797, 802, 859 |
| NXF[8:0] | 824, 826, 828, 830, 832, 834, 836, 838, 822 |
| NYF[8:0] | 863, 865, 867, 869, 871, 873, 875, 877, 861 |
| CoX,CoY | (629, 618) * , (591, 621) * |
| PoX,PoY | 843, 882 |
| ParityChk[4:0] | 998, 1002, 1000, 1004, 854 |
| ZeroFlagOut[3:0] | 585, 575, 598, 610 |
| MiscMuxOut[10:0] | 623, 813, 818, 707, 715, 639, 673, 636, 820, 717, 704 |
| MiscOut[25:0] | 593, 594, 602, 809, 611, 599, 612, 600, 850, 848, 849, 851, 887, 298, 926, 892, 973, 993, 144, 601, 847, 815, 634, 810, 845, 656 |

```
/*****************************************************************************
*****

 *                                                                  *

 *  VERILOG HIGH-LEVEL DESCRIPTION OF THE ISCAS-85 BENCHMARK
CIRCUIT c5315  *

 *                                                                  *

 *

******************************************************************************
****/


module Circuit5315(

       in293, in302, in308, in316, in324, in341, in351,

       in361, in299, in307, in315, in323, in331, in338, in348,

       in358, in366, in206, in210, in218, in226, in234, in257,

       in265, in273, in281, in209, in217, in225, in233, in241,

       in264, in272, in280, in288, in54, in4, in2174, in1497,

       in332, in335, in479, in490, in503, in514, in523, in534,

       in446, in457, in468, in422, in435, in389, in400, in411,

       in374, in191, in200, in194, in197, in203, in149, in155,

       in188, in182, in161, in170, in164, in167, in173, in146,

       in152, in158, in185, in109, in43, in46, in100, in91,

       in76, in73, in67, in11, in106, in37, in49, in103,

       in40, in20, in17, in70, in61, in123, in52, in121,

       in116, in112, in130, in119, in129, in131, in115, in122,

       in114, in53, in113, in128, in127, in126, in117, in176,

       in179, in14, in64, in248, in251, in242, in254, in3552,
       in3550, in3546, in3548, in120, in94, in118, in97, in4091,
       in4092, in137, in4090, in4089, in4087, in4088, in1694, in1691,
       in1690, in1689, in372, in369, in292, in289, in562, in245,
       in552, in556, in559, in386, in132, in23, in80, in25,
       in81, in79, in82, in24, in26, in86, in88, in87,
       in83, in34, in4115, in135, in3717, in3724, in141, in2358,
       in31, in27, in545, in549, in3173, in136, in1, in373,
       in145, in2824, in140,
       out658, out690, out767, out807, out654, out651, out648,
       out645, out642, out670, out667, out664, out661, out688, out685,
       out682, out679, out676, out702, out699, out696, out693, out727,
       out732, out737, out742, out747, out752, out757, out762, out722,
       out712, out772, out777, out782, out787, out792, out797, out802,
       out859, out824, out826, out832, out828, out830, out834, out836,
       out838, out822, out863, out871, out865, out867, out869, out873,
       out875, out877, out861, out629, out591, out618, out615, out621,
       out588, out626, out632, out843, out882, out585, out575, out598,
       out610, out998, out1002, out1000, out1004, out854, out623, out813,
       out818, out707, out715, out639, out673, out636, out820, out717,
       out704, out593, out594, out602, out809, out611, out599, out612,
       out600, out850, out848, out849, out851, out887, out298, out926,
       out892, out973, out993, out144, out601, out847, out815, out634,
       out810, out845, out656, out923, out939, out921, out978, out949,
       out889, out603, out604, out606);
```

```
input
    in293, in302, in308, in316, in324, in341, in351,
    in361, in299, in307, in315, in323, in331, in338, in348,
    in358, in366, in206, in210, in218, in226, in234, in257,
    in265, in273, in281, in209, in217, in225, in233, in241,
    in264, in272, in280, in288, in54, in4, in2174, in1497,
    in332, in335, in479, in490, in503, in514, in523, in534,
    in446, in457, in468, in422, in435, in389, in400, in411,
    in374, in191, in200, in194, in197, in203, in149, in155,
    in188, in182, in161, in170, in164, in167, in173, in146,
    in152, in158, in185, in109, in43, in46, in100, in91,
    in76, in73, in67, in11, in106, in37, in49, in103,
    in40, in20, in17, in70, in61, in123, in52, in121,
    in116, in112, in130, in119, in129, in131, in115, in122,
    in114, in53, in113, in128, in127, in126, in117, in176,
    in179, in14, in64, in248, in251, in242, in254, in3552,
    in3550, in3546, in3548, in120, in94, in118, in97, in4091,
    in4092, in137, in4090, in4089, in4087, in4088, in1694, in1691,
    in1690, in1689, in372, in369, in292, in289, in562, in245,
    in552, in556, in559, in386, in132, in23, in80, in25,
    in81, in79, in82, in24, in26, in86, in88, in87,
    in83, in34, in4115, in135, in3717, in3724, in141, in2358,
    in31, in27, in545, in549, in3173, in136, in1, in373,
    in145, in2824, in140;
```

```
  output
      out658, out690, out767, out807, out654, out651, out648,
      out645, out642, out670, out667, out664, out661, out688, out685,
      out682, out679, out676, out702, out699, out696, out693, out727,
      out732, out737, out742, out747, out752, out757, out762, out722,
      out712, out772, out777, out782, out787, out792, out797, out802,
      out859, out824, out826, out832, out828, out830, out834, out836,
      out838, out822, out863, out871, out865, out867, out869, out873,
      out875, out877, out861, out629, out591, out618, out615, out621,
      out588, out626, out632, out843, out882, out585, out575, out598,
      out610, out998, out1002, out1000, out1004, out854, out623, out813,
      out818, out707, out715, out639, out673, out636, out820, out717,
      out704, out593, out594, out602, out809, out611, out599, out612,
      out600, out850, out848, out849, out851, out887, out298, out926,
      out892, out973, out993, out144, out601, out847, out815, out634,
      out810, out845, out656, out923, out939, out921, out978, out949,
      out889, out603, out604, out606;


/************************/
  wire VDD;
  assign VDD = 1'b1;

  wire [8:0]    X0bus, X1bus, Abus;
  wire [8:0]    Y0bus, Y1bus, Bbus;
wire          CinFX, CinFY;
wire          CinParX, CinParY;
wire          MuxSelX, MuxSelY;
wire [10:0]   MuxSelPF;
wire [8:0]    QF1bus, QF2bus, QF3bus, QF4bus;
wire [8:0]    WXbus, WYbus;
wire          QP1, QP2, QP3, QP4;
wire [7:0]    ContLogic;
wire [1:0]    ParXin, ParYin;
wire [5:0]    ContParChk;
wire [16:0]   MiscMuxIn;
wire [7:0]    MiscContIn;
wire [8:0]    MiscInbus;
wire [1:0]    WparX, WparY;


wire [8:0]    OF1bus, OF2bus, OF3bus, OF4bus;
wire          OP1, OP2, OP3, OP4;
wire          SumLogicParXout, SumLogicParYout;
wire          CoutFX_in0, CoutFY_in0;
wire          PropThruX, PropThruY;
wire [8:0]    NotXFbus, NotYFbus;
wire [3:0]    ZeroFlagOut;
wire [4:0]    ParChkOut;
wire [10:0]   MiscMuxOut;
wire [25:0]   MiscOutbus;
```

```
/************************/

// inputs

  assign
    X0bus[8:0] = { in293, in302, in308, in316, in324,
                   VDD, in341, in351, in361 },
    X1bus[8:0] = { in299, in307, in315, in323, in331,
                   in338, in348, in358, in366 };
  assign
    Y0bus[8:0] = { in206, in210, in218, in226, in234,
                   in257, in265, in273, in281 },
    Y1bus[8:0] = { in209, in217, in225, in233, in241,
                   in264, in272, in280, in288 };
  assign
    CinFX = in54,     CinFY = in4,
    CinParX = in2174, CinParY = in1497;

  assign
    MuxSelX = in332, MuxSelY = in335;

  assign
    Abus[8:0] = { VDD, VDD, in479, in490, in503,
                  in514, in523, in534, VDD };
```

```
assign
  Bbus[8:0] = { in446, in457, in468, in422, in435,
                in389, in400, in411, in374 };
assign
  QF1bus[8:0] = { in191, in194, in197, in203, in200,
                  in149, in155, in188, in182 },
  QF2bus[8:0] = { in161, in164, in167, in173, in170,
                  in146, in152, in158, in185 },
  QF3bus[8:0] = { in109, in46, in100, in91, in43,
                  in76, in73, in67, in11 },
  QF4bus[8:0] = { in106, in49, in103, in40, in37,
                  in20, in17, in70, in61 };

assign
  WXbus[8:0] = { in123, in121, in116, in112, in52,
                 in130, in119, in129, in131 },
  WYbus[8:0] = { in115, in114, in53, in113, in122,
                 in128, in127, in126, in117 };

assign
  QP1 = in176, QP2 = in179, QP3 = in14, QP4 = in64;

assign
  ContLogic[7:0] = { in248, in251, in242, in254,
                     in3552, in3550, in3546, in3548 };
```

```
assign
  WparX[1:0] = { in120, in94 },
  WparY[1:0] = { in118, in97 };


assign
  MuxSelPF[10:0] = { in4091, in4092, in137, in4090, in4089, in4087,
                       in4088, in1694, in1691, in1690, in1689 };
assign
  ParXin[1:0] = { in372, in369 },
  ParYin[1:0] = { in292, in289 };


assign
  ContParChk[5:0] = { in562, in245, in552, in556, in559, in386 };


assign
  MiscMuxIn[16:0] = { in132, in23, in80, in25, in81,
                       in79, in82, in24, in26, in86, in83, in88, in88,
                       in87, in83, in34, in34 };
assign
  MiscContIn[7:0] = { in4115, in135, in3717, in3724,
                       in141, in2358, in31, in27 };
assign
  MiscInbus[8:0] = { in545, in549, in3173, in136, in1,
                       in373, in145, in2824, in140 };
```

```
// outputs

  assign
    out658 = OP1, out690 = OP2, out767 = OP3, out807 = OP4;

  assign
    { out654, out651, out648, out645, out642,
        out670, out667, out664, out661 } = OF1bus[8:0],
    { out688, out685, out682, out679, out676,
        out702, out699, out696, out693 } = OF2bus[8:0],
    { out727, out732, out737, out742, out747,
        out752, out757, out762, out722 } = OF3bus[8:0],
    { out712, out772, out777, out782, out787,
        out792, out797, out802, out859 } = OF4bus[8:0];

  assign
    { out824, out826, out828, out830, out832,
        out834, out836, out838, out822 } = NotXFbus[8:0],
    { out863, out865, out867, out869, out871,
        out873, out875, out877, out861 } = NotYFbus[8:0];

  assign
    out629 = CoutFX_in0, out591 = CoutFY_in0,
    out618 = CoutFX_in0, out621 = CoutFY_in0;
  assign
    out615 = PropThruX, out588 = PropThruY,
    out626 = PropThruX, out632 = PropThruY;

  assign
    out843 = SumLogicParXout, out882 = SumLogicParYout;

  assign
    { out585, out575, out598, out610 } = ZeroFlagOut[3:0];

  assign
    { out998, out1002, out1000, out1004, out854 } = ParChkOut[4:0];

  assign
    { out623, out813, out818, out707, out715, out639,
        out673, out636, out820, out717, out704 } = MiscMuxOut[10:0];
  assign
    { out593, out594, out602, out809, out611, out599,
        out612, out600, out850, out848, out849, out851,
        out887, out298, out926, out892, out973, out993,
        out144, out601, out847, out815, out634, out810,
        out845, out656 } = MiscOutbus[25:0];

// identical misc. outputs
  assign
```

```
     out923 = out144, out939 = out993, out921 = out993,
     out978 = out993, out949 = out993, out889 = out887,
     out603 = out594, out604 = out594, out606 = out602;


/* instantiate top level circuit */

  TopLevel5315 Ckt5315( X0bus, X1bus, Abus, Y0bus, Y1bus, Bbus, CinFX, CinFY,
                        CinParX, CinParY, MuxSelX, MuxSelY, MuxSelPF,
                        QF1bus, QF2bus, QF3bus, QF4bus, QP1, QP2, QP3, QP4,
                        WXbus, WYbus, ContLogic, ParXin, ParYin, ContParChk,
                        MiscMuxIn, MiscContIn, MiscInbus, WparX, WparY,

                        OF1bus, OF2bus, OF3bus, OF4bus, OP1, OP2, OP3, OP4,
                        SumLogicParXout, SumLogicParYout, CoutFX_in0,
CoutFY_in0,
                        PropThruX, PropThruY, NotXFbus, NotYFbus, ZeroFlagOut,
                        ParChkOut, MiscMuxOut, MiscOutbus     );


endmodule // Circuit5315


/*********************************************************************
****/
```

```
/*******************************************************************
****/


module TopLevel5315( X0bus, X1bus, Abus, Y0bus, Y1bus, Bbus, CinFX, CinFY,
                    CinParX, CinParY, MuxSelX, MuxSelY, MuxSelPF,
                    QF1bus, QF2bus, QF3bus, QF4bus, QP1, QP2, QP3, QP4,
                    WXbus, WYbus, ContLogic, ParXin, ParYin, ContParChk,
                    MiscMuxIn, MiscContIn, MiscInbus, WparX, WparY,


                    OF1bus, OF2bus, OF3bus, OF4bus, OP1, OP2, OP3, OP4,
                    SumLogicParXout, SumLogicParYout, CoutFX_in0, CoutFY_in0,
                    PropThruX, PropThruY, NotFXbus, NotFYbus, ZeroFlagOut,
                    ParChkOut, MiscMuxOut, MiscOutbus    );


  input [8:0]     X0bus, X1bus, Abus;
  input [8:0]     Y0bus, Y1bus, Bbus;
  input  CinFX, CinFY;
  input  CinParX, CinParY;
  input  MuxSelX, MuxSelY;
  input [10:0]   MuxSelPF;
  input [8:0]     QF1bus, QF2bus, QF3bus, QF4bus;
  input  QP1, QP2, QP3, QP4;
  input [8:0]     WXbus, WYbus;
  input [1:0]     WparX, WparY;
  input [7:0]     ContLogic;
  input [1:0]     ParXin, ParYin;

  input [5:0]     ContParChk;

  input [16:0]    MiscMuxIn;

  input [7:0]     MiscContIn;

  input [8:0]     MiscInbus;


  output [8:0]    OF1bus, OF2bus, OF3bus, OF4bus;

  output          OP1, OP2, OP3, OP4;

  output          SumLogicParXout, SumLogicParYout;

  output          CoutFX_in0, CoutFY_in0;

  output          PropThruX, PropThruY;

  output [8:0]    NotFXbus, NotFYbus;

  output [3:0]    ZeroFlagOut;

  output [4:0]    ParChkOut;

  output [10:0] MiscMuxOut;

  output [25:0] MiscOutbus;



  wire [8:0]      Xbus, Ybus;

  wire [8:0]      FXbus, FYbus;

  wire [8:0]      SumXbus, LogicXbus, SumYbus, LogicYbus;

  wire [3:0]      ContLogicPar, NotContLogic3_0;

  wire [35:0]     ContLogicInX, ContLogicInY;

  wire            Not_SumLogicParX, Not_SumLogicParY;
```

```
   wire GND;

   assign GND = 1'b0;


   Mux9bit_2_1 M1( X0bus, X1bus, MuxSelX, Xbus );

   Mux9bit_2_1 M2( Y0bus, Y1bus, MuxSelY, Ybus );


   assign ContLogicPar[3:0] = ContLogic[7:4];


// parity blocks


   CalcParity  M3( X0bus, { GND, Abus[7:0] }, Xbus, Abus, WparX,

                        MuxSelPF[10:9], ContLogicPar, CinParX,

                        Not_SumLogicParX, SumLogicParXout );

   CalcParity  M4( Y0bus, Bbus, Ybus, Bbus, WparY,

                        MuxSelPF[10:9], ContLogicPar, CinParY,

                        Not_SumLogicParY, SumLogicParYout );


   MuxesPar_4  M5( Not_SumLogicParX, Not_SumLogicParY, QP1, QP2, QP3, QP4,

                        MuxSelPF[8:0], OP1, OP2, OP3, OP4 );


// sum-logic blocks


   Invert4 M0( ContLogic[3:0], NotContLogic3_0 );

   assign

     ContLogicInX[35:0] = { ContLogicPar, ContLogicPar, ContLogicPar,
                        ContLogicPar, NotContLogic3_0, NotContLogic3_0,
                        NotContLogic3_0, NotContLogic3_0, ContLogicPar },

   ContLogicInY[35:0] = { ContLogicPar, NotContLogic3_0, NotContLogic3_0,
                        NotContLogic3_0, NotContLogic3_0, NotContLogic3_0,
                        NotContLogic3_0, NotContLogic3_0, NotContLogic3_0 };

  CalcSumLogic M6( X0bus, { GND, Abus[7:0] }, Xbus, Abus, CinFX, WXbus,
                 ContLogicInX, MuxSelPF[10:9],
                 LogicXbus, SumXbus, FXbus, CoutFX_in0, PropThruX );

  CalcSumLogic M7( Y0bus, Bbus, Ybus, Bbus, CinFY, WYbus,
                 ContLogicInY, MuxSelPF[10:9],
                 LogicYbus, SumYbus, FYbus, CoutFY_in0, PropThruY );

  MuxesF8bit_4 M8( FXbus, FYbus, QF1bus, QF2bus, QF3bus, QF4bus,
MuxSelPF[8:0],
                 OF1bus, OF2bus, OF3bus, OF4bus );

// other logic

  Invert9 M9( FXbus, NotFXbus ),
       M10( FYbus, NotFYbus );

  ZeroFlags M11( SumXbus, LogicXbus, SumYbus, LogicYbus, ZeroFlagOut );
```

```
   BusParityChk M12( X0bus, Xbus, Y0bus, Ybus, ParXin, ParYin,
                     MuxSelX, MuxSelY, ContParChk, ParChkOut );

// miscellaneous logic

   MiscLogic M13( MiscMuxIn, MiscContIn, MiscInbus, ContParChk,
                  Xbus[8], LogicXbus[8], SumXbus[8], WXbus[8],
                  X1bus[3:0], X1bus[8], X0bus[8], MuxSelPF[8],
                  MiscMuxOut, MiscOutbus );



endmodule // TopLevel5315

/***********************************************************************
****
 * Module: Mux9bit_2_1
 *
 * Function: 9-bit 2:1 Muxes
************************************************************************
***/


module Mux9bit_2_1( In0, In1, ContIn, Out );
   input [8:0]     In0, In1;
   input           ContIn;
   output [8:0] Out;
```

```
  Mux4bit_2_1 Mux9_0( In0[3:0], In1[3:0], ContIn, Out[3:0] ),
           Mux9_1( In0[7:4], In1[7:4], ContIn, Out[7:4] );
  Mux2_1     Mux9_2( In0[8], In1[8], ContIn, Out[8] );

endmodule // Mux9bit_2_1

/*****************************************/

module Mux4bit_2_1( In0, In1, ContIn, Out );
  input [3:0]   In0, In1;
  input ContIn;
  output [3:0]  Out;

  Mux2_1 Mux4_0( In0[0], In1[0], ContIn, Out[0] ),
  Mux4_1( In0[1], In1[1], ContIn, Out[1] ),
  Mux4_2( In0[2], In1[2], ContIn, Out[2] ),
  Mux4_3( In0[3], In1[3], ContIn, Out[3] );

endmodule // Mux4bit_2_1


/*******************************************************************
****
 * Module: CalcParity
 *
 * Function: calculates the parity of the result (XYsumbus+ABsumbus+CinPar),
 *  and of (XYlogicbus OPR ABlogicbus), where OPR is a logical operator
 *  specified by ContLogicPar.
 *
 *  - ContLogicPar is 4 bits wide, so the parity of 16 different logical
 *    functions can be calculated.
 *

**********************************************************************
***/

module CalcParity( XYlogicbus, ABlogicbus, XYsumbus, ABsumbus, Wpar,
                   MuxSel, ContLogicPar, CinPar,
                   NotSumLogicPar, SumLogicParOut );

  input [8:0] XYlogicbus, ABlogicbus;
  input [8:0] XYsumbus, ABsumbus;
  input [1:0] Wpar;
  input [1:0] MuxSel;
  input [3:0] ContLogicPar;
  input       CinPar;
  output      NotSumLogicPar, SumLogicParOut;

  LogicParity CalP0( XYlogicbus, ABlogicbus, ContLogicPar, LogicPar );
  SumParity   CalP1( XYsumbus, ABsumbus, CinPar, SumPar );
  Muxes2_Mux4 CalP2( LogicPar, SumPar, Wpar, MuxSel,
                  NotSumLogicPar, SumLogicParOut );

endmodule // CalcParity

/*******************************************/

module LogicParity( XYlogicbus, ABlogicbus, ContLogicPar, LogicPar );

  input [8:0] XYlogicbus, ABlogicbus;
  input [3:0] ContLogicPar;
  output      LogicPar;

  wire [35:0] ContLogicIn;
  wire [8:0]  LogicOut;

  assign
    ContLogicIn[35:0] = { ContLogicPar, ContLogicPar, ContLogicPar,
                          ContLogicPar, ContLogicPar, ContLogicPar,
                          ContLogicPar, ContLogicPar, ContLogicPar };

  ComputeLogic   LP0( XYlogicbus, ABlogicbus, ContLogicIn, LogicOut );

  ParityTree9bit LP1( LogicOut, LogicPar );
```

```
endmodule // LogicParity

/******************************************/

module ComputeLogic( In1bus, In2bus, ContLogicIn, Outbus );

  input [8:0]    In1bus, In2bus;
  input [35:0]  ContLogicIn;
  output [8:0]  Outbus;

  LogicBlock CL0( In1bus[0], In2bus[0], ContLogicIn[3:0],   Outbus[0] ),
  CL1( In1bus[1], In2bus[1], ContLogicIn[7:4],   Outbus[1] ),
  CL2( In1bus[2], In2bus[2], ContLogicIn[11:8],  Outbus[2] ),
  CL3( In1bus[3], In2bus[3], ContLogicIn[15:12], Outbus[3] ),
  CL4( In1bus[4], In2bus[4], ContLogicIn[19:16], Outbus[4] ),
  CL5( In1bus[5], In2bus[5], ContLogicIn[23:20], Outbus[5] ),
  CL6( In1bus[6], In2bus[6], ContLogicIn[27:24], Outbus[6] ),
  CL7( In1bus[7], In2bus[7], ContLogicIn[31:28], Outbus[7] ),
  CL8( In1bus[8], In2bus[8], ContLogicIn[35:32], Outbus[8] );

endmodule // ComputeLogic

/******************************************
 * LogicBlock: implements all 16 functions of
 *  In1 and In2 as selected by the 4-bit
 *  ContLogic input.
 ******************************************/

module LogicBlock( In1, In2, ContLogic, Out );

  input       In1, In2;
  input [3:0] ContLogic;
  output      Out;

  Mux2_1 LB0( ContLogic[0], ContLogic[1], In1, line0),
  LB1( ContLogic[2], ContLogic[3], In1, line1);
  or2    LB2( .A(In2), .B(line0), .Y(line2) );
  nand2    LB3( .A(In2), .B(line1), .Y(line3) );
  and2    LB4( .A(line2), .B(line3), .Y(Out) );

endmodule // LogicBlock

/*****************************************************************
 * Submodule: SumParity
 *
 * Function: calculates the parity of the sum (In1bus + In2bus + Cin)
 *
 *  The parity is calculated separately for the lower 5-bit block
 *  and the upper 4-bit block. In each case, two parities are calculated:
```

```
 *  one with an assumed carry of 0 to that block, and another with 1.
 *  For the 5-bit block, the correct parity is determined by Cin.
 *  For the 4-bit block, the carry input Cin as well as the carry from
 *  the (lower) 5-bit block to the (higher) 4-bit block determine
 *  the correct parity.
 *

***********************************************************************
/


module SumParity( In1bus, In2bus, Cin, SumPar );


  input [8:0] In1bus, In2bus;
  input       Cin;
  output      SumPar;


  wire [8:0]  Genbus, Propbus;
  wire [8:0]  LocalC0, LocalC1;


  GenProp9       SP0( In1bus, In2bus, Genbus, Propbus );


  // first caculate the local carries
  //  (local carries in 8th position are not needed)


  GenLocalCarry5 SP1( Genbus[4:0], Propbus[4:0], LocalC0[4:0], LocalC1[4:0] );
  GenLocalCarry3 SP2( Genbus[7:5], Propbus[7:5], LocalC0[7:5], LocalC1[7:5] );
  SerialParity9nc SP3( { Propbus[4:0], LocalC0[3:0] }, ParLo0 );

  SerialParity9c  SP4( { Propbus[4:0], LocalC1[3:0] }, ParLo1 );

  SerialParity7nc SP5( { Propbus[8:5], LocalC0[7:5] }, ParHi0 );

  SerialParity7c  SP6( { Propbus[8:5], LocalC1[7:5] }, ParHi1 );


  Mux2_1 SP7( ParLo0, ParLo1, Cin, ParLo),

  SP8( ParHi0, ParHi1, LocalC0[4], ParHiCin0),

  SP9( ParHi0, ParHi1, LocalC1[4], ParHiCin1),

  SP10( ParHiCin0, ParHiCin1, Cin, ParHi);


  XOR2a  SP11( .A(ParLo), .B(ParHi), .Y(SumPar) );


endmodule // SumParity


/*******************************************/


module GenProp9( In1bus, In2bus, Gbus, Pbus);


  input [8:0]    In1bus, In2bus;

  output [8:0]  Gbus, Pbus;


  and2  GP9_0( .A(In1bus[0]), .B(In2bus[0]), .Y(Gbus[0]) ),

  GP9_1( .A(In1bus[1]), .B(In2bus[1]), .Y(Gbus[1]) ),

  GP9_2( .A(In1bus[2]), .B(In2bus[2]), .Y(Gbus[2]) ),
  GP9_3( .A(In1bus[3]), .B(In2bus[3]), .Y(Gbus[3]) ),
  GP9_4( .A(In1bus[4]), .B(In2bus[4]), .Y(Gbus[4]) ),
  GP9_5( .A(In1bus[5]), .B(In2bus[5]), .Y(Gbus[5]) ),
  GP9_6( .A(In1bus[6]), .B(In2bus[6]), .Y(Gbus[6]) ),
  GP9_7( .A(In1bus[7]), .B(In2bus[7]), .Y(Gbus[7]) ),
  GP9_8( .A(In1bus[8]), .B(In2bus[8]), .Y(Gbus[8]) );

  XOR2a GP9_9( .A(In1bus[0]), .B(In2bus[0]), .Y(Pbus[0]) ),
  GP9_10( .A(In1bus[1]), .B(In2bus[1]), .Y(Pbus[1]) ),
  GP9_11( .A(In1bus[2]), .B(In2bus[2]), .Y(Pbus[2]) ),
  GP9_12( .A(In1bus[3]), .B(In2bus[3]), .Y(Pbus[3]) ),
  GP9_13( .A(In1bus[4]), .B(In2bus[4]), .Y(Pbus[4]) ),
  GP9_14( .A(In1bus[5]), .B(In2bus[5]), .Y(Pbus[5]) ),
  GP9_15( .A(In1bus[6]), .B(In2bus[6]), .Y(Pbus[6]) ),
  GP9_16( .A(In1bus[7]), .B(In2bus[7]), .Y(Pbus[7]) ),
  GP9_17( .A(In1bus[8]), .B(In2bus[8]), .Y(Pbus[8]) );

endmodule // GenProp9

/******************************************/

module GenLocalCarry5( Gbus, Pbus, LocalC0, LocalC1 );

  input [4:0]    Gbus, Pbus;
  output [4:0]  LocalC0, LocalC1;
```

```
   GenLocalCarry4 GLC5_0( Gbus[3:0], Pbus[3:0],
                                    LocalC0[3:0], LocalC1[3:0] );

   AND_OR5a GLC5_1( Gbus[4], Pbus[4], Gbus[3], Pbus[3], Gbus[2],
                         Pbus[2], Gbus[1], Pbus[1], Gbus[0],
                         LocalC0[4] );
   AND_OR6b GLC5_2( Gbus[4], Pbus[4], Gbus[3], Pbus[3], Gbus[2],
                         Pbus[2], Gbus[1], Pbus[1], Gbus[0], Pbus[0],
                         LocalC1[4] );

endmodule // GenLocalCarry5

/****************************************************/

module GenLocalCarry4( Gbus, Pbus, LocalC0, LocalC1 );

   input [3:0]    Gbus, Pbus;
   output [3:0]  LocalC0, LocalC1;

   GenLocalCarry3 GLC4_0( Gbus[2:0], Pbus[2:0],
                                    LocalC0[2:0], LocalC1[2:0] );

   AND_OR4a GLC4_1( Gbus[3], Pbus[3], Gbus[2], Pbus[2], Gbus[1],
                         Pbus[1], Gbus[0], LocalC0[3] );
```

```
  AND_OR5b GLC4_2( Gbus[3], Pbus[3], Gbus[2], Pbus[2], Gbus[1],
                   Pbus[1], Gbus[0], Pbus[0], LocalC1[3] );

endmodule // GenLocalCarry4

/****************************************************/

module GenLocalCarry3( Gbus, Pbus, LocalC0, LocalC1 );

  input [2:0]   Gbus, Pbus;
  output [2:0]  LocalC0, LocalC1;

  assign LocalC0[0] = Gbus[0];
  or2 GLC4_0( .A(Gbus[0]), .B(Pbus[0]), .Y(LocalC1[0]) );

  AND_OR2  GLC4_1( Gbus[1], Pbus[1], Gbus[0], LocalC0[1] );
  AND_OR3b GLC4_2( Gbus[1], Pbus[1], Gbus[0], Pbus[0], LocalC1[1] );

  AND_OR3a GLC4_3( Gbus[2], Pbus[2], Gbus[1], Pbus[1], Gbus[0],
                   LocalC0[2] );
  AND_OR4b GLC4_4( Gbus[2], Pbus[2], Gbus[1], Pbus[1], Gbus[0],
                   Pbus[0], LocalC1[2] );

endmodule // GenLocalCarry3
```

```
/****************************************************/

module SerialParity9nc( Inbus, Out);

  input [8:0] Inbus;
  output      Out;

  SerialParity7nc SP9nc0( Inbus[6:0], line0 );
  XOR2a           SP9nc1( .A(Inbus[7]), .B(line0), .Y(line1) ),
  SP9nc2( .A(Inbus[8]), .B(line1), .Y(Out) );

endmodule // SerialParity9nc

/****************************************************/

module SerialParity9c( Inbus, Out);

  input [8:0] Inbus;
  output      Out;

  // Inbus[6] is inverted in SerialParity7c
  SerialParity7c SP9nc0( Inbus[6:0], line0 );
  XOR2a          SP9nc1( .A(Inbus[7]), .B(line0), .Y(line1) ),
  SP9nc2( .A(Inbus[8]), .B(line1), .Y(Out) );
```

```
endmodule // SerialParity9c

/****************************************************/

module SerialParity7nc( Inbus, Out);

  input [6:0] Inbus;
  output      Out;

  XOR2a SP7nc0( .A(Inbus[0]), .B(Inbus[1]), .Y(line0) ),
  SP7nc1( .A(Inbus[2]), .B(line0), .Y(line1) ),
  SP7nc2( .A(Inbus[3]), .B(line1), .Y(line2) ),
  SP7nc3( .A(Inbus[4]), .B(line2), .Y(line3) ),
  SP7nc4( .A(Inbus[5]), .B(line3), .Y(line4) ),
  SP7nc5( .A(Inbus[6]), .B(line4), .Y(Out) );

endmodule // SerialParity7nc

/****************************************************/

module SerialParity7c( Inbus, Out);

  input [6:0] Inbus;
  output      Out;
  wire [6:0]  NewInbus;

  // invert one bit to complement the output
  // -- Inbus[6] is chosen so the inverter is not on the longest path

  inv  SP7c0( .A(Inbus[6]), .Y(NewInbus[6]) );
  assign NewInbus[5:0] = Inbus[5:0];

  SerialParity7nc SP7c2( NewInbus, Out );

endmodule // SerialParity7c

/****************************************************/

module Muxes2_Mux4( LogicPar, SumPar, Wpar, MuxSel,
                    NotSumLogicPar, SumLogicParOut );

  input       LogicPar, SumPar;
  input [1:0] Wpar, MuxSel;
  output      NotSumLogicPar, SumLogicParOut;

  inv    M2M4_0( .A(LogicPar), .Y(NotLogicPar) ),
  M2M4_1( .A(SumPar), .Y(NotSumPar) );
  Mux2_1 M2M4_2( NotLogicPar, NotSumPar, MuxSel[1], line0 ),
  M2M4_3( line0, Wpar[0], MuxSel[0], NotSumLogicPar );
  Mux4_1 M2M4_4( LogicPar, Wpar[1], SumPar, 1'b1,
                 MuxSel[1], MuxSel[0], SumLogicParOut );

endmodule // Muxes2_Mux4



/**********************************************************************
****
 * Module: MuxesPar_4
 *
 * Function: includes a set of 4 muxes.
 *  The outputs of two of the muxes can be masked with an AND gate.
 *

***********************************************************************
***/

module MuxesPar_4( ParX, ParY, QP1, QP2, QP3, QP4, MuxSelbus,
                   OP1, OP2, OP3, OP4 );

  input      ParX, ParY, QP1, QP2, QP3, QP4;
  input [8:0] MuxSelbus;
  output     OP1, OP2, OP3, OP4;

  Muxes4  MP0( ParX, ParY, QP1, QP2, QP3, QP4, MuxSelbus,
                 NotOP1, NotOP2, OP3, OP4 );
  inv    MP1( .A(NotOP1), .Y(OP1) ),
  MP2( .A(NotOP2), .Y(OP2) );

endmodule // MuxesPar_4

/*******************************************/

module Muxes4( InM1, InM2, In1, In2, In3, In4, MuxSelbus,
             Out1, Out2, Out3, Out4 );

  input      InM1, InM2, In1, In2, In3, In4;
  input [8:0] MuxSelbus;
  output     Out1, Out2, Out3, Out4;

  Mux4_1 MXS0( InM1, InM2, In1, In2, MuxSelbus[1], MuxSelbus[0], tempOut1 ),
  MXS1( InM1, InM2, In1, In2, MuxSelbus[3], MuxSelbus[2], tempOut2 ),
  MXS2( InM1, InM2, In3, In4, MuxSelbus[5], MuxSelbus[4], Out3 ),
  MXS3( InM1, InM2, In3, In4, MuxSelbus[7], MuxSelbus[6], Out4 );

  and2  MXS4( .A(tempOut1), .B(MuxSelbus[8]), .Y(Out1) ),
  MXS5( .A(tempOut2), .B(MuxSelbus[8]), .Y(Out2) );

endmodule // Muxes4
```

```
/***********************************************************************
****
 * Module: CalcSumLogic
 *
 * Function: calculates the sum (XYsumbus + ABsumbus + Cin), and
 * the logical operation (XYlogicbus OPR ABlogicbus), both of which
 * are 9 bits wide.
 *
 * -Note that the OPR is not uniform for all bit positions; that's why
 *  it's 36 bits wide, 4 bits for each bit.
 *
 * -Also computed by the Adder9 module are Cout_in0 and PropThru.
 *   Cout_in0: the carry-out bit assuming Cin=0
 *   PropThru: AND of all propagate bits, so it indicates whether
 *   Cin can propagate all the way through 9 bits.
 *  (The actual carry output can be calculated by Cout_in0+Cin.PropThru)
 *

***********************************************************************
***/


module CalcSumLogic( XYlogicbus, ABlogicbus, XYsumbus, ABsumbus, Cin,
WXYbus,
                    ContLogicIn, MuxSel,
                    Logicbus, Sumbus, FXYbus, Cout_in0, PropThru );
```

```
  input [8:0]   XYlogicbus, ABlogicbus;

  input [8:0]   XYsumbus, ABsumbus;

  input Cin;

  input [8:0]   WXYbus;

  input [35:0]  ContLogicIn;

  input [1:0]   MuxSel;

  output [8:0]  Sumbus, Logicbus;

  output [8:0]  FXYbus;

  output        Cout_in0, PropThru;



  ComputeLogic CSL0( XYlogicbus, ABlogicbus, ContLogicIn, Logicbus );


  Adder9      CSL1( XYsumbus, ABsumbus, Cin, Sumbus, Cout_in0, PropThru );


  Mux9bit_4_1  CSL2( Logicbus, WXYbus, Sumbus, { 9'b000000000 },

                     MuxSel[1], MuxSel[0], FXYbus );


endmodule // CalcSumLogic


/*****************************************************************

 * Submodule: Adder9

 *

 * Function: calculates the sum (In1bus + In2bus + Cin).

 *
 *  The structure of this adder is slightly different from the
 *  one that computes the parity of the result.
 *  A CLA is used to compute the sum outputs for the lower
 *  6 bits. Two sets of sum signals are computed for the upper
 *  3 bits: one assuming carry[4]=0, and another assuming carry[4]=1
 *  The actual carry[4] signal selects the correct sum bits.
 *
 *****************************************************************/

module Adder9 ( In1bus, In2bus, Cin, Sumbus, Cout_in0, PropThru );

  input [8:0]    In1bus, In2bus;
  input Cin;
  output [8:0]  Sumbus;
  output         Cout_in0, PropThru;

  wire [8:0]     Genbus, Propbus;
  wire [2:0]     LocalHC0, LocalHC1;       // for bits # 7-5
  wire [4:0]     Carry;
  wire [5:0]     SumH01bus;


  GenProp9 Add0( In1bus, In2bus, Genbus, Propbus );

  // generate actual carry lines #0-4
```

```
   // Cout_in0 is the carry for the entire operation with Cin=0

   CLAblock Add1( Genbus, Propbus, Cin, Carry, Cout_in0, PropThru );

   // generate local carries for bits #7-5
   GenLocalCarry3 Add2( Genbus[7:5], Propbus[7:5], LocalHC0, LocalHC1 );

   // for bits # 0-5, generate sum directly : prop XOR carry
   XOR2a6bit Add3( Propbus[5:0], { Carry[4:0], Cin }, Sumbus[5:0] );

   // for bits #6-8, generate two sums, one assuming Carry[4]=0,
   //                        the other assuming Carry[4]=1
   XOR2a6bit Add4( { Propbus[8:6], Propbus[8:6] },
                   { LocalHC1[2:0], LocalHC0[2:0] }, SumH01bus );

   // now choose the correct sums #6-8
   Mux2_1 Add5( SumH01bus[0], SumH01bus[3], Carry[4], Sumbus[6] ),
   Add6( SumH01bus[1], SumH01bus[4], Carry[4], Sumbus[7] ),
   Add7( SumH01bus[2], SumH01bus[5], Carry[4], Sumbus[8] );

endmodule // Adder9

/******************************************/

module CLAblock( Gbus, Pbus, Cin, Carry, Cout_in0, PropThru );
```

```
input [8:0]    Gbus, Pbus;

input Cin;

output [4:0]  Carry;

output         Cout_in0, PropThru;


wire           LocalC0_4;


// actual carry lines #0-3

AND_OR2  CB0( Gbus[0], Pbus[0], Cin, Carry[0] );

AND_OR3a CB1( Gbus[1], Pbus[1], Gbus[0], Pbus[0], Cin, Carry[1] );

AND_OR4a CB2( Gbus[2], Pbus[2], Gbus[1], Pbus[1], Gbus[0],

               Pbus[0], Cin, Carry[2] );

AND_OR5a CB3( Gbus[3], Pbus[3], Gbus[2], Pbus[2], Gbus[1], Pbus[1],

               Gbus[0], Pbus[0], Cin, Carry[3] );


// LocalC0_4 is the carry out of bit #4 with Cin=0

AND_OR5a CB4( Gbus[4], Pbus[4], Gbus[3], Pbus[3], Gbus[2], Pbus[2],

               Gbus[1], Pbus[1], Gbus[0], LocalC0_4 );


and5     CB5( .A(Pbus[0]), .B(Pbus[1]), .C(Pbus[2]),

               .D(Pbus[3]), .E(Pbus[4]), .Y(Prop4_0) );

and2     CB6( .A(Cin), .B(Prop4_0), .Y(PropCin) );

or2     CB7( .A(LocalC0_4), .B(PropCin), .Y(Carry[4]) );
```

```
  // now Cout_in0 (the carryout line for the entire operation with Cin=0)
  AND_OR5a CB8( Gbus[8], Pbus[8], Gbus[7], Pbus[7], Gbus[6], Pbus[6],
                    Gbus[5], Pbus[5], LocalC0_4, Cout_in0 );

  // Propthr: and of all propagate lines
  and4 CB9( .A(Pbus[5]), .B(Pbus[6]), .C(Pbus[7]), .D(Pbus[8]),
             .Y(Prop8_5) );
  and2 CB10( .A(Prop4_0), .B(Prop8_5), .Y(PropThru) );

endmodule // CLAblock


/**********************************************************************
****
 * Module: MuxesF8bit_4
 *
 * Function: includes four sets of 9-bit Muxes whose inputs are
 *  FXbus and FYbus, the outputs of the CalcSumLogic modules, and
 *  input buses QF1, QF2, QF3, QF4.
 *

**********************************************************************
***/

module MuxesF8bit_4( FXbus, FYbus, QF1bus, QF2bus, QF3bus, QF4bus, MuxSelbus,
                       OF1bus, OF2bus, OF3bus, OF4bus );
  input [8:0]    FXbus, FYbus, QF1bus, QF2bus, QF3bus, QF4bus;
  input [8:0]    MuxSelbus;
  output [8:0]  OF1bus, OF2bus, OF3bus, OF4bus;

  MuxesF4bit_4 MF8_0( FXbus[3:0], FYbus[3:0], QF1bus[3:0], QF2bus[3:0],
                      QF3bus[3:0], QF4bus[3:0], MuxSelbus[8:0],
                      OF1bus[3:0], OF2bus[3:0], OF3bus[3:0], OF4bus[3:0] ),
  MF8_1( FXbus[7:4], FYbus[7:4], QF1bus[7:4], QF2bus[7:4],
         QF3bus[7:4], QF4bus[7:4], MuxSelbus[8:0],
         OF1bus[7:4], OF2bus[7:4], OF3bus[7:4], OF4bus[7:4] );
  Muxes4       MF8_2( FXbus[8], FYbus[8], QF1bus[8], QF2bus[8],
                      QF3bus[8], QF4bus[8], MuxSelbus[8:0],
                      OF1bus[8], OF2bus[8], OF3bus[8], OF4bus[8] );

endmodule // MuxesF8bit_4

/*****************************************/

module MuxesF4bit_4( FXbus, FYbus, QF1bus, QF2bus, QF3bus, QF4bus, MuxSelbus,
                     OF1bus, OF2bus, OF3bus, OF4bus );

  input [3:0]    FXbus, FYbus, QF1bus, QF2bus, QF3bus, QF4bus;
  input [8:0]    MuxSelbus;
  output [3:0]  OF1bus, OF2bus, OF3bus, OF4bus;
  Muxes4 MF4_0( FXbus[0], FYbus[0], QF1bus[0], QF2bus[0],
                QF3bus[0], QF4bus[0], MuxSelbus[8:0],
                OF1bus[0], OF2bus[0], OF3bus[0], OF4bus[0] ),
  MF4_1( FXbus[1], FYbus[1], QF1bus[1], QF2bus[1],
         QF3bus[1], QF4bus[1], MuxSelbus[8:0],
         OF1bus[1], OF2bus[1], OF3bus[1], OF4bus[1] ),
  MF4_2( FXbus[2], FYbus[2], QF1bus[2], QF2bus[2],
         QF3bus[2], QF4bus[2], MuxSelbus[8:0],
         OF1bus[2], OF2bus[2], OF3bus[2], OF4bus[2] ),
  MF8_3( FXbus[3], FYbus[3], QF1bus[3], QF2bus[3],
         QF3bus[3], QF4bus[3], MuxSelbus[8:0],
         OF1bus[3], OF2bus[3], OF3bus[3], OF4bus[3] );

endmodule // MuxesF4bit_4


/**********************************************************************
****
 * Module: ZeroFlags
 *
 * Function: generates the zero signal for four 9-bit buses:
 *   SumX, LogicX, SumY and LogicY.
 *   In each case, the zero signal is equal to the NOR of all the inputs.
 *
```

```
*************************************************************************
***/

module ZeroFlags( SumX, LogicX, SumY, LogicY, ZeroFlagOut );

  input [8:0]    SumX, LogicX, SumY, LogicY;
  output [3:0]  ZeroFlagOut;

  NOR9 ZF0( SumX, ZeroFlagOut[3] ),
  ZF1( SumY, ZeroFlagOut[2] ),
  ZF2( LogicX, ZeroFlagOut[1] ),
  ZF3( LogicY, ZeroFlagOut[0] );

endmodule // ZeroFlags

/*************************************************************************
****
 * Module: BusParityChk
 *
 * Function: computes the parity of four 10-bit buses:
 *  X0bus, Xbus, Y0bus and Ybus, each with an additional input.
 *  ParChkOut[0] is the AND of all the bus parities and can be masked
 *  by ContParChk inputs.
 *
```

```
**********************************************************************
***/

module BusParityChk( X0bus, Xbus, Y0bus, Ybus, ParXin, ParYin,
                     MuxSelX, MuxSelY, ContParChk, ParChkOut );

  input [8:0]    X0bus, Xbus, Y0bus, Ybus;
  input [1:0]    ParXin, ParYin;
  input MuxSelX, MuxSelY;
  input [5:0]    ContParChk;
  output [4:0]   ParChkOut;

  wire           ParX, ParY;
  wire [3:0]     NotParChk;

  Mux2_1 BPC0( ParXin[0], ParXin[1], MuxSelX, ParX ),
  BPC1( ParYin[0], ParYin[1], MuxSelY, ParY );

  ParityTree10bit BPC2( { ParX, Xbus[8:0] }, ParChkOut[4] ),
  BPC3( { ParXin[0], X0bus[8:0] }, ParChkOut[3] ),
  BPC4( { ParY, Ybus[8:0] }, ParChkOut[2] ),
  BPC5( { ParYin[0], Y0bus[8:0] }, ParChkOut[1] );

  Invert4  BPC6( ParChkOut[4:1], NotParChk );
  and5     BPC7( .A(NotParChk[3]), .B(NotParChk[2]), .C(NotParChk[1]),
                  .D(NotParChk[0]), .E(ContParChk[5]), .Y(line7) );
  and4     BPC8( .A(ContParChk[0]), .B(ContParChk[1]), .C(ContParChk[2]),
                  .D(ContParChk[3]), .Y(line8) );
  and3     BPC9( .A(line8), .B(line7), .C(ContParChk[4]),
                  .Y(ParChkOut[0]) );

endmodule // BusParityChk

/**********************************************************************
****
 * Module: MiscLogic
 *
 * Function: contains muxes and gates that are mostly unstructured
 *  and unrelated to the rest of the circuit.
 *
 *  - The MiscMuxLogic block includes four 2:1 and 4:1 muxes with
 *    independent inputs.
 *  - The MiscRandomLogic block contains mostly inverters and buffers.
 *

**********************************************************************
***/

module MiscLogic( MiscMuxIn, MiscContIn, MiscInbus, ContParChk,
                  Xbus_8, LogicXbus_8, SumXbus_8, WXbus_8,
                  X1bus3_0, X1bus_8, X0bus_8, MuxSelPF_8,
                  MiscMuxOut, MiscOutbus );

  input [16:0]   MiscMuxIn;
  input [7:0]    MiscContIn;
  input [8:0]    MiscInbus;
  input [5:0]    ContParChk;
  input  Xbus_8, LogicXbus_8, SumXbus_8, WXbus_8;
  input  X1bus_8, X0bus_8, MuxSelPF_8;
  input [3:0]    X1bus3_0;
  output [10:0] MiscMuxOut;
  output [25:0] MiscOutbus;

  wire           ContBeta;

  MiscMuxLogic UM13_0( { Xbus_8, LogicXbus_8, SumXbus_8, WXbus_8,
MiscMuxIn },
                         MiscContIn, ContBeta, MiscMuxOut );

  MiscRandomLogic UM13_1( { X1bus3_0, X1bus_8, X0bus_8, MuxSelPF_8,
MiscInbus },
                          ContParChk, MiscContIn, ContBeta, MiscOutbus );

endmodule // MiscLogic

/*******************************************/
```

```
module  MiscMuxLogic( NewMuxIn, MiscContIn, ContBeta, MiscMuxOut );

   input [20:0]   NewMuxIn;
   input [7:0]    MiscContIn;
   output         ContBeta;
   output [10:0] MiscMuxOut;

   wire [3:0]     tempOut1, tempOut2, tempOut3;

   and2 MML0( .A(MiscContIn[0]), .B(MiscContIn[1]), .Y(ContBeta) );
   inv  MML1( .A(ContBeta), .Y(NotContBeta) ),
   MML2( .A(MiscContIn[2]), .Y(NotContIn2) );

   Mux4bit_2_1 MML3( NewMuxIn[3:0], NewMuxIn[7:4], NotContIn2,
                     tempOut1 );
   Mux4bit_4_1 MML4( NewMuxIn[11:8], NewMuxIn[15:12], { 4'b1111 },
                     { 4'b1111 }, NotContBeta, MiscContIn[2],
                     tempOut2 );

   // MiscMuxOut[3:0] and MiscMuxOut[7:4]
   Mask_And4bit MML5( tempOut1, ContBeta, tempOut3 );
   Invert4      MML6( tempOut3, MiscMuxOut[3:0] );
   Mask_And4bit MML7( tempOut2, MiscContIn[3], MiscMuxOut[7:4] );

   // MiscMuxOut[8] -- out818
```

```
  inv     MML8( .A(NewMuxIn[20]), .Y(NotMuxIn20) );

  XOR2b  MML9( .A(NotMuxIn20), .B(NewMuxIn[16]), .Y(tempMuxin) );

  Mux4_1 MML10( NewMuxIn[19], tempMuxin, NewMuxIn[17], NewMuxIn[18],

                    MiscContIn[5], MiscContIn[4], tempMuxout );

  nand2    MML11( .A(MiscContIn[6]), .B(MiscContIn[7]), .Y(tempMuxcont) );

  and2    MML12( .A(tempMuxcont), .B(tempMuxout), .Y(MiscMuxOut[8]) );


  // MiscMuxOut[9] -- out813

  XOR2b  MML13( .A(tempMuxin), .B(NewMuxIn[18]), .Y(MiscMuxOut[9]) );


  // MiscMuxOut[10]=not(SumXbus[8]) -- out623

  inv     MML14( .A(NewMuxIn[18]), .Y(MiscMuxOut[10]) );


endmodule // MiscMuxLogic


/******************************************/


module MiscRandomLogic( NewMiscbus, ContParChk, MiscContIn, ContBeta,
MiscOutbus );


  input [15:0]   NewMiscbus;

  input [5:0]     ContParChk;

  input [7:0]     MiscContIn;

  input  ContBeta;

  output [25:0] MiscOutbus;
```

```
// NewMiscbus: { X1bus3_0, X1bus_8, X0bus_8, MuxSelPF_8, MiscInbus }
//              15-12      11       10        9           8-0

nand2   MRL0( .A(ContBeta), .B(NewMiscbus[0]), .Y(MiscOutbus[0]) );

inv    MRL1( .A(NewMiscbus[1]), .Y(NotMisc1) );
and2   MRL2( .A(NotMisc1), .B(MiscContIn[0]), .Y(line2) );
inv    MRL3( .A(line2), .Y(MiscOutbus[1]) );

and2   MRL4( .A(MiscContIn[3]), .B(NewMiscbus[2]), .Y(MiscOutbus[2]) );

nand2   MRL5( .A(NewMiscbus[3]), .B(NewMiscbus[4]), .Y(line6) );
inv    MRL6( .A(line6), .Y(MiscOutbus[3]) );

inv    MRL7( .A(NewMiscbus[6]), .Y(NotMisc6) );
and2   MRL8( .A(NewMiscbus[5]), .B(NotMisc6), .Y(MiscOutbus[4]) );

and2   MRL9( .A(ContParChk[0]), .B(ContParChk[2]), .Y(line12) );
inv    MRL10( .A(line12), .Y(MiscOutbus[5]) );

and2   MRL11( .A(ContParChk[3]), .B(ContParChk[5]), .Y(MiscOutbus[6]) );

Buffer7 MRL12( { NewMiscbus[11:9], NewMiscbus[7:6], NewMiscbus[4],
                 MiscContIn[3] }, MiscOutbus[13:7] );
```

```
  Invert4 MRL13( { ContParChk[5:3], ContParChk[1] }, MiscOutbus[17:14] );


  Invert4 MRL14( NewMiscbus[15:12], MiscOutbus[21:18] );


  Invert4 MRL15( { NewMiscbus[11], NewMiscbus[8:7], ContBeta },
                MiscOutbus[25:22] );

endmodule // MiscRandomLogic



/**********************************************************************
****
 * Description of some basic gates/modules

**********************************************************************
***/


/******************************************/


module ParityTree10bit( Inbus, ParOut );


  input [9:0] Inbus;
  output     ParOut;


  XOR2a PT0( .A(Inbus[5]), .B(Inbus[6]), .Y(line0) ),
  PT1( .A(Inbus[7]), .B(Inbus[8]), .Y(line1) ),
  PT2( .A(Inbus[0]), .B(Inbus[9]), .Y(line2) ),

  PT3( .A(Inbus[1]), .B(Inbus[2]), .Y(line3) ),

  PT4( .A(Inbus[3]), .B(Inbus[4]), .Y(line4) );

  XOR2a PT5( .A(line0), .B(line1), .Y(line5) );

  XOR3a PT6( .A(line2), .B(line3), .C(line4), .Y(line6) );

  XOR2a PT7( .A(line5), .B(line6), .Y(ParOut) );


endmodule // ParityTree10bit


/*******************************************/


module ParityTree9bit( Inbus, ParOut );


  input [8:0] Inbus;

  output      ParOut;


  XOR2a PT1( .A(Inbus[5]), .B(Inbus[6]), .Y(line1) ),

  PT2( .A(Inbus[7]), .B(Inbus[8]), .Y(line2) ),

  PT3( .A(Inbus[1]), .B(Inbus[2]), .Y(line3) ),

  PT4( .A(Inbus[3]), .B(Inbus[4]), .Y(line4) );

  XOR2a PT5( .A(line1), .B(line2), .Y(line5) );

  XOR3a PT6( .A(line3), .B(Inbus[0]), .C(line4), .Y(line6) );

  XOR2a PT7( .A(line5), .B(line6), .Y(ParOut) );


endmodule // ParityTree9bit
```

```
/*******************************************/

module Invert4( Inbus, Outbus );

  input [3:0]    Inbus;
  output [3:0]  Outbus;

  inv Inv4_0( .A(Inbus[0]), .Y(Outbus[0]) ),
  Inv4_1( .A(Inbus[1]), .Y(Outbus[1]) ),
  Inv4_2( .A(Inbus[2]), .Y(Outbus[2]) ),
  Inv4_3( .A(Inbus[3]), .Y(Outbus[3]) );

endmodule // Invert4

/*******************************************/

module Invert9( Inbus, Outbus );

  input [8:0]    Inbus;
  output [8:0]  Outbus;

  Invert4 Inv9_0( Inbus[3:0], Outbus[3:0] ),
  Inv9_1( Inbus[7:4], Outbus[7:4] );
  inv     Inv9_2( .A(Inbus[8]), .Y(Outbus[8]) );
```

```
endmodule // Invert9

/******************************************/

module Buffer7( Inbus, Outbus );

  input [6:0]    Inbus;
  output [6:0]  Outbus;

  buffer B7_0( .A(Inbus[0]), .Y(Outbus[0]) ),
  B7_1( .A(Inbus[1]), .Y(Outbus[1]) ),
  B7_2( .A(Inbus[2]), .Y(Outbus[2]) ),
  B7_3( .A(Inbus[3]), .Y(Outbus[3]) ),
  B7_4( .A(Inbus[4]), .Y(Outbus[4]) ),
  B7_5( .A(Inbus[5]), .Y(Outbus[5]) ),
  B7_6( .A(Inbus[6]), .Y(Outbus[6]) );

endmodule // Buffer7

/******************************************/

module XOR2a6bit( In1bus, In2bus, Outbus );

  input [5:0]    In1bus, In2bus;
  output [5:0]  Outbus;


  XOR2a X2a6_0( .A(In1bus[0]), .B(In2bus[0]), .Y(Outbus[0]) ),

  X2a6_1( .A(In1bus[1]), .B(In2bus[1]), .Y(Outbus[1]) ),

  X2a6_2( .A(In1bus[2]), .B(In2bus[2]), .Y(Outbus[2]) ),

  X2a6_3( .A(In1bus[3]), .B(In2bus[3]), .Y(Outbus[3]) ),

  X2a6_4( .A(In1bus[4]), .B(In2bus[4]), .Y(Outbus[4]) ),

  X2a6_5( .A(In1bus[5]), .B(In2bus[5]), .Y(Outbus[5]) );


endmodule // XOR2a6bit


/******************************************/


module Mux4_1( In0, In1, In2, In3, ContHi, ContLo, Out );


  input  In0, In1, In2, In3, ContHi, ContLo;

  output Out;


  inv  Mux4_0( .A(ContLo), .Y(Not_ContLo) ),

  Mux4_1( .A(ContHi), .Y(Not_ContHi) );

  and3 Mux4_2( .A(In0), .B(Not_ContHi), .C(Not_ContLo), .Y(line2) ),

  Mux4_3( .A(In1), .B(Not_ContHi), .C(ContLo), .Y(line3) ),

  Mux4_4( .A(In2), .B(ContHi), .C(Not_ContLo), .Y(line4) ),

  Mux4_5( .A(In3), .B(ContHi), .C(ContLo), .Y(line5) );

  or4 Mux4_6( .A(line2), .B(line3), .C(line4), .D(line5), .Y(Out) );
```

```
endmodule // Mux4_1

/******************************************/

module Mux2_1( In0, In1, ContIn, Out );

   input  In0, In1, ContIn;
   output Out;

   inv  Mux2_0( .A(ContIn), .Y(Not_ContIn) );
   and2 Mux2_1( .A(In0), .B(Not_ContIn), .Y(line1) ),
   Mux2_2( .A(In1), .B(ContIn), .Y(line2) );
   or2 Mux2_3( .A(line1), .B(line2), .Y(Out) );

endmodule // Mux2_1

/******************************************/

module Mux9bit_4_1( In1bus, In2bus, In3bus, In4bus,
                    ContHi, ContLo, Outbus );

   input [8:0]    In1bus, In2bus, In3bus, In4bus;
   input ContHi, ContLo;
   output [8:0]  Outbus;
   Mux4bit_4_1 Mx9_0( In1bus[3:0], In2bus[3:0], In3bus[3:0], In4bus[3:0],
                      ContHi, ContLo, Outbus[3:0] ),
   Mx9_1( In1bus[7:4], In2bus[7:4], In3bus[7:4], In4bus[7:4],
          ContHi, ContLo, Outbus[7:4] );
   Mux4_1     Mx9_2( In1bus[8], In2bus[8], In3bus[8], In4bus[8],
                      ContHi, ContLo, Outbus[8] );

endmodule // Mux9bit_4_1

/*****************************************/

module Mux4bit_4_1( In1bus, In2bus, In3bus, In4bus,
                    ContHi, ContLo, Outbus );

   input [3:0]    In1bus, In2bus, In3bus, In4bus;
   input ContHi, ContLo;
   output [3:0]  Outbus;

   Mux4_1 Mx4_0( In1bus[0], In2bus[0], In3bus[0], In4bus[0],
                 ContHi, ContLo, Outbus[0] ),
   Mx4_1( In1bus[1], In2bus[1], In3bus[1], In4bus[1],
          ContHi, ContLo, Outbus[1] ),
   Mx4_2( In1bus[2], In2bus[2], In3bus[2], In4bus[2],
          ContHi, ContLo, Outbus[2] ),
   Mx4_3( In1bus[3], In2bus[3], In3bus[3], In4bus[3],
          ContHi, ContLo, Outbus[3] );

endmodule // Mux4bit_4_1

/****************************************************/

module Mask_And4bit( Inbus, Mask, Outbus );

   input [3:0]   Inbus;
   input Mask;
   output [3:0]  Outbus;

   and2 Ma0( .A(Inbus[0]), .B(Mask), .Y(Outbus[0]) ),
   Ma1( .A(Inbus[1]), .B(Mask), .Y(Outbus[1]) ),
   Ma2( .A(Inbus[2]), .B(Mask), .Y(Outbus[2]) ),
   Ma3( .A(Inbus[3]), .B(Mask), .Y(Outbus[3]) );

endmodule // AND4bit

/****************************************************/

module AND_OR2( O, P, Q, YY);

   input  O, P, Q;
   output YY;

   and2 Ao2_0( .A(P), .B(Q), .Y(line0) );
   or2  Ao2_1( .A(O), .B(line0), .Y(YY) );

endmodule // AND_OR2

/****************************************************/

module AND_OR3a( O, P, Q, R, S, YY);

   input  O, P, Q, R, S;
   output YY;

   and2 Ao3a_0( .A(P), .B(Q), .Y(line0) );
   and3 Ao3a_1( .A(P), .B(R), .C(S), .Y(line1) );
   or3  Ao3a_2( .A(O), .B(line0), .C(line1), .Y(YY) );

endmodule // AND_OR3a

/****************************************************/

module AND_OR3b( O, P, Q, R, YY);

   input  O, P, Q, R;
```

```
   output YY;


   and2 Ao3a_0( .A(P), .B(Q), .Y(line0) );
   and2 Ao3a_1( .A(P), .B(R), .Y(line1) );
   or3  Ao3a_2( .A(O), .B(line0), .C(line1), .Y(YY) );

endmodule // AND_OR3b

/****************************************************/

module AND_OR4a( O, P, Q, R, S, T, U, YY);

   input  O, P, Q, R, S, T, U;
   output YY;


   and2 Ao4a_0( .A(P), .B(Q), .Y(line0) );
   and3 Ao4a_1( .A(P), .B(R), .C(S), .Y(line1) );
   and4 Ao4a_2( .A(P), .B(R), .C(T), .D(U), .Y(line2) );
   or4  Ao4a_3( .A(O), .B(line0), .C(line1), .D(line2), .Y(YY) );

endmodule // AND_OR4a

/****************************************************/

module AND_OR4b( O, P, Q, R, S, T, YY);
```

```
  input  O, P, Q, R, S, T;
  output YY;

  and2 Ao4a_0( .A(P), .B(Q), .Y(line0) );
  and3 Ao4a_1( .A(P), .B(R), .C(S), .Y(line1) );
  and3 Ao4a_2( .A(P), .B(R), .C(T), .Y(line2) );
  or4  Ao4a_3( .A(O), .B(line0), .C(line1), .D(line2), .Y(YY) );

endmodule // AND_OR4a

/****************************************************/

module AND_OR5a( O, P, Q, R, S, T, U, V, W, YY);

  input  O, P, Q, R, S, T, U, V, W;
  output YY;

  and2 Ao5a_0( .A(P), .B(Q), .Y(line0) );
  and3 Ao5a_1( .A(P), .B(R), .C(S), .Y(line1) );
  and4 Ao5a_2( .A(P), .B(R), .C(T), .D(U), .Y(line2) );
  and5 Ao5a_3( .A(P), .B(R), .C(T), .D(V), .E(W), .Y(line3) );
  or5  Ao5a_4( .A(O), .B(line0), .C(line1), .D(line2), .E(line3), .Y(YY) );

endmodule // AND_OR5a
```

```
/*****************************************************/

module AND_OR5b( O, P, Q, R, S, T, U, V, YY);

  input  O, P, Q, R, S, T, U, V;
  output YY;

  and2 Ao5a_0( .A(P), .B(Q), .Y(line0) );
  and3 Ao5a_1( .A(P), .B(R), .C(S), .Y(line1) );
  and4 Ao5a_2( .A(P), .B(R), .C(T), .D(U), .Y(line2) );
  and4 Ao5a_3( .A(P), .B(R), .C(T), .D(V), .Y(line3) );
  or5  Ao5a_4( .A(O), .B(line0), .C(line1), .D(line2), .E(line3), .Y(YY) );

endmodule // AND_OR5b

/*****************************************************/

module AND_OR6a( O, P, Q, R, S, T, U, V, W, X, Y, YY);

  input  O, P, Q, R, S, T, U, V, W, X, Y;
  output YY;

  and2 Ao6a_0( .A(P), .B(Q), .Y(line0) );
  and3 Ao6a_1( .A(P), .B(R), .C(S), .Y(line1) );
```

```
   and4 Ao6a_2( .A(P), .B(R), .C(T), .D(U), .Y(line2) );
   and5 Ao6a_3( .A(P), .B(R), .C(T), .D(V), .E(W), .Y(line3) );
   and6 Ao6a_4( .A(P), .B(R), .C(T), .D(V), .E(X), .F(Y), .Y(line4) );
   or6  Ao6a_5( .A(O), .B(line0), .C(line1), .D(line2), .E(line3),
                 .F(line4), .Y(YY) );

endmodule // AND_OR6a

/****************************************************/

module AND_OR6b( O, P, Q, R, S, T, U, V, W, X, YY);

   input  O, P, Q, R, S, T, U, V, W, X;
   output YY;

   and2 Ao6a_0( .A(P), .B(Q), .Y(line0) );
   and3 Ao6a_1( .A(P), .B(R), .C(S), .Y(line1) );
   and4 Ao6a_2( .A(P), .B(R), .C(T), .D(U), .Y(line2) );
   and5 Ao6a_3( .A(P), .B(R), .C(T), .D(V), .E(W), .Y(line3) );
   and5 Ao6a_4( .A(P), .B(R), .C(T), .D(V), .E(X), .Y(line4) );
   or6  Ao6a_5( .A(O), .B(line0), .C(line1), .D(line2), .E(line3),
                 .F(line4), .Y(YY) );

endmodule // AND_OR6b
```

```
/*****************************************************/

module XOR2a ( A, B, Y );

   input  A, B;
   output Y;

   inv   Xo0( .A(A), .Y(NotA) ),
   Xo1( .A(B), .Y(NotB) );

   nand2 Xo2( .A(NotA), .B(B), .Y(line2) ),
   Xo3( .A(NotB), .B(A), .Y(line3) ),
   Xo4( .A(line2), .B(line3), .Y(Y) );

endmodule // XOR2a

/*****************************************************/

module XOR2b ( A, B, Y );

   input  A, B;
   output Y;

   nand2 Xo0( .A(A), .B(B), .Y(NotAB) );
   and2  Xo1( .A(A), .B(NotAB), .Y(line1) ),
   Xo2( .A(NotAB), .B(B), .Y(line2) );
   or2   Xo3( .A(line1), .B(line2), .Y(Y) );

endmodule // XOR2b

/*******************************************/

module XOR3a( A, B, C, Y);

   input  A, B, C;
   output Y;

   inv   Xo3_0( .A(A), .Y(NotA) ),
   Xo3_1( .A(B), .Y(NotB) ),
   Xo3_2( .A(C), .Y(NotC) );
   and3  Xo3_3( .A(NotA), .B(NotB), .C(C), .Y(line3) ),
   Xo3_4( .A(NotA), .B(B), .C(NotC), .Y(line4) ),
   Xo3_5( .A(A), .B(NotB), .C(NotC), .Y(line5) ),
   Xo3_6( .A(A), .B(B), .C(C), .Y(line6) );
   nor2  Xo3_7( .A(line3), .B(line4), .Y(line7) ),
   Xo3_8( .A(line5), .B(line6), .Y(line8) );
   nand2 Xo3_9( .A(line7), .B(line8), .Y(Y) );

endmodule // XOR3a
```

```
/*******************************************/

module NOR9(In, Out);

  input [8:0] In;
  output      Out;

  nor9 n9(.A(In[0]), .B(In[1]), .C(In[2]), .D(In[3]), .E(In[4]), .F(In[5]),
          .G(In[6]), .H(In[7]), .I(In[8]), .Y(Out) );

endmodule // NOR9
```

**RTL Development**

We use a hybrid C++/Verilog simulation approach for the Scale RTL. After implementing the RTL for a block of the design, we use Tenison VTOC to translate the Verilog into a C++ module with input and output ports and a clock-tick evaluation method. We then wrap this module with the necessary glue logic to connect it to the C++ microarchitectural simulator. Using this methodology we are able to avoid constructing custom Verilog test harnesses to drive each block as we develop the RTL. Instead, we leverage our existing set of test programs as well as our software infrastructure for easily compiling and running directed test programs. This design approach allowed us to progressively expand the RTL code base from the starting point of a single cluster, to a single lane, to four lanes; and then to add the AIB fill unit, the vector memory unit, the control processor, and the memory system.

**Datapath Pre-Placement**

We used a C++-based procedural datapath tiler which manipulates standard cells and creates design databases using the Open Access libraries. After constructing a datapath, we export a Verilog netlist together with a DEF file with relative placement information.

We incorporate datapath pre-placement into our CAD tool flow by separating out the datapath modules in the source RTL; for example, the cluster datapaths for Scale include the ALU, shifter, and many 32-bit muxes and latches. We then write tiler code to construct these datapaths and generate cell netlists. During synthesis we provide these netlists in place of the source RTL for the datapath modules, and we flag the pre-placed cells as *dont touch*. In this way, Design Compiler can correctly optimize logic which interfaces with the datapath blocks. During the floorplanning step before place-and-route, we use scripts to flexibly position each datapath wherever we want on the chip. These scripts process the relative placement information in the datapath DEF files, combining these into a unified DEF file with absolute placement locations. We again use *dont touch* to prevent Encounter from modifying the datapaths cells during placement and optimization. We use Encounter to do the datapath routing automatically; this avoids the additional effort of routing by hand, and we have found that the tool does a reasonable job after the datapath arrays have been pre-placed.

As a simple example of the ease with which we can create pre-placed datapath arrays, Figure 30(a) shows a small snippet of Verilog RTL from Scale which connects a 32-bit mux with a 32-bit latch. Figure 30(b) shows the corresponding C++ code which creates the pre-placed datapath diagrammed in Figure 30(c). The placement code is simple and very similar to the RTL, the only extra information is the output drive strength of each component. The supporting component builder libraries (*dpMux2 and dpLatch h en*) each add a column of cells to the virtual grid in the tiler (tl). By default, the components are placed from left to right. In this example, the *dpMux2* builder creates each two-input multiplexer using three NAND gates. The component builders also add the necessary clock gating and driver cells on top of the datapath, and the code automatically sets the size of these based on the bit-width of the datapath. We used our datapath pre-placement infrastructure to create parameterizable builders for com- ponents like muxes, latches, queues, adders, and shifters. It is relatively straightforward to assemble these components into datapaths, and easy to modify the datapaths as necessary. In the end, we pre-placed 230 thousand cells, 58% of all standard cells in the Scale chip.

```
wire [31:0] xw_sd_pn;
wire [31:0] w_sd_np;

dpMux2 #(32)
  xw_sd_mux
    ( .s    (xw_sdsel_pn),
      .i0   (xw_reg_pn),
      .i1   (xw_sdc_pn),
      .o    (xw_sd_pn));

dpLatch_h_en #(32)
  w_sd_hl
    ( .clk  (clk),
      .d_n  (xw_sd_pn),
      .en_p (x_sden_np),
      .q_np (w_sd_np));
```

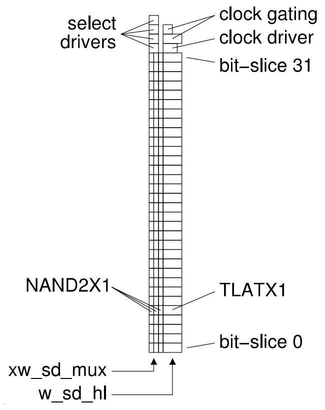
(a) Verilog RTL

```
Tiler tl;
tl.addNet  ( "xw_sd_pn", 32 );
tl.addNet  ( "w_sd_np", 32 );

dpMux2 ( tl, Drive::X1, 32,
    "xw_sd_mux",
    "xw_sdsel_pn",
    "xw_reg_pn",
    "xw_sdc_pn",
    "xw_sd_pn");

dpLatch_h_en ( tl, Drive::X1, 32,
    "w_sd_hl",
    "clk",
    "xw_sd_pn",
    "x_sden_np",
    "w_sd_np");
```

(b) C++ pre-placement code



(c) Datapath cells

Figure 30: Datapath pre-placement code example.

## REFERENCE

[1] C. Chiang, J. Kawa, *Design for Manufacturability and Yield for Nano-scale CMOS*. (Springer, Dordrecht, 2007), pp. 14-15.

[2] W. Zhao, Y. Cao, F. Liu, K. Agarwal, D. Acharyya, S. Nassif, K. Nowka, "Rigorous extraction of process variations for 65 nm CMOS design," in *Proceedings of European Solid-State Device Research Conference (ESSDERC)*, pp. 89-92, Sept 2007.

[3] G.W. Roberts, B. Dufort, "Making complex mixed-signal telecommunication integrated circuits testable," in *IEEE Communication Magazine*, pp. 90-96 (1999).

[4] A. Zjajo, J.P. de Gyvez, "Evaluation of signature-based testing of RF/analog circuits," in Proceedings *of European Test Symposium*, pp. 62-67 (2005).

[5] C. Visweswariah, "Death, taxes and failing chips," In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 343-347, Anaheim, California, USA, June 2003.

[6] P. S. Zuchowski, P. A. Habitz, J. D. Hayes, and J. H. Oppold, "Process and environmental variations impacts on ASIC timing," In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 336-342, San Jose, California, USA, November 2004.

[7] H. F. Jyu, S. Malik, S. Devadas, and K. W. Keutzer, "Statistical timing analysis of combinational logic circuits," in *IEEE Transactions on Very Large Scale Integration Systems*, vol. 1, no 2, pp. 126-137, June 1993.

[8] A. Papoulis and S. U. Pillai, *Probability, random variables, and stochastic processes*. McGraw-Hill, Boston, USA, 2002.

[9] C. P. Robert and G. Casella, *Monte Carlo statistical methods*. Springer, New York, NY, USA, 1999.

[10] Abu Baker, Yingtao Jiang, "Modeling and Architectural Simulations of the Statistical Static Timing Analysis on the non-Gaussian Variation Sources for VLSI Circuits," in *International Journal of Scientific and Research Publications*, vol. 3, no 1, January 2013.

[11] K. Wakabayashi and T. Okamoto, "C-based SoC design flow and EDA tools: An ASIC and system vendor perspective," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1507-1522, Dec. 2000.

[12] S. Das, S. Pant, D. Roberts, and S. Seokwoo, "A self-tuning DVS processor using delay-error detection and correction," in *Proc. IEEE Symp. VLSI Circuits*, pp. 258-

261, 2005.

[13] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De, “Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage,” in *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1396-1402, Nov. 2002.

[14] G. Nanz and L. Camilletti, “Modeling of chemical-mechanical polishing: A review,” in *IEEE Trans. Semicond. Manuf.*, vol. 8, no. 4, pp. 382-389, Nov. 1995.

[15] C. Mack, “Understanding focus effects in submicrometer optical lithography: A review,” *Opt. Eng.*, vol. 32, no. 10, pp. 2350-2362, Oct. 1993.

[16] L. Scheffer, “Physical CAD changes to incorporate design for lithography and manufacturability,” in *Proc. ASP-DAC*, pp. 768-773, 2004.

[17] F. Huebbers, A. Dasdan, and Y. Ismail, “Computation of accurate interconnect process parameter values for performance corners under process variations,” in *Proc. DAC*, pp. 797-800, 2006.

[18] J. Yang, L. Capodieci, and D. Sylvester, “Advanced timing analysis based on post-OPC extraction of critical dimensions,” in *Proc. DAC*, pp. 359-364, 2005.

[19] P. Gupta and F. Heng, “Toward a systematic-variation aware timing methodology,” in *Proc. DAC*, pp. 321-326, 2004.

[20] A. Agarwal, D. Blaauw, and V. Zolotov, “Statistical timing analysis for intra-die process variations with spatial correlations,” In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pp. 900-907, San Jose, California, USA, November 2003.

[21] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R Panda, “Statistical delay computation considering spatial correlations,” In *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 271-276, Kitakyushu, Japan, January 2003.

[22] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula, “Computation and refinement of statistical bounds on circuit delay,” In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 348-353, Anaheim, California, USA, June 2003.

[23] M. Berkelaar, *Statistical delay calculation, a linear time method*. 1997. (Personal communication).

[24] S. Bhardwaj, S. B. K. Vrudhula, and D. Blaauw, “τ AU: Timing analysis under uncertainty,” In *Proceedings of the ACM/IEEE International Conference on*

*Computer Aided Design*, pp. 615-620, San Jose, California, USA, November 2003.

[25] R. B. Brawhear, N. Menezes, C. Oh, L. Pillage, and R. Mercer, "Predicting circuit performance using circuit-level statistical timing analysis," In *Proceedings of European Design and Test Conference*, pp. 332-337, Paris, France, 1994.

[26] B. Choi and D. M. H. Walker, "Timing analysis of combinational circuits including capacitive coupling and statistical process variation," *In Proceedings of the IEEE VLSI Test Symposium*, pp. 49-54, Montreal, Canada, April 2000.

[27] C. E. Clark, "The greatest of a finite set of random variables," *Operations Research*, vol. 9, pp. 145-162, March-April 1961.

[28] Y. Deguchi, N. Ishiura, and S. Yajima, "Probabilistic CTSS: Analysis of timing error probability in asynchronous logic circuits," In *Proceedings of IEEE/ACM Design Automation Conference*, pp. 650-655, San Francisco, CA, USA, 1991.

[29] A. Devgan and C. Kashyap, "Block-based static timing analysis with uncertainty," In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pp. 607-614, San Jose, California, USA, November 2003.

[30] A. Gattiker, S. Nassif, R. Dinakar, and C. Long, "Timing yield estimation from static timing analysis," In *Proceedings of the International Symposium on Quality Electronic Design*, pp. 437-442, San Jose, CA, 2001.

[31] E. T. A. F. Jacobs and M. Berkelaar, "Gate sizing using a statistical delay model," In *Proceedings of Design Automation and Test in Europe*, pp. 283-290, Paris, France, March 2000.

[32] J. A. G. Jess, K. Kalafala, S. R. Naidu, R. H. J. Otten, and C. Visweswariah, "Statistical timing for parametric yield prediction of digital integrated circuits," In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 932-937, Anaheim, California, USA, June 2003.

[33] V. Khandelwal, A. Davoodi, and A. Srivastava, "Efficient statistical timing analysis through error budgeting," In *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, pp. 473-477, San Jose, CA, 2004.

[34] R. B. Lin and M. C. Wu, "A new statistical approach to timing analysis of VLSI circuits," In *Proceedings of International Conference on VLSI Design*, pp. 507-513, Chennai, India, 1998.

[35] J. J. Liou, K. T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 661-666, Las Vegas, Nevada, USA, June 2001.

[36] J. J. Liou, A. Krstic, L. C. Wang, and K. T. Cheng, "False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation," In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 566-569, New Orleans, Louisiana, USA, June 2002.

[37] D. F. Morrison, *Multivariate Statistical Methods*. McGraw-Hill, New York, NY, USA, 1976.

[38] S. Naidu, "Timing yield calculation using an impulse-train approach," In *Proceedings of the 15th International Conference on VLSI Design*, pp. 219-224, Bangalore, India, January 2002.

[39] S. R. Nassif, "Design for variability in DSM technologies," In *Proceedings of the IEEE International Symposium on Quality of Electronic Design*, pp. 451-454, San Jose, California, USA, March 2000.

[40] M. Orshansky and K. Keutzer, "A general probabilistic framework for worst case timing analysis," In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 556-561, New Orleans, Louisiana, USA, June 2002.

[41] M. Orshansky, L. Milor, P. Chen, K. Keutzer, and C. Hu, "Impact of spatial intrachip gate length variability on the performance of high-speed digital circuits," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 5, pp. 544-553, May 2002.

[42] S. S. Sapatnekar, *Timing*. Kluwer Academic Publishers, Boston, MA, 2004.

[43] S. Tsukiyama, M. Tanaka, and M. Fukui, "A statistical static timing analysis considering correlations between delays," In *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 353-358, Yokohama, Japan,January 2001.

[44] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 331-336, San Diego, California, USA, June 2004.

[45] K. Wakabayashi and T. Okamoto, "C-based SoC design flow and EDA tools: An ASIC and system vendor perspective," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1507-1522, Dec. 2000.

[46] D. Gajski, N. Dutt, and A., *Wu, High-level synthesis: Introduction to chip and system design*. Kluwer Academic Publishers, 1992.

[47] D. Sinha, N. Shenoy, and H. Zhou, "Statistical gate sizing for timing yield optimization," In *Proc. ICCAD*, pp. 1037-1041, 2005.

[48] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah, “Parameterized block-based statistical timing analysis with non-Gaussian parameters, nonlinear delay functions,” In *Proc. Design Automation Conf.*, June 2005, pp. 71-76.

[49] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu, “New paradigm of predictive MOSFET and interconnect modeling for early circuit design,” In *Proc. Custom Integrated Circuits Conf.*, pp.201-204, May 2000.

[50] Z. Liang, M. Ikeda, and K. Asada, “Analysis of noise margins due to device parameter variations in sub-100nm CMOS technology,” In *Proc. Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 1-6, April 2007.

[51] T. Kirkpatrick and N. Clark, “PERT as an aid to logic design,” in *IBM J.Res. Develop.*, vol. 10, no. 2, pp. 135-141, Mar. 1966.

[52] C. Visweswariah, K. Ravindran, and K. Kalafala, “First-order parameterized block-based statistical timing analysis,” in *TAU’04*, Feb 2004.

[53] H. Chang and S. S. Sapatnekar, “Statistical timing analysis considering spatial correlations using a single pert-like traversal,” in *ICCAD’03*, pp. 621-625, Nov 2003.

[54] A. Agarwal, V. Zolotov, and D. Blaauw, “Statistical timing analysis using bounds and selective enumeration,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 9, pp. 1243-1260, Sept 2003.

[55] A. Devgan and C. Kashyap, “Block-based static timing analysis with uncertainty,” in *ICCAD’03*, pp. 607-614, Nov 2003.

[56] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, “Statistical timing anlaysis: From basic principles to state of the art,” in *IEEE Transactions on Computeraided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 589-607, April 2008.

[57] J. Liu, J. Zeng, A. Hong, “Process Variation Statistical Modeling for VLSI Timing Analysis,” in the *9th International Symposium on Quality Electronic Design*, pp. 730-733, 2008.

[58] Goel, A., Vrudhula, S., Taraporevala, F., Ghanta, P. , “A Methodology for Characterization of Large Macro Cells and IP Blocks Considering Process Variations,” in *Proc. of 9th International Symposium on Quality Electronic Design*, pp. 200-206, March, 2008.

[59] Sinha, D., Bhanji, A., Visweswariah, C., “A hierarchical transistor and gate level statistical timing analysis flow for microprocessor designs,” in *Proc. Design Automation Conference*, July, 2009.

[60] Sundareswaran, S., Abraham, A., Panda, R., and Ardelea, "A. Characterization of standard cells for intra-cell mismatch variations," in *IEEE Transactions on Semiconductor Manufacturing*, vol. 22, no 1, pp. 40-49, Feb. 2009.

[61] Amin, C. S., Menezes, N., Killpack, K., Dartu, F., Choudhury, U., Hakim, N., Ismail, Y. I., "Statistical static timing analysis: how simple can we get?," in *Proc. of the 42nd annual Design Automation Conference*, pp. 652-657, 2005.

[62] Arvind N.V., Somayaji, A., Mishra, A., Mandal, A., Hariprasad T.T., Sandeep, P., Verkinderen, n., Colin, D., "Variation aware analysis using Primetime-VX," in *Proc. of Synopsys User Group*, San Jose, 2009.

[63] Chang, H., and Sapatnekar, S., "Statistical timing analysis under spatial correlation," in *IEEE Trans. Computer Aided Design of Integrated Circuit and Systems*, vol. 24, no 9, Sept. 2005, pp. 1467-1482.

[64] Li, Z., Lu, X., and Shi, W., "Process variation dimension reduction based on SVD," in *Proc. of the International Symposium on Circuits and Systems*, vol. 4, pp. 672-675, May 2003.

[65] Newmark, D., Sharma, M., Spector, J., Schreiber, R., Zhan, Y., "A methodology for transistor level static timing analysis of a l2 cache on a high performance microprocessor," in *Austin Conference on Integrated Systems & Circuits*, 2006.

[66] Nitta, I., Shibuya, T., Homma, K., "Statistical timing analysis technology," in *Fujitsu Sc. Tech. J*, vol. 43, no 4, pp. 516-523, Nov. 2007.

[67] C. Ababei and K. Bazargan, "Timing minimization by statistical timing hMetis-based partitioning," in *Proceedings of International Conference on VLSI Design*, pp. 58-63, New Delhi, India, 2003.

[68] M. R. C. M. Berkelaar and J. A. G. Jess, "Gate sizing in MOS digital circuits with linear programming," in *Proceedings of European Design Automation Con ference*, pp. 217-221, Glasgow, Scotland, March 1990.

[69] S. H. Choi, B. C. Paul, and K. Roy, "Novel sizing algorithm for yield improvement under process variation in nanometer technology," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 454-459, San Diego, California, USA, June 2004.

[70] L. Deng and M. D. F. Wong, "Buffer insertion under process variations for delay minimization," in *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, pp. 317-321, San Jose, California, USA, 2005.

[71] P. Gupta and A. B. Khang, "Manufacturing-aware physical design," in *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, pp. 681-

687, San Jose, CA, USA, 2003.

[72] M. Hashimoto and H. Onodera, “A performance optimization method by gate sizing using statistical static timing analysis,” in *Proceedings of International Symposium on Physical Design*, pp. 111-116, San Diego, California, USA, April 2000.

[73] B. Lu, J. Hu, G. Ellis, and H. Su, “Process variation aware clock tree routing,” in *Proceedings of the ACM International Symposium on Physical Design*, pp. 174-181, Monterey, CA, USA, April 2003.

[74] M. Mani, A. Devgan, and M. Orshansky, “An effcient algorithm for statistical minimization of total power under timing yield constraints,” in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 309-314, Anaheim, CA, USA, 2005.

[75] S. Raj, S. B. K. Vrudhula, and J. Wang, “A methodology to improve timing yield in the presence of process variations,” in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 448-453, San Diego, California, USA, June 2004.

[76] A. K. Singh, M. Mani, and M. Orshansky, “Statistical technology mapping for parametric yield,” in *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, pp. 511-518, San Jose, CA, USA, 2005.

[77] A. Srivastava, D. Sylvester, and D. Blaauw, “Statistical optimization of leakage power considering process variations using dual-Vth and sizing,” in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 773-778, San Diego, California, USA, June 2004.

# CV

**Abu M Baker**

## Degrees

Ph.D. candidate in Electrical Engineering, University of Nevada
M. S. in Electrical Engineering, Oklahoma State University
B. Sc. in Electrical and Electronic Engineering, BUET, Dhaka, Bangladesh.

## Educational Achievements, Activities and Service

- Third Place Winner in the 9th Annual Student Contest from IEEE Computer Society Student Chapter at University of Louisiana.
- Honor Student recognized by the University of Louisiana Honors Convocation Committee.
- Outstanding Student Award from Honor Society "PHI KAPPA PHI".
- Research Assistantship Award from The Center for Advanced Computer Studies.
- Travel Award for IEEE CAMPS'06.
- Travel Award for IEEE MWSCAS'05.
- Teaching Assistantship Award from Texas A&M University System.
- Outstanding Leadership Award from Honor Society "ALPHA THETA CHI".
- Research Assistantship Award from University of South Alabama.
- Travel Award for IEEE NCTT'03.
- Research Fellowship Award from University Technology Malaysia.
- IEEE Student Member.
- Technical reviewer for the following conferences: CAMPS'06, SiPS'05, ISCAS'05, ISACAS'06, ISCAS'07, MWSCAS'05.
- Technical reviewer for the following Journal: IEEE CAS-I.
- Graduate Adviser for Continuing Students in The Center for Advanced Computer Studies.

## Work Experience

**Microsoft Corporation,** Redmond, Washington, April 2012 - Present
*SDET Engineer (Surface Design and Engineering)*

**Aruba Networks**, Sunnyvale, California, May 2008 – March 2012
*Member of Technical Staff*

## Publications

### *Journal Papers*

- ***Abu Baker***, Yingtao Jiang, "Modeling and Architectural Simulations of the Statistical Static Timing Analysis on the non-Gaussian Variation Sources for VLSI Circuits", *International Journal of Scientific and Research Publications*, Volume 3, Issue 1, January 2013.

- ***Abu Baker***, M. S. Alam et al, “Electromagnetic Compatibility Analysis in Buildings Affected by Lightning Strike”, *An International Journal in Generation, Transmission, Distribution and Utilization of Electric Power*, Elsevier Science, Vol. 73, Issue 2, February 2005.

***Conference Papers***

- ***Abu Baker***, Yingtao Jiang, “High Level Circuit Synthesis with System Level Statistical Static Timing Analysis Under Process Variation”, *IEEE Midwest Circuits and Systems*, Cincinnati, Columbus, August 2013.
- Ming Zhu, ***Abu Baker***, Yingtao Jiang, “On a Parallel Decimal Multiplier based on Hybrid 8421-5421 BCD Recoding”, *IEEE Midwest Circuits and Systems*, Cincinnati, Columbus, August 2013
- ***Abu Baker***, Magdy Bayoumi et al, “Design and Realization of Analog Phi-Function for LDPC Decoder”, *IEEE International Symposium on Circuits and Systems*, New Orleans, Louisiana, May 2007.
- ***Abu Baker,*** Ashok Kumar, Magdy Bayoumi et al, "Multisensor Data Fusion Schemes for Wireless Sensor Networks," *International Workshop on Computer Architecture for Machine Perception and Sensing*, Montreal, Canada, September, 2006.
- ***Abu Baker,*** Sajedur Rahman, Magdy Bayoumi, “Tunable Level-Shifter / Buffer for Dual supply systems and Low-Power clock-tree Design in Deep-Submicron Application”, *IEEE APACE'05*, Johor Bharu, Malaysia, December 2005.
- ***Abu Baker***, Magdy Bayoumi et al, “A low Power VLSI Paradigm for Iterative Decoders”, *IEEE Signal Processing Systems*, Greece, Athens, November, 2005.
- ***Abu Baker***, Magdy Bayoumi et al, “Wronskian Change Detector Architecture for Automated Visual Surveillance”, *IEEE Signal Processing Systems*, Greece, Athens, November 2005.
- ***Abu Baker***, Magdy Bayoumi et al, “A Fast, On-the-Fly Realization of Wronskin Change Detector in VLSI”, *IEEE Midwest Circuits and Systems*, Cincinnati, Ohio, August 2005.

**Dissertation Title**

MAX operation in Statistical Static Timing Analysis on the non-Gaussian Variation Sources for VLSI Circuits

**Thesis Examination Committee**

Chair, Yingtao Jiang, Ph.D
Committee member, Biswajit Das, Ph.D
Committee member, Mei Yang, Ph.D
Graduate College Faculty Representative, Hui Zhao, Ph.D