Architecture and CAD Techniques for Optimizing FPGAs and Reliability of Integrated Circuits

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

Satish Barghav Sivaswamy

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

Dr. Kia Bazargan, Advisor
September, 2009

Acknowledgments

I am deeply indebted to my advisor, Prof. Kia Bazargan, for taking me under his wing and guiding me through my graduate study. Without him, I would never have been able to fulfill my dream of completing my PhD. He has been a pillar of support through various ups and downs during my PhD and for that, I will forever be grateful. I would like to thank Prof. Marc Riedel for the various discussions we had when I was working on circuit reliability. I would also like to thank Prof. Sachin Sapatnekar for introducing me to several new research problems in the EDA industry.

I would like to acknowledge my friends/colleagues Pongstorn Maidee and Hushrav Mogal for several stimulating intellectual discussions over the course of my study. I would like to thank Prof. Cristinel Ababei for introducing me to the academic FPGA design tool VPR, and for showing me the ropes when I started out on my PhD program. I would also like to acknowledge my numerous friends outside my research lab who made this journey enjoyable. I would like to thank my friend, Ajitha Rajan, for always finding the time to listen to me and to support me throughout my PhD.

I owe a great deal to my parents and my sister for always being there for me. Without their prayers and support, I would never have been able to successfully complete my PhD.

Dedication

To my parents, and my sister.

Abstract

Prohibitive ASIC mask costs and stringent time-to-market windows have made FPGAs attractive implementation platforms in recent years. Modern FPGA architectures provide ample routing resources so that designs can be routed successfully. However, providing such great flexibility comes at a high cost in terms of area, delay and power. In the first part of this thesis, we propose a new FPGA routing architecture that utilizes a mixture of hardwired and traditional flexible switches. This mixture is obtained from careful profiling of benchmark circuits. The result is about 30% reduction in leakage power consumption, 5% smaller area and 20% shorter delays. Despite the increase in clock speeds, the overall power consumption is reduced.

With constant scaling of process technologies in the ultra deep-submicron regime, chip design is becoming increasingly difficult due to process variations. The FPGA community has only recently started focusing on the effects of process variability. In the second part of this thesis, we propose CAD and architecture techniques to mitigate the impact of process variations in FPGAs. We present a variation-aware router that optimizes statistical criticality. We then propose a modification to the clock network to deliver programmable skews to different flip-flops. Finally, we combine the two techniques and show a 9X reduction in yield loss that translates to a 12% improvement in timing yield. When the desired timing yield is set to 99%, our combined statistical routing and skew assignment technique results in a delay improvement of about 10% over a purely deterministic approach.

Another challenge with aggressive technology scaling is to ensure the reliability of circuits. Issues with circuit reliability manifest as intermittent failures caused by random particle strikes or permanent failures due to thermal stress. In the third

part of this thesis, we develop computationally efficient techniques for analyzing and optimizing reliability of circuits subject to transient errors. We propose a hybrid method that combines exact symbolic analysis with probabilistic measures to estimate reliability. We use such measures in rewiring and gate-sizing based methods to optimize reliability. We study trade-offs involved in terms of area, power and delay when optimizing reliability. Our proposed approach offers a speedup of 56X when compared to a Monte Carlo simulation based approach with only a 3.5% loss in accuracy. Our rewiring-based optimization framework improves reliability by 10% along with area and power improvements of 14% and 18% respectively. When we combine the rewiring and gate-sizing based optimization techniques, reliability is improved by 17% with modest area and power overheads.

In the final part of this thesis, we propose a fast thermal simulation technique for single-processor and chip multi-processor systems. Our technique can be used to estimate thermal stress in modern processors efficiently. A fast and accurate estimation of thermal stress in a system is critical to improving its reliability by preventing catastrophic permanent failures. Our proposed technique of evaluating temperatures across the chip is based on moment matching and moves most of the computation offline. Our temperature computation technique offers a speedup of 441X when compared to a conventional technique based on a Backward-Euler approach with average and maximum errors of 0.89 °C and 2.7 °C respectively. We observe that lateral heat conduction in the active and substrate layers are significant only for a short distance. We leverage this information to further improve the efficiency of thermal estimation and achieve a speedup of 1900X.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Field-Programmable Gate Array Architectures

Shrinking process technology, increasing chip complexity, complicated verification tasks and short time to market have all combined to make chip design extremely difficult. In an attempt to alleviate these problems, Field-Programmable Gate Arrays (FPGAs) have emerged as the medium of choice for several digital design initiatives. FPGAs are integrated circuits that can be programmed to implement any digital design after fabrication. In addition to being able to statically implement multiple designs, FPGAs can also be reconfigured during runtime to implement different tasks at different times. The ability to instantly program the device reduces the time-to-market as well [5].

From an economic standpoint, the high costs involved in manufacturing masks for Application Specific Integrated Circuits (ASICS) make it viable only for extremely large volumes of production. For general purpose integrated circuits like FPGAs, this cost is amortized by several users and is the preferred medium of implementation for low- and medium-volume of production. The cost tradeoffs involved with FPGAs and ASICs are shown in Table 1.1 [75].

There is a steep price to pay to reap the benefits outlined above with FPGAs. In several key design metrics such as performance, power and area, FPGAs lag ASICs by a huge margin. The main factors that impact the quality of FPGAs are

1. The underlying FPGA architecture.

	FPGA	Cell-Based ASIC
Total Design Cost	\$165K	\$5.5M
Vendor NRE	None	\$1M $\sim$ \$3M
Cost of EDA tools	\$30K	>\$300K
Man Power	1 to 2	5 to 7
Price per Chip	\$200 $\sim$ \$1K	\$30
Unit Cost (Qty 1K)	\$1000	\$55K
Unit Cost (Qty 5K)	\$220	\$1.1K
Unit Cost (Qty 500K)	\$40	\$11

Table 1.1: FPGA Vs ASIC Development Costs

2. CAD tools used to implement designs on FPGAs.

We briefly review a basic FPGA architecture in Section 1.1.1 to illustrate the role played by architecture and CAD tools in making FPGA design solutions slower, larger and more power-hungry than their ASIC counterparts. In section 1.3, we outline techniques that we have developed in this thesis to address these shortcomings.

1.1.1 SRAM Based FPGAs

In an SRAM based FPGA, SRAM cells are used to store configuration bits needed to program a chip to perform a function. This allows instant reprogrammability. Logic as well as interconnections in the device can be reprogrammed by changing the values stored in the SRAM cells.

Figure 1.1 shows a conventional six-transistor SRAM cell. In addition to facilitating easy reprogrammability, SRAM cells can be implemented using leading edge CMOS processes because of the regular structure of memory arrays. Since SRAM cells are volatile, the data stored is lost when the power is turned off. Therefore,

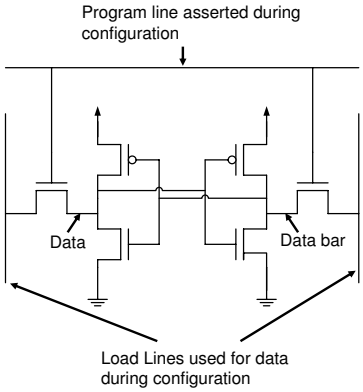


Figure 1.1: Six transistor SRAM cell

additional off-chip memory, like electrically erasable programmable read only memory is necessary to store the configuration bits and program the FPGA during power up [5].

Logic Blocks

Lookup tables (LUTs) are the basic building blocks of most FPGA architectures. A K-input LUT is a memory with 2^K bits that can be used to implement any logic function with K or fewer inputs. The truth table of the function to be implemented is stored in the memory unit forming the LUT. This memory is addressed by the K input lines. Figure 1.2 shows an example of a 2 input LUT implementing an AND gate. Modern FPGAs typically contain LUTs that have anywhere between 4 and 7 inputs. To implement sequential logic, LUTs are typically combined with flip-flops. Multiplexers are used to select either registered or the unregistered outputs. An LUT and flip-flop pair is called a basic logic element (BLE). A BLE is illustrated in Figure 1.3. A collection of BLEs along with fast routing resources between them is called a logic cluster.

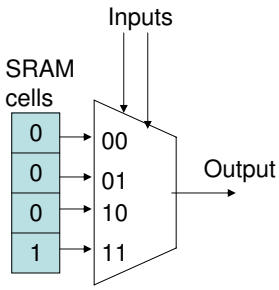


Figure 1.2: Two input LUT

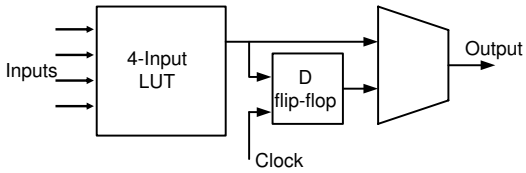


Figure 1.3: Basic Logic Element

Routing Architecture

Connections between logic blocks are implemented using prefabricated metal tracks. Logic blocks are surrounded by horizontal and vertical channels containing several metal tracks. Programmable connection matrices are used to connect input and output pins of logic blocks to metal tracks. Switch blocks provide connections between metal tracks at the intersections of horizontal and vertical channels. A high level view of the routing architecture is shown in Figure 1.4.

Connection matrices and switch blocks are made of programmable switches that implement connections between different routing tracks and between logic block pins and routing tracks. These switches can be buffered or unbuffered as shown in Figure 1.5.

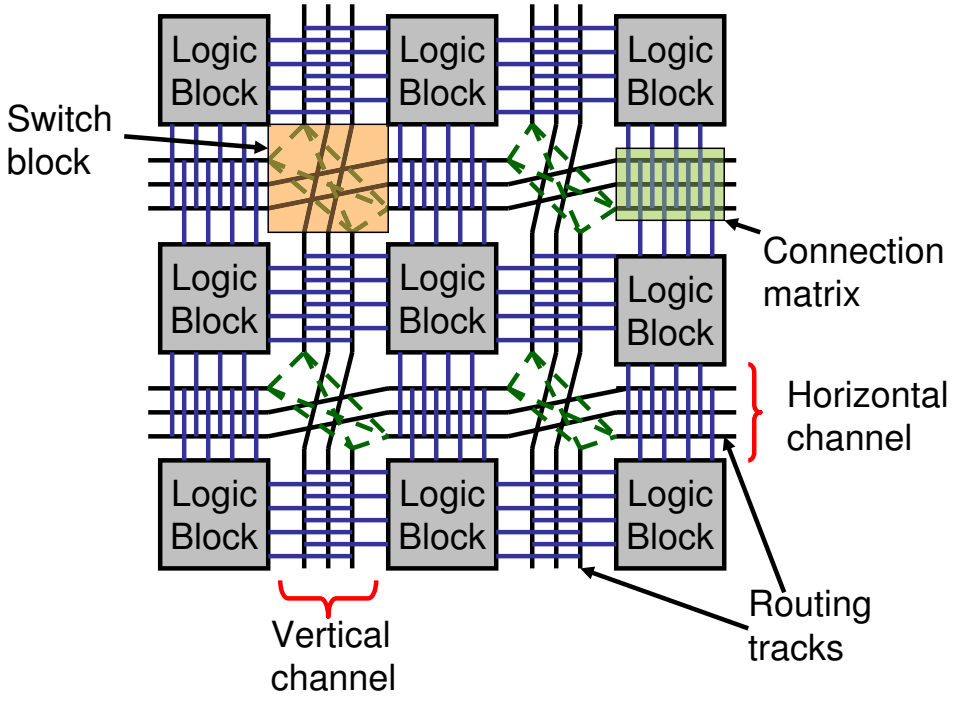
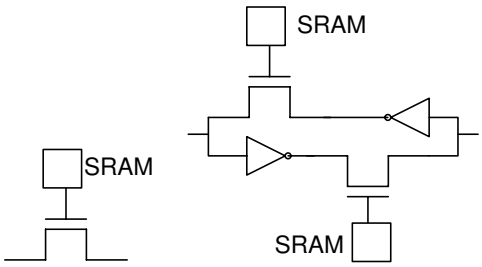


Figure 1.4: Routing Architecture



(a) Unbuffered (b) Bidirectional

Figure 1.5: Programmable Switches

1.2 Challenges in Circuit Design

Some of the challenges involved in designing circuits in the deep nanometer regime are outlined in Sections 1.2.1, 1.2.2 and 1.2.3. This dissertation attempts to address these challenges.

1.2.1 FPGA Shortcomings

The flexibility offered by FPGAs comes with a steep price in terms of performance, power and area. Programmable switches that make FPGAs flexible, adversely impact other design criteria leading to a wide gap between the performance, area and power that can be achieved by a custom design and a design that is mapped on a general purpose FPGA. While technology scaling and architectural modifications in modern FPGAs such as carry chains, embedded multipliers, etc., have reduced the gap between FPGAs and ASICs, the difference is still significant. Newer architecture and design techniques are needed to further shrink this gap before FPGAs can be used to implement high performance designs.

1.2.2 Process Variations

As process technologies continue to shrink to keep up with Moore's law, process variations are starting to threaten the viability of transistor scaling. Process variations such as uncertainties in physical dimensions of transistors, dopant concentration, oxide thickness etc., are caused by variations in the fabrication process. They are fast becoming critical aspects to consider during circuit design. Failure to do so would result in large numbers of chips failing to meet timing and power requirements.

To alleviate problems due to variability, ASIC designers have been using statistical techniques to anticipate the impact of variations by modeling them as statistical distributions. CAD tools have been redesigned to work with these distributions instead

of relying on purely deterministic quantities to make more accurate design decisions.

FPGA designers on the other hand, have mostly ignored the effects of variations. However, as FPGAs become more complex and faster with every technology generation, the impact of process variations would soon need to be explicitly considered. A paradigm shift in the FPGA design flow is needed to make circuits robust against variations.

1.2.3 Reliability of Circuits

All the issues discussed so far are related to the performance of circuits and do not affect their functionality in most cases when safety precautions such as guard-bands to compensate for variations are adopted. While the results maybe sub-optimal in terms of performance, area and power, circuits would still continue to implement the desired function for the most part.

The impact of aggressive transistor scaling on circuit reliability on the other hand may lead to catastrophic failures of circuits. These issues manifest themselves as either transient or permanent failures. Transient failures are caused by effects such as thermal fluctuations or radiation strikes and depending on the application, can be tolerated by certain systems. On the other hand, permanent failures, caused by effects such as thermal stress, are more severe.

With shrinking transistor dimensions, the frequency of transient and permanent failures are increasing sharply [32, 69]. For smaller devices, the energy required to create random bit flips due to particle strikes is reducing with every technology generation. Previously, it was thought that transient errors mainly affected memory cells. However, it has been shown that even combinational circuits are beginning to get affected by transient errors [62]. Another byproduct of transistor scaling is rising power densities of chips. As a result, elevated on-chip temperatures and tempera-

ture variations are increasing the thermal stress and can cause permanent damage to chips.

To ensure that circuits implement what they are built for, reliability-aware techniques need to be integrated in circuit design. Reliability has been an after-thought for most designers for a long time. However, if we aggressively continue to scale down process technology, reliability will soon be a dominant parameter in the design of circuits. Innovative techniques to efficiently estimate and prevent circuit failures are needed to meet the demands of the future.

1.3 Contributions

This dissertation attempts to address some of the issues listed in Section 1.2. The work is divided into four chapters with each providing a possible solution to a design challenge. The solutions outlined in this thesis are by no means exhaustive and it is virtually impossible to cover the entire range of potential solutions to design problems. This thesis is merely an attempt to give the industry and research community a firm starting point to tackle these issues as they become more prominent in the future.

Figure 1.6 illustrates how this thesis is organized.

In Chapter 2, we attempt to bridge the performance gap between FPGAs and ASICs (outlined in Section 1.2.1). It has been established that programmable switches in FPGAs are responsible for their poor performance [18,38]. Our approach to tackle this problem involves asking the following questions.

1. Are we providing too much flexibility in the FPGA architecture?

2. Can we sacrifice some flexibility to improve performance?

We develop architectural modifications to the FPGA that improve performance at

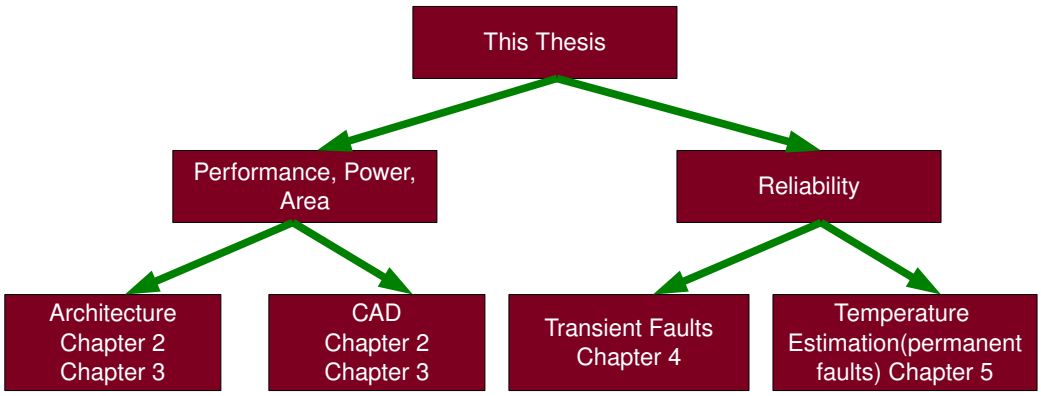


Figure 1.6: Thesis Organization

the cost of some flexibility. We also develop CAD algorithms to deal with the new architecture.

While Chapter 2 provides a solution to the performance problem from a purely deterministic viewpoint, Chapter 3 addresses the same problem from a statistical viewpoint. The impact of process variations is becoming greater with every technology generation and may soon be at a point where variability erodes any gain from scaling. In the first half of the chapter, we develop a process variation aware router for FPGAs to alleviate this problem. We choose the routing stage since it plays a dominant role in the performance of FPGAs. The second half of the chapter develops architectural modifications to the clock distribution network to distribute programmable skews to different flip-flops in the design to compensate for process variations. CAD algorithms are developed to support the new clock distribution architecture and to robustly distribute skews.

Chapters 4 and 5 address the orthogonal issue of circuit reliability. We develop accurate and efficient techniques to estimate reliability of circuits in the presence of transient errors in Chapter 4. We combine probabilistic measures with symbolic analysis techniques to develop a novel reliability estimation algorithm. We also de-

velop optimization approaches by using circuit rewiring techniques and gate sizing to harden circuits to transient errors. The impact of our proposed techniques on metrics such as performance, power and area are analyzed. In Chapter 5, we develop a fast thermal simulation technique based on the principles of moment matching to estimate on-chip temperatures. This is vital to efficiently estimate thermal stress in circuits. Knowledge of thermal stress can be used to prevent permanent failures from occurring. While Chapter 5 deals with the estimation of temperatures, optimizing thermal stress to reduce permanent failures is beyond the scope of this thesis.

Finally, Chapter 6 summarizes the work in this dissertation and points to directions for future work.

Chapter 2

HARP : Hard-Wired Routing Pattern FPGAs

2.1 Introduction

Prohibitive ASIC mask costs and stringent time-to-market windows have made FPGAs an attractive implementation platform in recent years. However, circuits implemented on FPGAs are typically slower, occupy more area, and consume more power than ASIC circuits [83]. The FPGA routing architecture is the main culprit in making FPGAs worse than ASIC chips in area, delay and power; a typical FPGA routing architecture uses about 70-90% of the total transistors on the die [26].

A significant body of work from the past two decades focused on switch box design and segmented routing architectures. The basic idea is to use highly flexible switches where horizontal and vertical tracks meet, to facilitate all possible connections between the adjacent tracks. A sketch of the disjoint switch box is shown in Figure 2.1.

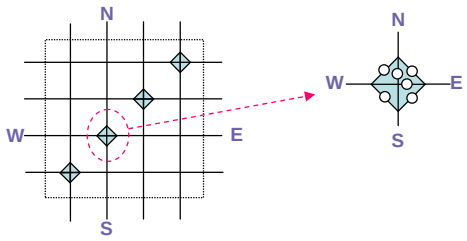


Figure 2.1: SRAM-based Switch Box.

Assuming all tracks have unit length, the disjoint switch box (see Figure 2.1) can route a large subset of possible routing trees to connect the terminals of a net. As a result, the overall channel width will not be high. However, flexibility in routing comes with great performance costs. Building a routing tree from many segments that are connected by switches has the following disadvantages:

- **Circuit Delay:** The delay of a net is mainly dependent on the number of programmable switches in its routing path [18, 38]. Hence, a large number of programmable switches contributes greatly to the overall circuit delay.
- **Area:** By increasing the number of programmable pass transistors (which correspond to the small circles in the switch on the right in Figure 2.1) inside each switch, we pay an area penalty as each of the programmable pass transistors requires an SRAM cell for programming it and possibly buffers to improve signal slew.
- **Leakage Power:** Leakage power is becoming a major component of the total power consumption [7] and the majority of the leakage power consumption in FPGAs occur in the routing switches [29].

2.1.1 Related Work

There has been a lot of work on programmable architectures to improve the performance of FPGAs. Modern FPGAs utilize multi-length horizontal and vertical segments. Recently, there has been a flurry of research in structured ASIC solutions [79], which aim to provide a middle ground between ASICs and FPGA. Tong *et. al.* [70], Jayakumar and Khatri [33], and Yan and Marek-Sadowska [58] proposed via-configurable gate array implementation platforms, in which connections are programmed by the presence or absence of vias. This results in improvements in

power-delay performance but the flexibility and cost savings are limited because of its mask programmability.

There have been several CAD techniques aimed at reducing the number of "bends" in the routing architecture. For example, the work in [36, 37] focused on the concept of using prespecified patterns to route a net. By doing so, one would allow a more accurate prediction mechanism for metrics such as congestion and wirelength earlier in the design flow. The work in [36, 37] focused on an ASIC design flow, rather than the FPGA flow found in this chapter.

Maidee *et. al.* [47] proposed a "terminal alignment" heuristic, which reduces the number of bends on nets, and hence eliminates switches that need to connect horizontal tracks to vertical ones. As a result, the number of switches used in routing of critical nets decreases. They achieved 5% delay improvement over VPR using a simulated annealing engine.

2.1.2 The Idea of HARP (HArd-wired Routing Pattern) FPGAs

In this work, we extend the idea of eliminating bends and switches (inspired by FPGA architectures such as Xilinx's Virtex, and placement and routing tools such as [36, 37] and [47]) to two dimensions; instead of just hardwiring two horizontal or two vertical segments to form longer wires, e.g. segmented routing architectures such as Xilinx Virtex, we form hardwired junctions between horizontal and vertical segments inside switch boxes. These junctions create routing segments in the shape of T's, L's and +'s and their rotated versions. An example of such a switch box is shown in Figure 2.11. As a result of hardwiring connections, we eliminate some programmable switches, which decreases the delay, area and power dissipation. However, we must be careful that the reduction in programmable switches does not severely affect the routing flexibility.

Figure 2.2 shows a conceptual diagram of a HARP architecture that contains 6x6 switch boxes (logic blocks are not shown in the figure). The figure shows a subset of the routing resources. Switch box 4D contains L and T HARP routing resources and one traditional flexible switch (the bottom track).Two of the L connections span one switch box on each side whereas the other two span two switch boxes on each side. Switch box A1 shows an L connection that spans 5 switch boxes on each side. The logic block at tile 6A can connect to the logic block at tile 1F through the fast L connection. An L HARP resource and a T resource are merged in tile 6E to form a more complex HARP pattern. Note that this connection is hardwired, as opposed to the connection between the flexible switch and the T HARP resource at tile 4D.

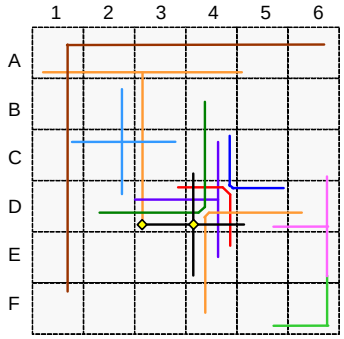


Figure 2.2: Global view of a HARP architecture.

Our contributions can be briefly described as follows:

- *Routing requirement analysis:* A number of circuits are placed and routed on traditional FPGA architectures, and the routing patterns that are formed in the switch boxes are analyzed.

- *HARP architecture generation:* based on the frequencies of the patterns that the router uses, we instantiate hard-wired routing patterns (HARPs) that replace some switches in the routing structure.

- *Placement and routing with HARPs:* We place and route circuits on the new HARP architecture.

After placement and routing on HARP architectures, we report results of area, delay, power and channel width. The experiments show that our technique maintains the programmability of FPGAs, while improving their performance metrics.

The rest of the chapter is organized as follows. In Section 2.2, we describe the terminology that we use and the overall flow of our architectural design. In Section 2.3, we perform an empirical analysis on detailed routings to find the most common routing patterns and their densities. Based on this analysis, we design the architecture of the switch boxes containing the hard-wired routing patterns. Details of this step are discussed in Section 2.3.3. Section 2.4 explains how hard-wired routing patterns inside the switch boxes are exploited by the router. Experimental results are presented in Section 2.5. Section 2.6 details a method that generates a regular HARP FPGA and presents the results of placement and routing on such architectures. We conclude in Section 2.7, by outlining our main contribution and discussing future research directions.

2.2 Preliminaries

2.2.1 Basic Terminology

The routing of a multi-terminal net is frequently modeled as a *rectilinear Steiner tree (RST)*. A RST has three types of *joint patterns*: L-shape, T-shape, and +-shape. An FPGA routing architecture with uniform unit-length segmentation has a switch at each of the joint patterns. Additionally, there are switches for horizontal (H) and vertical (V) routes that span more than one channel. Modern FPGA devices use multi-length segments (−-shape and |-shape) in order to reduce the number of

switches along the horizontal or vertical routes of the nets. This enhances the delay of the routing; however, it reduces the flexibility of the architecture.

We call the switch shown on the right side of Figure 2.1 a *flexible* switch. A multi-length segmented architecture merges the "W" track and the "E" track to form a longer segment. This is equivalent to removing the pass transistors (and their associated SRAM cells and buffers) that connect the "E" or "W" tracks to other tracks. The result is a hardwired connection between "E" and "W". The area of this new switch is smaller, however, it is less flexible than the original *flexible* switch. If we allow the horizontal track to also connect to the vertical track at this junction (*e.g.*, the way hex lines in Xilinx architectures connect to other segments on the middle point), then two more switches will be used to provide connectivity between wire segments "WE" and "N", and also between "WE" and "S". Obviously, the area and delay of this switch increases, but we gain flexibility.

To the best of our knowledge, no one has extended the idea of hardwiring pass transistors to junctions that are formed between horizontal and vertical tracks. In this work, we study *HARP* (HArd-wired Routing Pattern) architectures that utilize hardwired "switches" at certain junction patterns. The three joint patterns (L, T, +, and H/V) and their various orientations result in eleven possible HARPs: $\top$,$\perp$, $\dashv$, $\vdash$, $\ulcorner$, $\urcorner$, $\llcorner$, $\lrcorner$, |,− and +.

2.2.2 HARP-based FPGA Routing Architecture Design Flow

Figure 2.3 shows our design flow to introduce HARPs into *traditional* FPGA routing architectures. First, we place and route a number of circuits on a traditional FPGA architecture. By analyzing the routes of the circuits, we extract the frequency at which different HARP patterns are used in switch boxes. Next, we use the results of the pattern distribution analysis to create a new architecture that has a mixture of

flexible and HARP switches. Finally, we place and route designs on the new HARP architecture and compare the results with the traditional architectures.

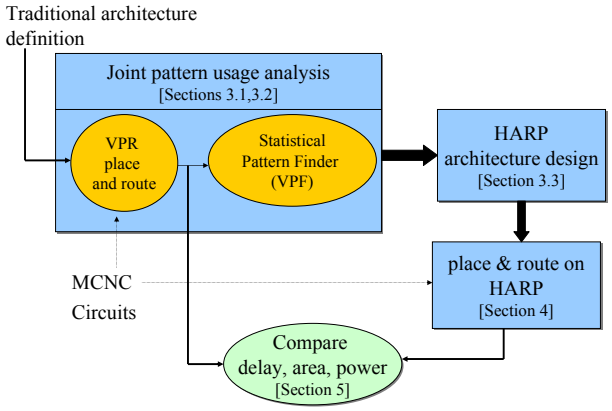


Figure 2.3: HARP-based Routing Architecture Design Flow.

2.3 Routing Pattern Analysis

In this section, we discuss the statistical analysis of routing pattern frequencies and correlations between circuits, architectures and these patterns.

2.3.1 Testing Benchmark and Routing Result Generation

After placing and routing the MCNC benchmark circuits, we observed how often each of the joint patterns can be found in a route of each net. Note that the placement and routing algorithms will affect the frequency of these patterns. We will discuss this issue in Section 2.7.

Statistical information can guide us on how often we need to replace flexible switches with HARP resources inside switch boxes. To find the pattern frequencies, we routed all the benchmarks on a given traditional segmented FPGA routing ar-

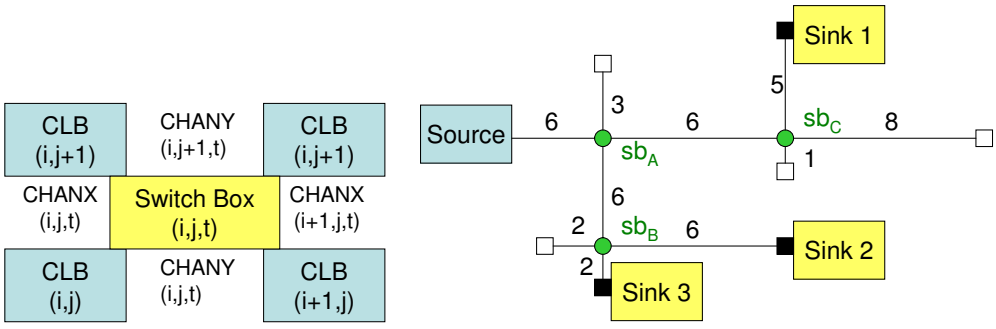
chitecture applying the VPR FPGA place-and-route tool [10]. For a given routing segmentation architecture, we routed each circuit, detected the patterns in the route files, and applied statistical analysis on the data.

In our studies, we considered two segmentation architectures: unit-length segmentation and multi-length segmentation (similar to Xilinx's Virtex family). We used the VPR router in three different modes: Timing-driven, Routability-driven without bend-cost, and Routability-driven with bend-cost. We experimented our technique on the MCNC benchmark suite [76].

2.3.2 Analysis of Routing Patterns

VPR Pattern Finder Tool

In order to analyze the behavior of the routing patterns, we have implemented *VPR Pattern Finder*(VPF), a graphic tool for parsing, visualizing and analyzing the VPR routing results [4]. VPF takes a VPR routing result file as input and automatically extracts the routing information, identifies the connection patterns at switch points, and in turn generates statistical reports for different patterns.



(a) switch box indexing (b) pattern labeling

Figure 2.4: Switch box indexing and pattern labeling in VPF

In our analysis, we focus on the connection patterns found in switch boxes. For a given FPGA layout, a switch point is indexed by a tuple (i, j, t) as shown in Figure 2.4(a), where i and j indicate the physical location of the switch box containing that switch point and t identifies the track being used. Based on the structure of the switch point, we have 11 possible connection patterns. They are $\top$, $\perp$, $\dashv$, $\vdash$, $\ulcorner$, $\urcorner$, $\llcorner$, $\lrcorner$, $|$, $-$ and $+$. As an example, Figure 2.4(b) shows a sample net, which contains only one source and three sinks. Empty rectangles show ends of the segments that are not connected to this net. The numbers on the lines denote the length of the connections (that are possibly formed by connecting two or more segments). Based on the above discussion, switch box sb_A has pattern $\top$, sb_B has pattern $\vdash$, and sb_C is of pattern $\lrcorner$.

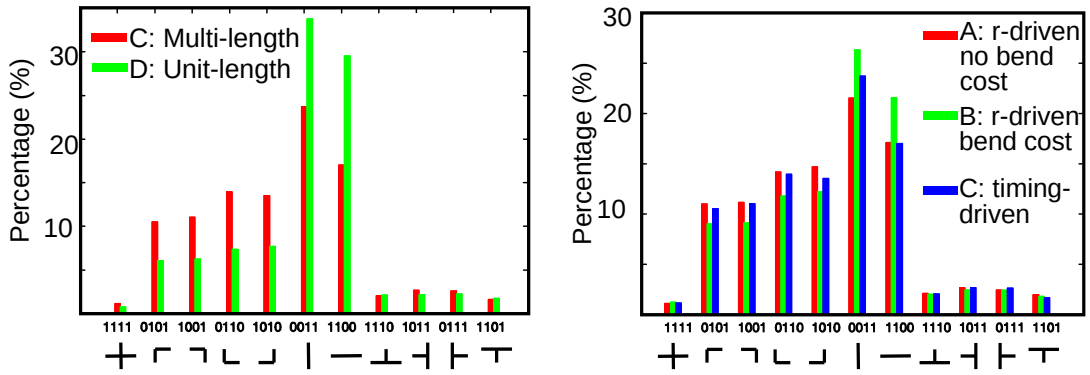
It is important to find out how each HARP extends along different directions - the *pattern length*. This information can provide more insight into the routing behavior and offer useful guidance for improving routing quality by hard-wiring these patterns. VPF performs this analysis using the following simple algorithm: (i) For each marked switch box, identify its pattern. Based on the pattern information, try to trace along the valid directions starting from the switch box; (ii) Stop when we meet a switch point that has a pattern other than $|$ when we are tracing vertically or $-$ when we are tracing horizontally. Furthermore, we need stop the tracing when we reach the source or a sink; (iii) Report this distance as the result of the current direction; (iv) Take the minimum among all directions as the pattern length of the current switch point. Following the same example shown in Figure 2.4(b) and assuming the numbers by the segments indicate the segment lengths, switch points sb_A, sb_B and sb_C have pattern lengths 6, 2 and 5 respectively.

Statistical Results and Analysis

Statistical information about the switch box patterns obtained from VPF provide insight into the behavior of the placement and the routing tools as well as resource demands of the circuits. Figure 2.5 shows the normalized pattern distributions (in percentage of all the switch points) for different benchmarks and segmented architectures. Among them, the unit-length (D) results are generated on an architecture that only supports segment of length one, while those of A, B and C are generated on a Virtex style architecture with multi-length segment routing architecture. A is generated with the routability-driven routing without bend cost, B corresponds to routability-driven with bend cost, and C is generated using the timing-driven routing mode. Placement for all experiments was done using the timing-driven mode. Routability-driven algorithm aims at minimizing the length of each route. When no bend cost is considered, many bends can appear in the routes. On the other hand, the router considering the bend cost generates the routes with smaller number of bends. Timing-driven router does not consider the bend cost directly. Since the delay of a route in FPGA is dominated by the number of switches and each bend includes a switch, the algorithm avoids generating many switches. As a result, each of the three routing algorithms manages the bends in the routes differently. We applied the three algorithms to observe the distribution of all different patterns when different routers are used.

One interesting observation that can be made from Figure 2.5(a) is that the multi-length segmented architecture greatly changes the pattern distribution compared to unit length. The combined frequency of vertical and horizontal patterns drops from 63.3% to 41.2% [1]. On the other hand, as seen in Figure 2.5(b), when using different

[1] The reason is that the multi-length segments are in essence horizontal and vertical connection patterns. For example, a horizontal segment of length 6 is the equivalent of 5 horizontal switch



(a) results for different architectures (both timing driven)

(b) results for different routing considerations on the same multi-length architecture (Virtex-like)

Figure 2.5: Switch box pattern distributions

router settings on the same Virtex style architecture, the results do not vary much. In other words, the architecture seems to have a much bigger impact on the switch box pattern distribution than the routing algorithm.

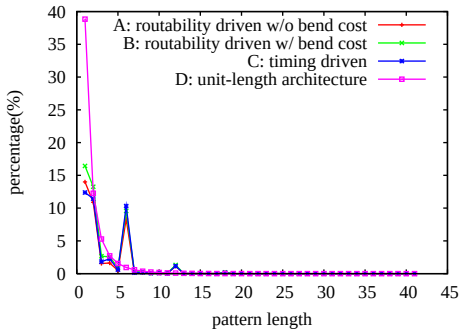
Figure 2.5(a) shows there is little change in the percentage of the T patterns or the + pattern when we switch from unit-length to the multi-length segment architecture. On the contrary, there is a significant increase for the L patterns. The combined frequency of all the L patterns increases from 27.46% for the unit-length architecture to 41.62% for the multi-length architecture. In other words, for the multi-length architecture, the possibility of having an L patterned switch point is comparable to (if not more than) that of a vertical or horizontal pattern. This is a notable difference compared to the unit-length results, in which the vertical and horizontal patterns overwhelmingly dominate the pattern distribution.

Next, we analyze the length of the patterns using the method discussed in Sec-

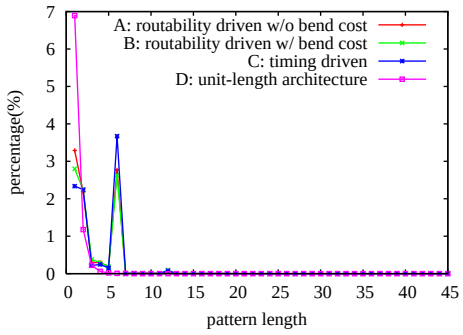
---

connections in the single segment architecture.

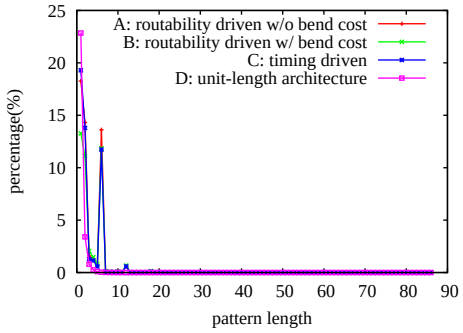
tion 2.3.2. Figure 2.6 illustrates the pattern length distributions for our testing benchmarks. The x-axis in these graphs is the pattern length, while the y-axis is the normalized percentage for switch point patterns with the given length. Benchmarks A, B, and C are generated using the same multi-length Virtex style architecture with different routing considerations, i.e. routability-driven without bend-cost, routability-driven with bend-cost,and timing-driven; Benchmark D is the timing-driven result using the unit-length architecture.



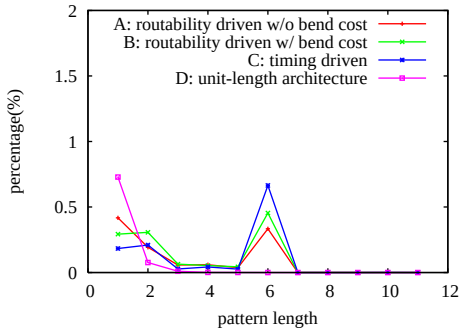
(a) Length distribution for pattern | and –



(b) Length distribution for T patterns



(c) Length distribution for L patterns



(d) Length distribution for pattern +

Figure 2.6: Distribution statistics on switch point pattern lengths

We can observe that all these graphs share some common characteristics. First, for the unit-length architecture, the pattern length distribution drops rapidly and

monotonically as the length increases. The results for the multi-length architecture are not monotonically decreasing but still follow a similar trend except for length 6, which shows spikes on all patterns. On all patterns except +, there is a small spike at length 12 too. Such harmonic behavior demonstrated by these spikes is not surprising because in the multi-length architecture, the majority of the segments are of length 6, where switch connections are allowed at both ends of the segments.

We also performed further analysis focusing on the geometric distribution of different patterns. We observed uniform distribution of all patterns with the exception of one benchmark[2]. It is important to note that our results are valid only for the class of circuits represented by the MCNC benchmarks that we used in these studies. For example the routing pattern statistics of data-path designs might show different characteristics than those shown here. A bus-based CAD flow such as [77] can be used to place and route such designs, and the routing analysis can be used to generate HARP architectures tailored to data-path circuits.

2.3.3 Architecture Design

Architecture design for FPGAs is a complex problem and much work has been done in this area since FPGAs were first proposed. There are many architectural factors (such as switch-block or switch-matrix style, switch-block flexibility F_s, connection-block flexibility F_c, frequency of switch-blocks along routing segments, channel segmentation and staggering, clustering of LUTs in CLBs), which contribute to the quality of the final FPGA platform. More details about such architectural features is provided in our FPGA'05 paper [67].

Based on the analysis presented in the previous section, we design a new switch

[2]For circuit "bigkey", the + patterns were concentrated in the two horizontal channels at the center of the chip.

box, which will include hard-wired routing patterns. This means that we remove a certain number of programmable switches (derived from statistical analysis of the routing profiles of various circuits) from the switch boxes and replace them with wires. The composition of these hard-wired patterns are chosen after careful analysis of the routing profiles, hence the effect on the routability of circuits is minimized. We have experimentally verified the minimum effect of HARP resources on the routing of circuits (see Section 2.5).

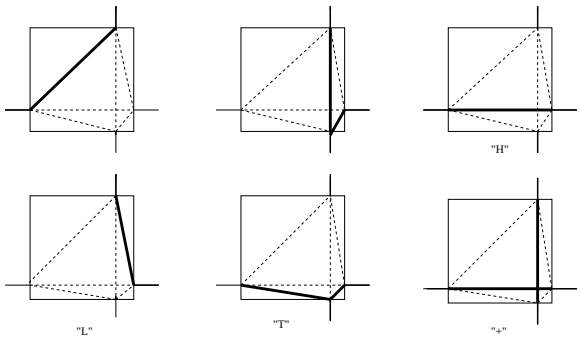


Figure 2.7: Some possible hard-wired patterns

Figure 2.7 shows some of the possible HARPs that can be present inside the switch boxes. The hard-wired patterns are shown using solid lines indicating that they are wires and not programmable switches. The next section describes how the routing tool is made aware of these patterns and how they are exploited to reduce the delay, area and power dissipation.

2.4 Routing with HARPs

To harness the advantages of HARP architectures, the placement and routing tools must be adapted to use hard-wired resources for timing critical nets and only use regular switches where hard-wired resources are not available.

In this work, we would like to fully exploit the hard-wired patterns present inside our switch-blocks. This can be done in the detailed routing stage by constructing a routing graph with the hard-wired routing patterns embedded as low cost edges. VPR [10], and the power model from Wilton, *et. al.* [56], which we use for our work employ a routing graph construction approach to perform detailed routing. The routing segments and the logic block input and output pins are represented as vertices in the routing graph with a certain cost associated with them. Edges in the routing graph correspond to the connections between them. Edges may be bidirectional or unidirectional depending on whether a pass-transistor or a buffered switch is used [10]. A sample routing graph is shown in Figure 2.8 [10].

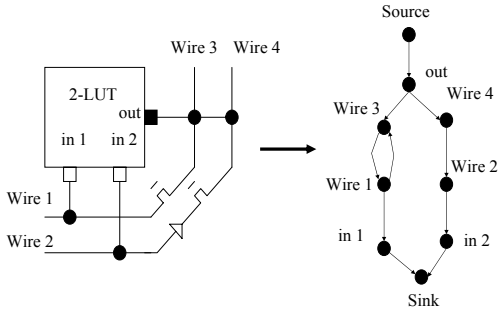


Figure 2.8: Sample routing graph

The way the routing graph is constructed changes with the presence of hard-wired patterns. These changes occur inside the switch boxes. Figure 2.9 shows the routing graph for a disjoint switch box (with pass transistor switches) with all tracks terminating at the switch box, and a disjoint switch box with a L-shaped hard wired pattern embedded in it. With the L-shaped pattern in the switch box, the routing graph contains only those edges forming the pattern and all the other edges are removed from the graph. For instance, in figure 2.9, edges between nodes (A,C),

(A,D), (B,C), (B,D) are removed from the routing graph since they do not participate in forming the patterns.

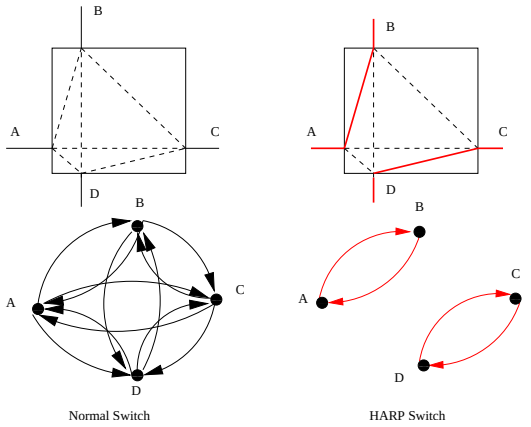


Figure 2.9: Routing graph with HARPs

Based on the results of the analysis presented in Section 2.3, we first determine the number of different HARP patterns that need to be inserted inside the switch boxes. Next, the FPGA chip is scanned row-by-row and patterns are inserted based on their desired percentages. When introducing these patterns, we make sure not to connect different hard-wired patterns together to form large trees. The reason for this restriction is illustrated by the example of Figure 2.10.

When we use two adjacent hard-wired patterns, an L-shaped pattern and a T-shaped pattern to connect terminal A to terminal B in the figure, a dangling segment is formed. This is undesirable as it adds extra capacitance and resistance, which is contrary to the goal of reducing overall power and delay. This problem is overcome by making sure that not many hard-wired patterns are connected back-to-back. In the rest of this section, we present our algorithms assuming that *no* two HARPs are allowed to connect back-to-back, but later on in Section 2.5, we relax this restriction a little and observe that the *limited* use of merged HARPs will improve the quality of the circuit.

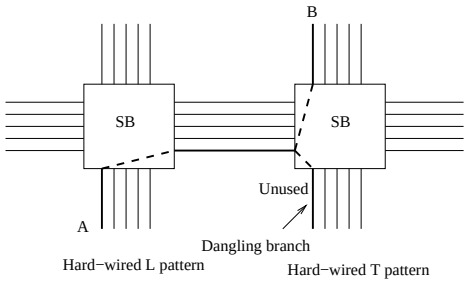


Figure 2.10: Connecting Hard-wired patterns together

Once we know the number and location of these hard-wired patterns, we change the way VPR constructs the routing graph and include only those edges (corresponding to wire segments) that are actually connected to the pattern. These edges are inserted as low cost edges so that the router will automatically choose these hard-wired patterns when performing detailed routing. The cost is calculated based on the lumped resistance and capacitance of the wire segment (including HARP and regular segments) connected to a switch. Interested readers can find the pseudo-code for inserting the hard-wired patterns in the architecture in our FPGA paper [67].

Note that ⌜ and ⌟ can be combined into one switch configuration which makes two disjoint connections (one between right and bottom segments, and the other between top and left segments). The same is true with ⌝ and ⌞. See Figure 2.11 for examples of L patterns (in SB2, the fourth switch from the bottom is an {⌜, ⌟} switch). Our architectural generation code is available for download from [1] for non-commercial use.

2.4.1 Estimation of Delay, Area and Power

We use the delay and area models in VPR and the power model developed by [56] to estimate the circuit delay, total area of the chip and the total power dissipation after inserting the hard-wired resources. VPR uses an Elmore delay model to estimate the

delay of every net. In this model, pass transistors are represented as resistors and diffusion capacitances to ground. Pass transistors add parasitic capacitance to the wire irrespective of whether they are on or off leading to a higher delay [10]. In our hard-wired resources, we eliminate the pass transistors and replace the resistance and capacitance values of the pass transistors, used in the delay model, with those of the metal wire (of segment length 1).

To accurately determine the delay of using a hard-wired resource, the capacitance of all the segments forming the pattern are included in the total capacitive load being switched. In addition, when only some of the segments of a hard-wired switch are used to route a signal, the remaining segments are made unavailable to route other nets. This avoids potential resource conflicts that could occur when different nets try to use different parts of the same hard-wired switch. For example, when only the vertical segment of the hard-wired L is used to make connections and the horizontal part is dangling. In this case, the horizontal segment is invalidated so that it is not available for use by other nets. More details of this procedure can be found in our FPGA paper [67].

The area model in VPR is based on counting the number of transistors required to implement the FPGA architecture. It reports area in terms of the number of minimum width transistor areas required to implement the circuit on the FPGA [10]. For our hard wired resources, we use the same procedure and count the total number of transistors in our implementation. We use the power model developed by [56] to estimate the total power dissipation. Leakage power is estimated by counting the number of unused transistors and SRAM cells and multiplying them with their individual leakage power. Dynamic power is dependent on the charging and discharging capacitance and the clock frequency, which is the critical path delay. The short circuit power is taken as 10% of the dynamic power. The charging and discharging

capacitance is obtained from the parasitics used in the delay model of VPR.

2.5 Experimental results and analysis

2.5.1 System Performance Improvements

We inserted the hard-wired patterns in the switch boxes and used a multi-segment routing architecture with routing-segments of lengths 1, 2, 6, and long lines. The distribution of the segments in each channel are 8%, 20%, 60% and 12% respectively (similar to the Virtex architecture) for all simulation experiments. HARPs were not inserted on long lines, though. We placed and routed the 20 MCNC circuit benchmarks of the VPR package and 3 of the large benchmarks of the Altera QUIP tool set[3] (oc_wb_dma, oc_mem_ctrl, oc_des_des3perf, which have 9872, 8611 and 38,218 CLBs, and 9654, 8726, and 38,452 nets respectively) on HARP architectures and report the results of circuit delay, area, leakage power, total power dissipation and channel width. It is important to note that the reported area is only the transistor count. A complete architecture generation flow would use an ASIC CAD tool to generate a detailed placement and routing of the FPGA architecture itself. Since the layout is going to be generated automatically, the area efficiency of the HARP FPGA would probably be worse than a traditional FPGA that is manually laid out. However, our tile-based HARP architecture (will be presented in Section 2.6) would enable designers to manually lay out a limited number of tiles and replicate them throughout the chip.

We updated the delay look up tables used by the placement tool of VPR to reflect

[3]Since these circuits make extensive use of register enables and other control signals that are not present in VPR, we recompiled the designs in Quartus by suppressing the use of adders and multipliers.

delays of HARP connections. However, this had only a marginal impact on the placement quality, more specifically, the delay improved 1-2% which is statistically insignificant, but channel width increase by about 5% and consequently the area and the total energy consumption increased by 3% and 5% respectively. The reason is that these delay lookup tables are built assuming no congestion is present, and hence the best routing resources for delay are always available. This is an optimistic lower bound on the routing delay between two points. In reality, the router will have to use traditional, slower switches for some nets due to congestion.

The problem is compounded by the fact that by using HARP resources, the delay difference between timing critical and non-critical nets reduces. When more nets become critical, congestion becomes a problem as many nets try to occupy the relatively few HARP resources. The resulting congestion increases the final channel width and worsens other metrics but the worst case delay remains more or less the same. Since we do not have a good estimate of the congestion at the placement level, we decided to leave the delay tables untouched to better capture the lack of enough routing resources at the routing level. We plan to improve the placement quality by using better models of the HARP architecture and use a tightly coupled timing analyzer in our future work.

Results of the execution of VPR with 50% of all switches replaced with HARPs and that of the traditional "Virtex-like" architecture is presented in Table 2.1. Columns labeled "Vtx" show the results of the traditional "Virtex-like" routing architecture. The last row of the table shows the ratio of the geometric mean values for Vtx and HARP.

We observe that the insertion of hard-wired routing patterns has a profound impact on delay and leakage power dissipation, reducing delay by about 17.45% and leakage power by 22.11% on average. Insertion of hard-wired routing patterns as low

Circuit	Delay ($x10^{-8}$)		Area ($x10^6$)		Channel Width		Leakage Power		Total power		Energy ($x10^{-8}$)	
	Vtx	HRP	Vtx	HRP	Vtx	HRP	Vtx	HRP	Vtx	HRP	Vtx	HRP
misex3	6.31	5.33	2.88	2.71	20	23	0.12	0.09	0.22	0.22	1.41	1.18
alu4	7.12	5.97	3.11	2.74	19	21	0.13	0.10	0.23	0.22	1.67	1.34
apex4	7.17	5.98	2.83	2.58	22	24	0.12	0.09	0.18	0.17	1.31	0.99
ex5p	6.40	5.50	2.36	2.23	22	25	0.10	0.08	0.17	0.16	1.11	0.90
des	7.89	6.00	6.02	5.75	16	18	0.25	0.21	0.41	0.41	3.27	2.45
seq	6.39	5.31	3.58	3.46	20	24	0.15	0.12	0.27	0.27	1.73	1.43
apex2	7.45	5.92	4.01	3.80	21	24	0.17	0.13	0.28	0.28	2.09	1.66
spla	12.40	8.93	9.90	9.70	28	33	0.43	0.34	0.52	0.48	6.51	4.28
pdc	14.20	10.70	14.80	13.70	33	39	0.64	0.50	0.75	0.65	10.65	6.98
ex1010	15.50	11.50	8.95	8.66	19	23	0.38	0.31	0.48	0.45	7.39	5.15
clma	18.6	14.2	19.6	18.6	24.2	28.8	0.83	0.60	1.10	1.02	20.6	14.6
dsip	4.39	3.88	3.92	3.96	13.4	16.8	0.15	0.10	0.39	0.42	1.74	1.63
diffeq	6.08	5.15	2.22	2.18	14.8	17.6	0.09	0.07	0.18	0.17	1.08	0.910
elliptic	8.87	6.58	7.38	6.98	20	23.4	0.32	0.22	0.51	0.51	4.55	3.38
frisc	9.28	7.97	8.26	7.70	23.6	27.6	0.36	0.24	0.46	0.41	4.32	3.27
s298	10.8	9.17	3.20	2.94	17.2	18.4	0.13	0.10	0.22	0.20	2.47	1.91
s38417	8.53	7.69	10.2	9.74	16.6	18.4	0.36	0.36	0.77	0.73	6.60	5.61
tseng	5.77	5.30	1.49	1.45	13.3	16.7	0.05	0.04	0.13	0.13	0.76	0.72
bigkey	4.46	3.94	4.02	3.96	14	17.3	0.14	0.12	0.37	0.39	1.69	1.57
s38584.1	8.62	7.53	10.4	9.96	17	19	0.38	0.30	0.70	0.63	5.36	4.51
oc_wb_dma	8.86	7.41	25.1	23.8	28	33	0.99	0.77	1.15	1.15	10.24	8.50
oc_mem_ctrl	8.28	7.02	16.46	15.47	19	23	0.62	0.49	0.89	0.83	7.33	5.82
oc_des_des3perf	14.91	12.60	56.97	54.19	16	19	2.17	1.71	2.67	2.60	39.74	32.77
G.Mean ratio (%)		82.55		95.04		116.66		77.89		96.06		79.46

Table 2.1: Comparison of 50% HARPs with no HARPs. (**G.Mean** is the geometric mean)

cost edges in the routing graph encourages the routing tool to use them whenever possible. This leads to a considerable speed up of the circuit. Also, the elimination of the program bits results in fewer SRAM cells and a lower leakage power dissipation. We find that the total area of the circuit decreases by about 5% on average. However, the average channel width increases by around 16.66%. This is expected, since, the introduction of hard-wired routing patterns reduces the flexibility of the routing architecture causing the router to use more tracks to route certain connections. However, the overall routing area of the circuit decreases because the reduction in individual switch area dominates the increase in number of switches caused by increased channel width.

Total power dissipation reduces on average by about 4%. In spite of the 22% reduction in the leakage power dissipation, the total power reduces by only around 4% on average. This is explained by taking into account the dynamic power dissipation. Dynamic power dissipation is dependent on the switching rate of the circuit. With hard-wired patterns in the switch boxes, the critical path delays of the circuits are reduced considerably resulting in a higher clock rate. This causes increased dynamic power dissipation. A fair comparison metric between the Virtex-like routing architecture and HARPs would probably be the power-delay product. We can explore different configurations by considering the energy dissipation or the power-delay product. Power-delay of HARP is 21.5% better than Vtx. Depending on whether optimizing for speed or power is more critical, we can clock the circuits at a higher clock frequency to get a faster circuit or we can clock the circuit at a lower speed (e.g., the clock speed that a traditional Virtex routing architecture can achieve) to achieve more savings in power dissipation.

We also explored the potential benefits of increasing the percentage of HARPs inside the switch boxes by allowing a small percentage (10%) of HARPs to connect

to each other. This is illustrated in Figure 2.11, which shows HARP resources (HARP SW) and regular switches (FlexSW) lumped to form distinct regions inside the switch boxes. This representation is just for illustration purposes. In reality, HARPs are distributed throughout the switch boxes, and not just at lower tracks.

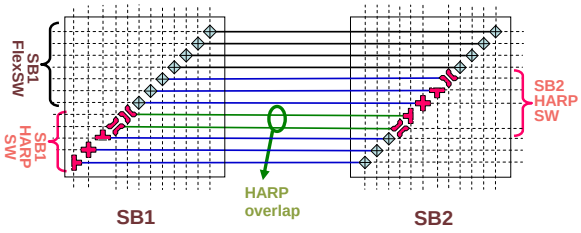


Figure 2.11: Overlaps in HARPs

Allowing a small percentage of HARPs to connect directly could create more complex routing patterns to be formed by combining hardwired patterns that we have used. However, doing so could also have the undesired affect of creating dangling wire segments (as illustrated in Figure 2.10) which could have an adverse effect on delay and power.

In terms of implementation, to increase the percentage of HARPS beyond 50%, we relax the constraint of not allowing different HARPs to connect together and allow HARPs to connect together sometimes (*e.g.*, to allow 60% of the switches to be HARP, we should allow HARPs to connect 10% of the time). As before, we take care not to form large trees of HARPs. This is done by making sure that there is at least one regular switch after every K switches, K being a constant (in our experiments, we used $K = 3$). The results of increasing the percentage of HARPs is presented in Table 2.2. We conducted some experiments to study the impact of increasing the overlap between HARPs beyond 60%. We observed that the results worsen as the percentage of overlaps increases beyond 70%. We believe that for each circuit, there is a sweet spot between 60% and 70% that would be optimum in terms of performance.

However, it is to be noticed that going from 50% to 60% of HARPs increases the channel width by about 7% and the improvement in delay is only about 3%. This indicates that it is not advisable to go much higher than 60% as the improvement in performance comes at a cost of a much higher increase in the channel width.

We observe that increasing the percentage of HARPs(60%) inside switch boxes increases the potential savings in circuit delay, energy and area to about 21%, 26% and 5% respectively.

2.5.2 HARP Usage Analysis

To better understand the benefits and potential future improvements on the proposed HARP architecture, we perform detailed analysis on the routing results of the HARP platform. As no major changes were made in our placement and routing algorithms compared to the traditional architectures, it is of our interest to find out how effectively the HARP connections are utilized in the final routing results. In order to find the utilization of HARP resources, we read the routing result files of the HARP architecture into the VPR Pattern Finder and compare the actual switch usage to the number of HARP resources provided in the architecture.

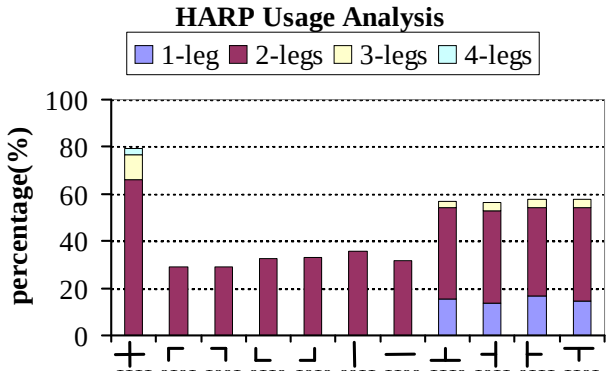
When comparing HARP resource availability to HARP resource utilization, we pay special attention to cases where more complex HARP connections such as the $\top$-shaped and +-shaped connections are only partially utilized, i.e., used as L's, H's and V's. As a result, utilization analysis, we report how many legs (e.g., up to 3 for the $\top$-shaped HARP connections and up to 4 for the + connections) are actually used. Zero-leg utilization means that the switch was not used at all.

The analysis is carried on the same 20 benchmarks. For each benchmark and for each HARP connection pattern, we report the percentages for each case when a different number of legs are used (ranging from 1 to 4). Then these numbers are

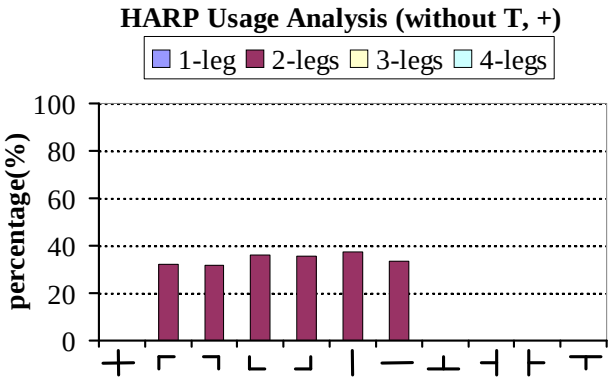
Circuit	Delay ($\times 10^{-8}$)	Area ($\times 10^{6}$)	Channel Width	Leakage Power	Total Power	Energy ($\times 10^{-8}$)
misex3	4.98	2.68	24	0.093	0.226	1.125
alu4	5.67	2.73	22	0.0101	0.228	1.292
apex4	5.74	2.59	26	0.093	0.168	0.964
ex5p	5.6	2.19	26	0.062	0.157	0.879
des	6.36	5.66	18	0.188	0.388	2.467
seq	5.30	3.45	26	0.121	0.268	1.420
apex2	6.10	3.74	25	0.114	0.270	1.647
spla	8.31	9.38	33	0.314	0.465	3.864
pdc	9.42	13.5	40	0.490	0.645	6.075
ex1010	11.4	8.45	24	0.299	0.429	4.891
clma	12.0	19.8	32.8	0.688	1.013	12.2
dsip	3.49	3.93	18	0.129	0.401	1.40
diffeq	4.80	2.15	18.4	0.0685	0.173	0.833
elliptic	6.71	7.24	27.4	0.247	0.504	3.38
frisc	7.95	8.05	31.4	0.267	0.411	3.27
s298	8.98	2.98	20.2	0.106	0.207	1.87
s38417	7.57	9.57	19.6	0.310	0.665	5.04
tseng	5.15	1.40	17.3	0.041	0.130	0.670
bigkey	3.85	3.90	18.0	0.107	0.358	1.38
s38584.1	7.45	9.90	20	0.252	0.57	4.25
oc_wb_dma	7.19	23.80	35	0.737	1.137	8.16
oc_mem_ctrl	6.68	14.54	23	0.436	0.801	5.35
oc_des_des3perf	12.05	60.24	22	1.855	2.461	29.66
Geometric Mean	0.796	0.949	1.241	0.690	0.938	0.745

Table 2.2: Comparison of 60% HARPs with no HARPs

normalized across all the benchmarks. The final result is aggregated in Figure 2.12(a). The graph shows that about 79% of the overall + shaped HARP resources are used in the final routing graphs. Among them, about 2.7% are fully used, i.e. all 4 hard-wired legs are involved in the connection, 10.5% use 3 hard-wired connections, and 66% use 2 hard-wired connections. The remaining 21% of this type of switch are not used in the final routing graph at all.



(a) Original HARP set



(b) Simplified HARP set: without ⊤ and +

Figure 2.12: HARP connection usage

Based on the results shown in Figure 2.12(a), we can observe that all types of

HARP connections are heavily used in the final routing graph. The usage of every switch type is always at least 29%. For $\top$-shaped and + shaped connections, more then 50% are utilized. However, for different $\top$-shaped and +-shaped connections, only a very small percentage are fully utilized. For most of them, only two legs are involved in the final routing. This may indicate that they are actually used as L, | or $-$ shaped connections. Moreover, considering that only small number of switches are configured as $\top$ and + (refer to Figure 2.5(b)), this motivates the idea to simplify the HARP pattern set by getting rid of the $\top$ and + patterns completely and at the meantime, providing more L, | and $-$ HARP connections.

Based on the observation made above, we are interested in investigating the effect of simplifying the HARP pattern set, namely by eliminating the $\top$-shaped and +-shaped HARPs and redistributing their percentages to the other HARP connections. We re-evaluated the performance gain on the benchmarks with this simplified HARP pattern set. The results are presented in Table 2.3. For the simplified HARP pattern set, we observe that the delay improves by 21%, which is about 4% better than the improvement obtained with the full pattern set. There are no significant differences in the other metrics.

The same HARP usage analysis is also performed on the routing results with the simplified HARP pattern set and the results are shown in Figure 2.12(b). Compared to Figure 2.12(a), there are no significant changes in the usage over different HARP connections.

Besides the HARP usage analysis, we are also interested in finding out whether the final routing connection behavior is changed after introducing the HARP architecture. More specifically, we want to cross check if HARP has any effect on the connection pattern distribution compared to traditional architectures.

Figure 2.13 extends Figure 2.5(b) by adding the data obtained from the routing

Circuit	Delay ($\times 10^{-8}$)	CW	Area ($\times 10^{6}$)	Energy ($\times 10^{-8}$)
misex3	5.17	25	2.81	1.15
alu4	5.93	22.8	2.90	1.32
apex4	6.06	27.8	2.77	0.987
ex5p	5.35	28	2.36	0.877
des	6.40	18.6	5.82	2.40
seq	5.22	27	3.62	1.39
apex2	6.16	27.6	3.99	1.65
spla	8.88	35.8	9.91	4.02
pdc	9.52	43.4	1.48	6.20
ex1010	11.6	27.4	9.38	5.05
tseng	4.70	17	1.58	7.25
bigkey	3.59	17.7	4.00	1.53
s38584.1	7.12	19.3	9.96	4.66
clma	11.9	32.4	20.2	12.9
dsip	3.51	17.8	4.04	1.42
diffeq	4.80	18.4	2.25	0.863
frisc	7.74	29.8	7.95	3.15
s298	8.52	19.2	3.00	1.86
s38417	8.22	19.2	9.92	5.37
elliptic	7.39	24.8	7.02	3.36
oc_wb_dma	6.80	34	24.80	8.31
oc_mem_ctrl	7.20	23	15.30	5.92
oc_des3_des3perf	12.10	19	55.20	30.98
Geometric Mean	0.796	1.25	0.99	0.763

Table 2.3: Results for Simplified Pattern set

results on the HARP architectures. It seems that it is safe to say that HARP does not change the pattern distribution characteristics fundamentally. However, one interesting observation can be made on the | and − patterns. With HARPs (either the full set or the simplified set) the percentage of | and − connections made in the final routing graphs drops significantly (especially when compared to the timing-driven results without HARPs). In the meantime, this drop seems to be compensated mainly in the increases on different L-shaped patterns. One explanation could be that by using HARP we reduce the delay of the L-patterns in the routing structure, and as a result, using L patterns becomes less costly than using an H followed by a V. Hence, the router is encouraged to use more turns in the routing paths compared to the non-HARP architectures.

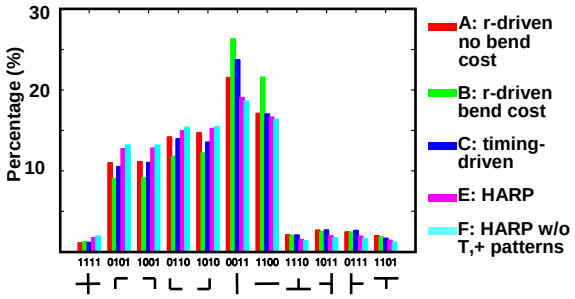


Figure 2.13: Switch box pattern distribution comparison

In the above discussion, results in HARP usage are normalized by the number of HARP resources, while results in connection pattern distribution analysis are normalized to the total number of switches utilized in the routing results. It is also interesting to see how effective the switch resources are utilized compared to the total number of switches provided by the architecture.

Table 2.4 shows the overall switch resource utilization data. In the table, *sim-harp* refers to the simplified HARP architecture (i.e. without ⊤ and + shaped HARPs),

and the usage percentages reported here are computed as:

$$percentage = \frac{\text{total \# of switches used in final routing}}{\text{total number of switches provided}} \tag{2.1}$$

Regardless of the underlying FPGA architecture, the results in Table 2.4 show that the circuits make poor utilization of the switch resources available in the FPGA. Moreover, without HARP-specific improvements on the placement and routing tools, this efficacy suffers a 7.8% $\sim$ 14.4% drop on the HARP architectures compared to a regular Virtex-II architecture.

2.6 Tile-Based Design of HARP FPGAs

One of the most complicated tasks in the design of FPGAs is the layout. Typically, manufacturers manually layout a single tile consisting of a logic block and switch block and replicate them across the entire chip. However, when HARPs are inserted randomly in the switch-boxes, they are no longer identical and a tile based layout is not possible. In this section, we present a method that facilitates layout in the presence of HARPs.

In a multi-segment architecture, the start points of different segments are staggered to enhance routability of the architecture. Figure 2.14 shows a staggered arrangement of tracks with each channel having three segments of length 3 [10].

When a disjoint switch-box topology with a staggered arrangement of segments, a tile based layout is possible when the number of tracks of a particular length is divisible by the length of the tracks. However, layout of a single tile with HARPs in the switch-box is not possible for the following reasons:

- When all the patterns are inserted in a single switch-box to enable a tile based layout, long patterns will be formed throughout the chip. This will increase the capacitive load on the segments. The track count to route circuits will also go

Circuit	x-size	y-size	Channel Width			Switch resource usage(%)		
			no-harps	harps	sim-harps	no-harps	harps	sim-harps
alu4	40	40	18	26	25	24.99	21.32	22.78
apex2	44	44	20	32	32	30.53	21.85	22.78
apex4	36	36	21	31	33	26.9	20.45	19.89
bigkey	54	54	18	18	18	14.71	16.22	17.03
clma	92	92	24	39	32	26.41	20.73	24.84
des	63	63	16	22	22	16.26	16.01	16.98
diffeq	39	39	14	20	21	22.88	19.74	18.72
dsip	54	54	14	18	18	15.84	16.31	15.66
elliptic	61	61	20	30	31	22.43	18.54	17.91
ex1010	68	68	22	32	24	24.87	20.59	26.75
ex5p	33	33	22	32	32	26.91	22.58	24.25
frisc	60	60	25	34	33	22.71	20.95	22.85
misex3	38	38	20	28	25	28.15	22.07	26.16
pdc	68	68	33	50	43	25.67	21.61	23.3
s298	44	44	18	23	22	23.46	21.19	22.58
s38417	81	81	17	23	23	18.53	16.83	17.65
s38584	81	81	16	18	19	17.91	17.58	18.22
seq	42	42	21	32	27	28.34	22.32	24.97
spla	61	61	28	43	36	24.76	19.88	23.86
tseng	33	33	14	16	17	20.94	19.51	19.72
Average			20.05	28.35	26.65	23.16	19.81	21.35

Table 2.4: How effective the switch resources are used in different architectures

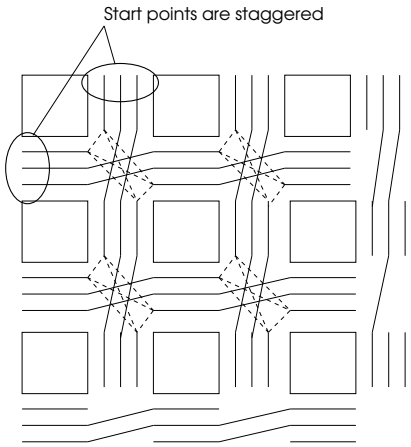


Figure 2.14: Staggered Arrangement of Tracks

up since, tracks having HARP resources will run the entire length of the chip and only one net will be able to use the track.

- There may not be enough tracks in the channel to accurately capture the distribution of various patterns. Some of the patterns have low percentages and may not appear in the switch-box.

In order to prevent long patterns from being formed in the chip, we need to look at the neighbors of every switch-point in a switch box and ensure that they do not form such patterns. This is illustrated in figure 2.15. For example, when we are looking at switch point a, we also need to consider the switch points b (top) and c (right) before deciding on the type of the switch (flexible or HARP) that can be used at a. It is to be noted that we do not have to consider the other neighbors(left and bottom) of a as they would also correspond to b and c.

This problem can be conceptually visualized as distributing HARPs in a 2×2 switch-box array so that long patterns are not formed. We solve this problem by formulating it as an Integer programming problem with constraints to prevent forming long patterns and to ensure that the distribution follows the statistical estimation

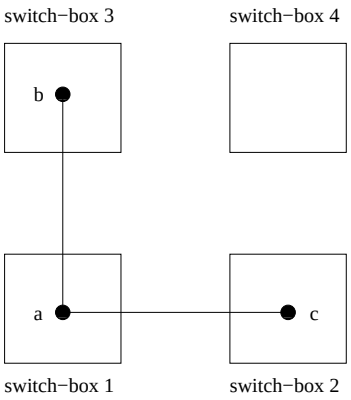


Figure 2.15: Switch point neighbors

presented in section 2.3. The rest of the section presents the formulation of the ILP. If *ntracks* is the number of tracks that require end-point connections at every switch-box, we have a total of $4 \times ntracks$ candidate switch-points in the 2×2 switch-box array. Each of them can be either be a flexible switch or a HARP resource based on the constraints. The HARP resources are labeled as shown in figure 2.16. We have not included the + pattern in this study since the percentage of +'s is very small when compared to the others.

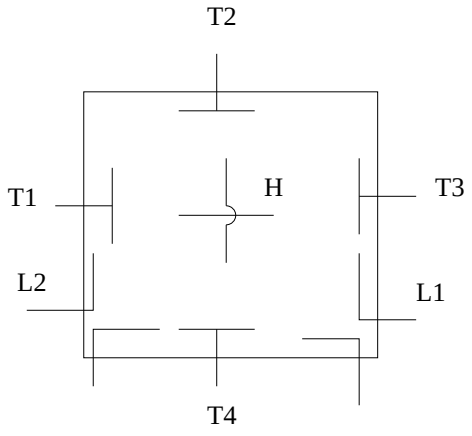


Figure 2.16: HARP labeling

The following variables in the ILP are used to represent the switches:

$x_{ij}n$: flexible,

$x_{ij}h$: horizontal and vertical HARP,

$x_{ij}l_k$ k=1,2 : L_1, L_2

$x_{ij}t_k$ $k = 1 \cdots 4$: T_1, T_2, T_3, T_4

Where i is the switch-box number and j is the track number. To maximize routing flexibility, we should ensure that flexible switches are distributed evenly between switch-boxes. To this end, we try to reduce the maximum number of flexible switches that are inserted in each switch-box. This coupled with the capacity constraints on the number of flexible switches and HARP resources will produce the right mixture of flexible and HARP resources in all switch-boxes. Thus the objective function is written as follows:

$$Minimize \quad W$$

subject to,

$$W \geq n_i \qquad i = 1 \cdots 4 \tag{2.2}$$

$$n_i = \sum_{j=1}^{ntracks} x_{ij}n \quad i = 1 \cdots 4 \tag{2.3}$$

Each of the x_{ij} variables can be either 0 or 1 depending on the type of switch present at switch box i and track j. Writing this down in the form of a constraint,

$$x_{ij}n, x_{ij}l_k, x_{ij}t_k, x_{ij}h \in \{0, 1\} \tag{2.4}$$

At every switch-point, we can have only one kind of switch. This means that exactly one of the x_{ij} variables is 1 and the rest are 0. This gives rise to the following constraint:

$$x_{ij}n + x_{ij}h + \sum_{p=1}^{2} x_{ij}l_p + \sum_{q=1}^{4} x_{ij}t_q = 1, \tag{2.5}$$

To prevent forming long horizontal and vertical patterns that run throughout the chip, restrictions are imposed on the location of HARPs inside different switch-boxes. For example, if two horizontal or vertical HARPs were to connect to each other and the patterns are replicated, it would form a long pattern throughout the chip. To avoid this, we have a constraint that prevents horizontal or vertical HARPs in adjacent switch locations. Similar to the example mentioned, there are certain restrictions on the locations of other patterns as well depending on their orientations. These are captured by the following set of constraints:

$$\begin{aligned} x_{ij}h + x_{targetj}h &\leq 1, \quad \forall \quad i,j \qquad (2.6) \\ if\ i = 4 \quad target &= 1, \\ else\ target &= i+1 \\ x_{pj}t_p + x_{qj}t_q &\leq 1 \qquad (2.7) \\ x_{pj}t_p + x_{qj}h &\leq 1 \\ x_{pj}h + x_{qj}t_p &\leq 1 \end{aligned}$$

$$(p,q) \in \{(1,2),(2,3),(3,4),(4,1)\}$$

To prevent forming loops with HARPs, we impose a constraint that for any HARP resource, at least one of its two neighbors is a flexible switch. This can be written as

$$x_{pj}n + x_{qj}n + x_{rj}n \geq 1 \qquad (2.8)$$

$$(p,q,r) \in \{(1,2,4),(2,1,3),(2,3,4),(3,4,1)\}$$

Finally, we need to add capacity constraints on the number of various patterns that

need to be present based on the required distribution.

$$\begin{aligned}
\sum_{i=1}^{4}\sum_{j=1}^{ntracks} x_{ij}h &= H_{required} \\
\sum_{i=1}^{4}\sum_{j=1}^{ntracks} x_{ij}l_k &= L_{k\ required} \quad k = 1, 2 \\
\sum_{i=1}^{4}\sum_{j=1}^{ntracks} x_{ij}t_k &= T_{k\ required} \quad k = 1, \cdots 4
\end{aligned} \tag{2.9}$$

The solution of the above ILP problem gives the location of the HARPs in the switch-boxes. It is to be noted that for segments that span C logic blocks, the neighbors(x_{ij}'s) correspond to switch-boxes that are spaced C units apart. This information is used to design a limited number of switch-boxes which are then replicated throughout the chip.

2.6.1 Results

We placed and routed benchmark circuits on the architecture generated using the solution to the ILP formulation and present the results in Table 2.5. We observe that when HARPs are distributed in switch-boxes and replicated throughout the chip, the performance worsens when compared with the case when they are distributed randomly.

When compared with the traditional architecture, delay improves by the same amount as that obtained with random distribution of HARPs inside switch-boxes. However, we observe an 6% increase in the area. This is attributed to an increase in channel width. Even though we avoid forming long patterns inside switch-boxes, distributing HARPs inside a few switch-boxes and replicating them restricts the freedom of the router. When patterns are distributed randomly, the router has different options at every switch-box and this provides more flexibility. With random distri-

Circuit	Delay ($\times 10^{-8}$)	Area ($\times 10^{6}$)	CW	Energy ($\times 10^{-8}$)
misex3	4.90	3.15	28	1.2
alu4	5.91	3.13	24	1.41
apex4	6.04	2.62	26	1.03
ex5p	6.52	2.68	29	1.00
des	5.95	6.32	20	2.41
seq	5.27	4.22	30	1.54
apex2	5.48	4.03	30	1.86
spla	8.06	11.7	43	4.32
pdc	1.25	16.1	45	8.25
ex1010	1.35	10.4	29	6.63
oc_wb_dma	7.45	24.9	34	8.88
oc_mem_ctrl	7.91	16.18	24	6.425
oc_des_des3perf	12.78	60.95	21	31.55
Geometric Mean	**0.828**	**1.06**	**1.35**	**0.82**

Table 2.5: Results of ILP formulation

bution, we observed a slight reduction in the area in spite of the increase in channel width since the average area of each switch-box reduced due to HARPs. In this case however, the increase in channel width offsets this and a net increase in area is observed. Though the area increases by about 6%, area-delay product which is a useful metric for comparing different architectures reduces by about 13% and the overall energy consumption reduces by 18%. This coupled with the fact that switch-box layout is not an issue anymore makes HARP an attractive design choice.

The results we have reported in Table 2.5 are probably a bit optimistic. Since the area increases, the routing segments would have to travel longer distances and their increased parasitics would have a negative impact on the signal delays that are mapped to them. Due to our limited resources we did not attempt laying out the HARP FPGA and hence cannot report accurate delay and area numbers in this work.

2.7 Summary

We propose a technique to reduce circuit delay, area and power dissipation by introducing hard-wired patterns inside switch boxes. The population of the HARPs is guided by statistical analysis of routing trees that are generated on a traditional architecture by the VPR tool. We analyzed the routing profiles of various circuit benchmarks and came up with a statistical measure of the routing patterns present inside the switch boxes. The routing graph construction of VPR was modified to include these patterns. Simulation results after detailed routing showed a potential improvement of 20% in circuit delay, 5% in the circuit area, 33% in the leakage power dissipation and about 6% in the total power dissipation. We observed that by introducing hardwired patterns, we can considerably speed up the circuit and at the same time achieve reasonable savings in circuit area and power dissipation.

In Section 2.3.1 we mentioned that the placement and routing algorithm and the

architecture will affect the outcome of the statistical analysis. In Section 2.3.2 we showed that the architecture has a bigger role compared to the routing algorithm. But nevertheless, both the physical design algorithms and the architecture will skew the pattern frequency analysis. Apart from the fact that there is a certain degree of inevitability in this influence, we argue that such effect could be considered useful. We would like to generate the architecture with an eye on the CAD algorithms (Eg. would the results change if we use an alignment based placement method such as [47]?). If we create an architecture that conforms to the behavior of the placement and routing algorithms, the potential benefits will be greater.

Further work is needed in making the placer aware of the changes in the routing architecture. We also need to look at the possibility of modifying Steiner tree routing algorithms to make full use of the hard-wired patterns and to achieve better correlation between the placement tool, routing tool and the routing architecture.

Chapter 3

Statistical Analysis and Process Variation Aware Routing and Skew Assignment for FPGAs

3.1 Introduction

As transistor scaling moves deeper into the sub 90nm regime, the impact of process variations on the performance of digital circuits is fast becoming a critical aspect to consider during design. Failure to do so would result in large numbers of chips failing the timing/power requirements or even causing circuits to fail completely. Speed-binning is an option for FPGA and micro-processor chips to sell slower chips at lower prices instead of discarding them completely, but this still incurs a loss compared to a method that places more chips in higher speed bins. To alleviate the problems due to variability, ASIC designers have been using statistical optimization techniques for a long time to optimize power and performance in the presence of variations [20, 30, 57]. Traditionally, the FPGA community has for the most part ignored the effects of variations since FPGA chips generally run much slower than ASICs and variation in clock frequency is a much smaller component than in ASICs. Secondly, the FPGA architecture is highly regular and as a result the impact of variability is less accentuated. As FPGAs are becoming faster, the effects of process variation can no longer be ignored. Recently there have been a number of efforts in the FPGA community that consider the impact of process variations [44, 45, 65, 74]. In [74], the authors performed in-depth architecture evaluation and studied the

impact of variations on timing and leakage yield. [45] proposes a variation aware placement algorithm to improve the timing yield of FPGAs. The authors also study the impact of speed-binning and different guard-band factors on timing yield. They extend this work in [44] and present a stochastic synthesis flow for FPGAs which considers interconnect uncertainty in the clustering stage in addition to considering timing uncertainty in physical design.

We divide this work into three parts. First, we make a comparative study of the effects of process variation on designs mapped on FPGA and ASICs to gauge the difference in its impact on the two implementation platforms. The motivation behind this study is to stimulate discussion about whether process variations have as much impact on FPGAs as ASICs. If the impact on FPGAs is less severe, designers can afford to continue to use purely deterministic optimization algorithms for some more time without having to spend valuable design time on statistical optimization techniques that are slower. However, as we continue to aggressively scale down technology nodes, statistical optimization techniques will eventually have to be adopted. In the next two parts of this chapter, we propose CAD and architectural techniques that can be used to alleviate the impact of variability.

In the second part of the chapter, we present a variation-aware router to improve timing yield. Experimental results show that the router reduces the timing yield loss by about 7.61x for 20 of the largest MCNC benchmarks and for the same yield, it reduces the circuit delay by 3.95% for a cluster size of 1. We presented these results in [65]. We performed more experiments with a different cluster size to study its effect on the delay distribution of circuits. We also performed speed-binning experiments to study the effect of variation aware routing on the number of chips in various speed bins. In addition, since the timing yield of circuits depends on clock period specifications, we also performed experiments with different cutoff-frequencies

to evaluate the effectiveness of the technique.

In the third part of the chapter, we investigate the applicability of time-borrowing or cycle stealing to further improve timing yield of sequential circuits where surplus timing slack across flip-flop (FF) boundaries can be used to fix timing violations in other parts. To achieve this, we need a tunable clock architecture where the clock skews at FFs can be adjusted to compensate for variations. This problem is a little different in FPGAs than ASICs because FPGAs have prefabricated logic and interconnects which can be configured based on specific needs of an application. Furthermore, FPGA clocks are not designed with an H-tree architecture due to problems in meshing it with a tiled layout. Due to these specific requirements of FPGAs, techniques developed for ASICs such as [71], [15] cannot be directly applied. There has been prior work in the FPGA community regarding skew assignment. In [63], the authors used several global clocks to deliver skews to various FFs. This approach incurs substantial power and routing overhead. In [78], the authors proposed a clock architecture with a global H-tree and local spine and ribs and inserted programmable delay elements (PDEs) on alternate branch points of the H-tree. In this work, we propose a skew distribution architecture and provide two skew assignment schemes to robustly assign skew in the presence of variations. As in the second part of the chapter, we perform speed-binning experiments. We study the effectiveness of the technique with conservative and aggressive timing constraints. Finally, we combine the variation-aware router with skew assignment to determine the overall improvement in timing yield that can be achieved for various timing specifications. We also study how much each part individually contributes to improving timing yield. Furthermore, we study the potential improvement in critical path delay that can be achieved by using the techniques described in this chapter when we fix our timing yield requirement at 99%.

The rest of the chapter is organized as follows. Section 3.2 presents information about modeling delay and process variations. The comparative study between designs mapped on FPGAs and ASICs is presented in Section 3.3. Section 3.4 presents our variation-aware router and discusses some results. Section 3.5.1 presents our skew distribution architecture and architecture-exploration studies. We present our skew assignment schemes in Section 3.5. We presented sections 3.5.1 and 3.5 of this work in [64]. We have performed experiments to compare the impact of a purely deterministic clock scheduling scheme with our statistical skew assignment technique. This study helps us to better understand the individual contributions of the architectural modifications and the variation aware CAD schemes in improving the timing yield. Section 3.6 presents results from combining the variation-aware router with the skew assignment technique and discusses about their impact. Section 3.7 discusses the overhead associated with our proposed techniques. Finally, Section 3.8 gives a brief summary of this chapter.

3.2 Variation and Delay Modeling

In this section, we present our process variation and delay models. We modeled all variations as spatially correlated gaussian distributions. We consider variations in the transistor length (L_{eff}), width (W_{eff}) and interconnect width (W_{int}) and thickness (T_{int}). Apart from the parameters we consider, there are truly random sources of variation such as oxide thickness, T_{ox} and dopant concentration, N_D and the variation models presented here can be easily extended to include them. L_{eff} is the dominant component of variation in devices, so we believe that ignoring the effects of T_{ox} and N_D will not significantly alter our results [23]. We use a 65nm predictive technology model [9] in this work. The framework presented here can be extended to handle any number of device parameters with only minor modifications. Since we did not have

access to real foundry data, we adapted the process parameters from [16]. They are shown in Table 3.1. Here μ refers to the mean and σ refers to the standard deviation

Table 3.1: Process Parameters

Parameters	L_{eff} (nm)	W_{eff} (nm)	W_{int} (nm)	T_{int} (nm)
μ	60.0	150.0	150.0	500.0
$3\sigma_{inter}$	9.0	11.25	15.0	25.0
$3\sigma_{intra}$	4.5	5.625	7.5	12.5

of the process parameters. The inter and intra subscripts refer to the inter and intra die component of the variation. The total variation is given by

$$X_{total} = X_{inter} + X_{intra} \tag{3.1}$$

Where X_{inter} refers to the inter-die component and X_{intra} refers to the intra die variation and $X \in \{L_{eff}, W_{eff}, W_{int}, T_{int}\}$. The total variations at the 3σ process corners of L_{eff}, W_{eff}, W_{int} and T_{int} are approximately 15%, 10%, 15% and 6% respectively. The inter-die component of variation affects all devices on a chip identically. The intra-die component on the other hand is spatially correlated. Devices located close to each other tend to exhibit similar characteristics. To capture the effects of spatial correlation, we use the approach proposed in [28]. The authors model the correlation coefficient as a piecewise linear function which starts off high and then decreases as the distance of separation increases until it reaches a certain critical distance beyond which it remains a constant. It is to be noted that the methods presented in this work are applicable irrespective of the exact correlation models used. We divide the chip into several grids and assume that process parameters are all perfectly correlated within a grid. Each grid has its own set of random variables for all parameters and

the correlations between parameters in different grids are given by the correlation modeling in [28]. Further, as in [16], we make an assumption that different types of process parameters such as L_{eff}, W_{eff} etc. are independent of each other. We first build co-variance matrices for all parameters with the intra-die component of variation. The inter-die component of variation is added to the entries of this co-variance matrix. When a piecewise linear correlation function is used, the covariance matrices obtained may not always be positive semi-definite, which is a requirement for using the principal component analysis technique (PCA). To get over this problem we use the approach presented in [59] to obtain the nearest valid covariance matrix. We then use the principal component analysis technique (PCA) to convert these correlated random variables into a set of mutually independent standard-normal random variables.

As in [16], we express the delay of a circuit element using a first order Taylor series expansion around the nominal point. Delay is expressed as

$$d = d_0 + \sum_{\forall p_i} S_{i_0} \Delta p_i \tag{3.2}$$

where d_0 is the nominal delay of the circuit element and the S_i's are the sensitivities of different process parameters to the delay at the nominal point. We obtain sensitivities from HSPICE simulations. The p_i's denote various process parameters. We use PCA to convert the correlated p_i's to independent components and then substitute them instead of p_i's in Equation 3.2. The expression for delay now becomes

$$d = d_0 + \sum_{\forall p_i} S_{i_0} \sum_{j=1}^{m} a_{ij} p'_j \tag{3.3}$$

where m is the number of grids into which the chip is divided. The p'_j variables are mutually independent standard-normal variables and $a'_{ij}s$ denote the coefficients obtained from PCA. We model interconnect delays in the same way as [16]. Equation 3.3 is the canonical expression for delay that we use in this work.

3.3 Impact of Process Variations: FPGAs Vs ASICs

ASIC designers have been worrying about the effects of process variations for a long time since ASICs operate at much higher clock frequencies where even small changes in timing due to variations can be a significant part of the cycle time. There are also several design-for-manufacturing issues due to the irregular nature of circuits and these also exacerbate the effects of variations. Hence they have started migrating from traditional corner-based analysis techniques to statistical optimization techniques. FPGA researchers on the other hand have only recently started to study the effects of process variation. In this section, we compare the impact of process variation on timing for designs mapped on FPGAs and ASICs by using block-based statistical static timing analysis (SSTA). This is to understand whether we have reached the stage where statistical analysis is as critical to FPGAs as for ASICs. The objective of this study is to find out whether FPGAs and ASICs behave differently when subjected to variations. In this study, for the ASIC platform we use a 65m standard cell library from the Open Access tool kit [54]. We used a 65nm SRAM based FPGA. All the tools that we used in this work are freely available to the research community. We have tried our best to optimize both our FPGA and ASIC design flows. In the research community, people typically use the ISCAS89 benchmarks for ASICs and the Toronto-20 subset of the MCNC benchmarks for FPGAs. To ensure fairness, we decided to use both the ISCAS89 and the MCNC benchmarks for both our design flows. We provide more details about our design flows in the next section.

3.3.1 Design Flows

In this work, we use an island style FPGA architecture with 4-input lookup tables (LUTs). The routing architecture consists of 10% length-1, 25% length 2 and 65% length-6 wires with buffered switches. We experimented with cluster sizes of 1 and

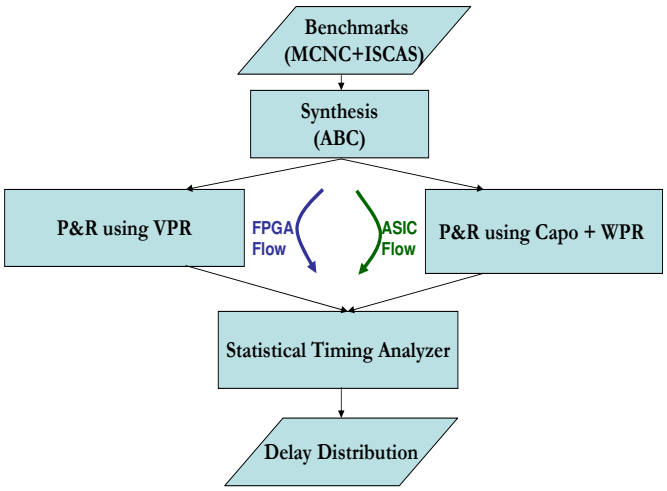


Figure 3.1: Design Flows

4. We used a subset switch-box topology. Synthesized benchmark circuits are first mapped to the architecture and then placed and routed using VPR [10] in the timing-driven mode. To alleviate routing congestion, the circuits were routed with 10% more tracks than the minimum channel width. Once we are done with placement and routing, we use our statistical static timing analyzer to obtain delay distributions at various nodes in each circuit. For the ASIC flow, we synthesize and map benchmark circuits using ABC [50] and then use Capo [14] in the timing driven mode to place the design. We then use the buffer driven router in [6] to perform routing. Finally, a statistical static timing analyzer is used to compute the delay distribution. This is adapted from the flow used in [16]. Figure 3.1 illustrates our FPGA and ASIC design flows. Ideally, to compare the impact of process variations on FPGAs and ASICs, the same tools should be used for both implementation platforms to remove any bias introduced by the tools. In practice however, different tools are used for FPGAs and ASICs and our goal in this study is to see the effect of variations on real designs; so for each platform we use its own set of tools. We do not perform any statistical

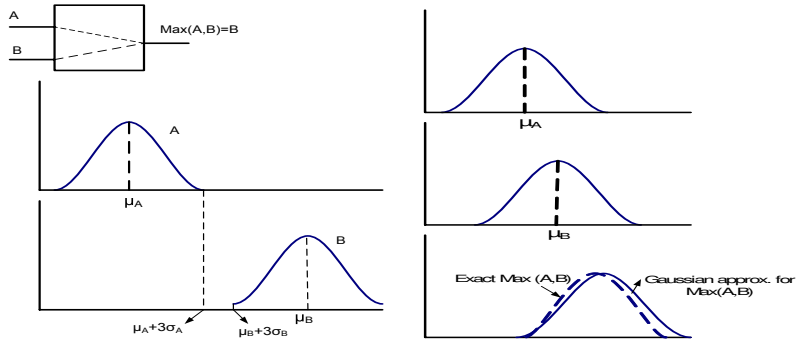
optimization for FPGAs or ASICs in this section since we are interested in observing the impact of ignoring the effect of variations on the accuracy of delay prediction and ultimately its impact on the design flow. We present statistical optimization techniques for FPGAs in Sections 3.4 and 3.5.1.

3.3.2 Comparative Studies

Statistical static timing analysis(SSTA) is used instead of deterministic static timing analysis[1](STA) primarily due to the inability of STA to accurately predict the arrival times at various nodes in the circuit caused by variability. This means that STA cannot accurately predict the paths in the circuit that may cause timing violations in the presence of variations. Extending this further to the granularity of logic blocks, this translates to the inability to predict the latest arriving input that determines the output arrival time. Hence, in an SSTA the basic operations involved in timing analysis such as the sum and max operations are replaced by their statistical counterparts and in the statistical max operation, the maximum of two normal random variables is approximated to be another normal random variable [22]. Since arrival times are random variables with a certain mean (μ) and standard deviation (σ), estimating the output arrival times by just considering the mean delays (STA) would lead to erroneous results if the means of the input arrival times are *sufficiently close*. We say that two random variables $X_1(\mu_1, \sigma_1)$ and $X_2(\mu_2, \sigma_2)$ are sufficiently close if $\mid \mu_1 - \mu_2 \mid < 3\sigma_1 + 3\sigma_2$. On the other hand, if the $\mu + 3\sigma$ point of the delay of the first input is lower than the $\mu - 3\sigma$ point of the delay of the second, there is no need to compute the statistical max operation. We can estimate the max simply as the second input. We call this the *pruning* of the max operation and set the *pruning threshold*

[1]From here on we will refer to deterministic and statistical timing analysis as STA and SSTA respectively

to 3σ [16]. This is illustrated in Figure 3.2(a). The figure shows the arrival-time



(a) Large Delay difference (b) Overlapping delays

Figure 3.2: Finding Max Delays

distributions of two inputs of a logic block. Input B completely dominates input A. The arrival time distribution at the output is thus determined only by B and there is no contribution from A. We can determine the arrival time at the output just by performing a statistical sum of the arrival time at B and the delay of the logic block itself. In this scenario, we can predict the input responsible for setting the output arrival time even with variability. On the other hand, when there is overlap in the arrival-time distributions of inputs as shown in Figure 3.2(b), there is contribution from both inputs to the statistical max and the resulting max is approximated as a gaussian using Clark's formulae [22]. As a first step in our studies, we compare FPGAs and ASICs and observe the amount of pruning in each case. The motivation here is to find out whether pruning occurs more frequently in FPGAs when compared to ASICs. If this is the case, then we can do a much better job for FPGAs at predicting the inputs responsible for setting output arrival times just by comparing the means of the input arrival times. This is because for instances where pruning occurs, the deterministic and statistical max both correspond to the same input. To answer these questions, we pass benchmark circuits through the flows described in

Figure 3.1 and the results are shown in Figure 3.3. The figure shows the average

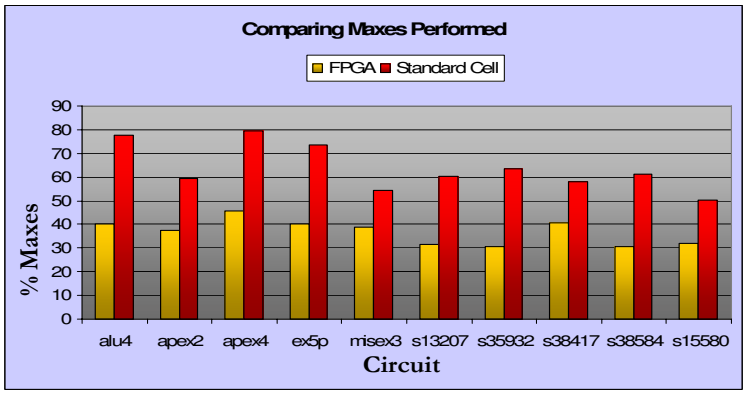


Figure 3.3: Comparison of Maxes performed

percentage of statistical max computations performed for FPGA and standard cell implementations for each benchmark circuit. For instance, if the percentage of max computations is 30%, it means that out of all attempted max operations 70% are pruned out due to the situation illustrated in Figure 3.2(a) and only 30% of the computations are performed. From the figure it is clear that the percentage of max operations performed for standard cells is significantly higher than FPGAs. On average, statistical max operation is performed only about 37% of the time for FPGA designs. This number however is about 63% for ASICs. This indicates that inputs to logic blocks are much closer in ASICs than FPGAs. The high amount of pruning in FPGAs suggests that for the most part, we may be able to deterministically predict inputs that are responsible for setting output arrival times even in the presence of variations. Though observing the amount of pruning in FPGAs and ASICs gives us valuable information about how they behave under the effect of variations, it is not fully accurate. This is because a closed-form solution exists only for the maximum of two random variables and as a result, we would have to make several pair-wise comparisons for logic blocks that have more than two inputs. Hence the order in

which max operations are performed becomes important since it determines the total number of max computations as well as the accuracy of the output distribution. For instance, consider the situation illustrated in Figure 3.4. The LUT in the figure has

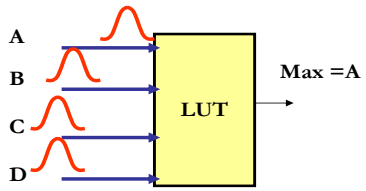


Figure 3.4: Order of Max Computations

four inputs (A,B,C,D). If we consider input A first, then we would not have to perform any max computation since A completely dominates the rest of the inputs. On the other hand, if we consider inputs B, C and D before A, we need to perform two max computations namely max(C,D) and max(B,max(C,D)). Furthermore, since the max of two gaussian random variables is only approximately gaussian, statistical max operations introduce errors and this may progressively push the delay distribution of the current max closer to A and may lead to errors in estimating max(A,B,C,D) if we consider D,C and B before A. We can prune out these unnecessary max operations with a pair-wise comparison of all inputs [52]. Due to this limitation, Figure 3.3 overestimates the average number of max computations actually needed. Since we are more interested in FPGAs, we performed more studies to gather information about tightness probabilities of the inputs of logic blocks since observing tightness probabilities will reduce the inaccuracies with counting the number of max computations. Tightness probability was defined in [72]. If we have two random variables A(N(μ_A,σ_A)) and B(N(μ_B,σ_B)), the tightness probability of A (T_A) is the probability that $A > B$. Analytical expressions for computing tightness probability are given in [72]. Given a large number of random variables, each has a tightness probability

corresponding to the probability that it is larger than the rest of the variables. We can get a better insight at the contribution of different LUT inputs to the output arrival times by observing the tightness probabilities of the inputs. We show the results of our analysis in Figure 3.5. We computed the tightness probabilities (TPs)

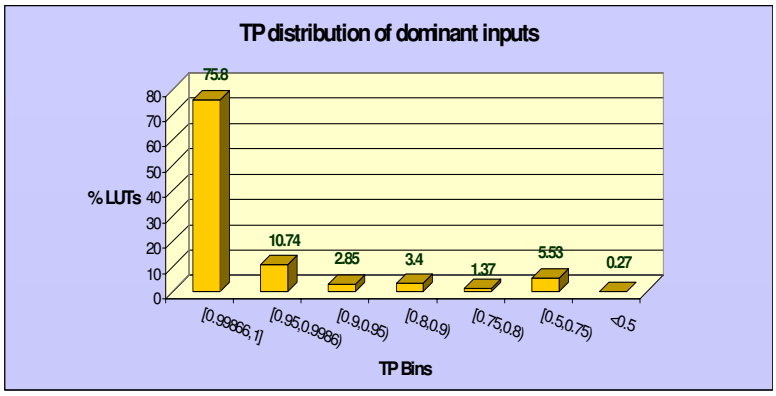


Figure 3.5: Binning Tightness Probabilities

of all inputs to a LUT and defined the dominant input to a LUT to be the one with the maximum TP. The statistics shown in the figure are average numbers over all benchmark circuits. The figure shows the histogram of the LUTs that are binned according to the TP of the dominant input. The X-axis shows the various TP bins that we considered and the Y-axis shows the percentage of LUTs that have the TP of their dominant inputs in the respective TP bins. For instance, on average over all benchmarks, about 76% of all LUTs fall in the TP bin labeled [0.99866,1]. This means that for 76% of LUTs, a single input dominates the output arrival time when the pruning threshold is 3σ. This is a significant result since higher tightness probabilities indicate lower sensitivities to variations. Thus, if an LUT has an input with a tightness probability of [0.99866,1], it means that even with variations, the same input continues to be responsible for setting the output arrival time for the most part and thus no statistical max operation is required. From Figure 3.5, we observe that

in most cases (76%), variations do not change the longest path through LUTs. Furthermore, some of the max computations performed are redundant and are performed only due to sub-optimal variable ordering.

We believe that two reasons contribute to the fact that in the majority of logic blocks in FPGAs a single input dominates all other inputs. First, different routes pass through different numbers of programmable switching resources and this causes the means to be well separated because of discrete jumps in delay. If the means are well separated, then the impact of variations decreases. Secondly due to the slow nature of the programmable interconnects and logic blocks in FPGAs, the fraction of delay change due to variations is insignificant when compared to the delay at the nominal process conditions. In ASICs, this fraction is likely to be more since they are inherently much faster than FPGAs and consequently any change in delay would be more significant in ASICs than FPGAs. This discussion further illustrates that we may be able to do a reasonably good job of deterministically predicting inputs responsible for setting output arrival times in FPGAs. However, TPs provide information only about individual LUTs and not paths in circuits. Analyzing different paths in circuits would provide more detailed information about the extent to which critical and near-critical paths are affected by variations. This leads us to perform more studies on criticalities of different paths in the presence of variations. Another aspect to consider here is the impact of cluster size on delay variations in different paths. When a smaller cluster size is used, there is more inter-cluster routing which is typically slower than routing within a cluster and this might have an effect on how variations affect different paths. In our critical path studies, we present results with cluster sizes of 1 and 4.

To perform critical path analysis, we used the flow illustrated in Figure 3.6. We first enumerate the top 10 critical paths of circuits by performing a deterministic

timing analysis with all parameters set to their nominal values. We then perform Monte Carlo simulations with 10,000 samples and then enumerate the top 10 statistically most critical paths. We compute the statistical criticality of a net as the ratio of the number of times the net was on a critical path while performing Monte Carlo simulations to the total number of trials. Statistical criticalities of nodes are computed in a similar fashion.

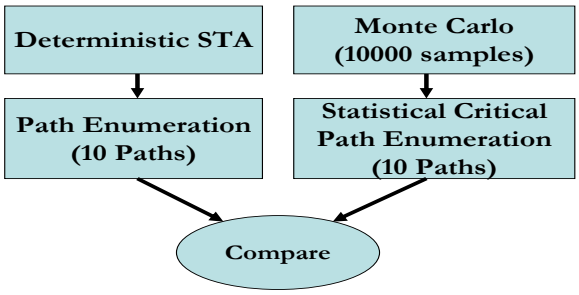


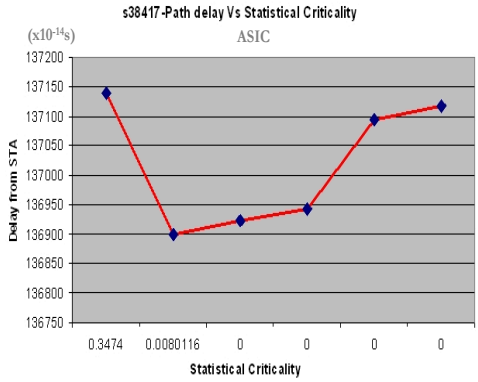
Figure 3.6: Critical Path Analysis

Statistically critical paths are then enumerated using the technique proposed in [72]. If all the paths in a circuit are considered, the critical path coverage is 1, meaning that at any point in the process space the critical path will be among those paths that we consider. However, the total number of paths in a circuit can potentially be exponential in terms of the circuit size. Hence we settled for the 10 most statistically critical paths since they correspond to a critical path coverage of 0.9999 for our benchmarks and process parameters i.e. if we have 10,000 chips then in 9999 of those, the critical path is among the set of paths that we consider. The number of paths to be considered is dependent on the amount of variation. If the sigma chosen is very high then more paths need to be considered to achieve reasonable path coverage. On the other hand if the sigma is low, fewer paths may be sufficient. The motivation behind this study was to check whether the paths that were deemed critical by STA also had a high criticality probability under the effect of variations. If the same paths

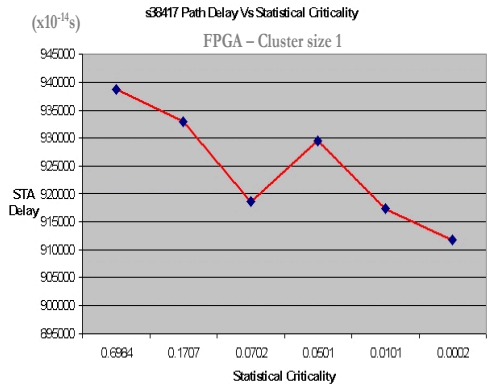
are reported as critical by both STA and Monte Carlo, then we can conclude that even in the presence of variations STA predicts critical paths with high accuracy.

We performed these studies for both MCNC and ISCAS89 benchmark circuits. The results of a benchmark, s38417 are shown in figure 3.7. The results of other MCNC and ISCAS89 benchmarks were similar. The y-axis shows the path delays obtained from deterministic static timing analysis and the x-axis shows the path criticalities of the same paths obtained from Monte Carlo simulation. Figures 3.7(b) and 3.7(c) are the results for FPGAs with cluster sizes of 1 and 4 respectively and 3.7(a) gives the path analysis results for ASICs. The figures show a marked difference in the way FPGA and ASICs are affected by variations.

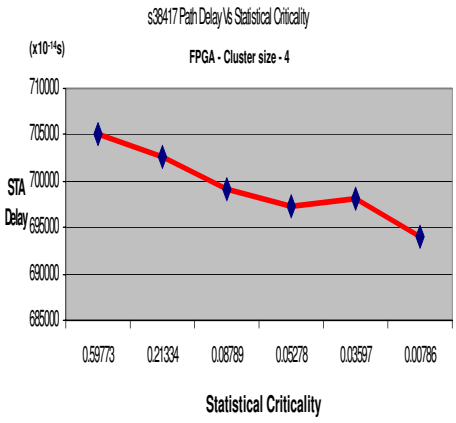
For the FPGA implementation, the top 6 paths reported by STA were also among the top 6 paths reported by Monte Carlo. We observe similar results for both cluster sizes. The top 2 critical paths as identified by deterministic STA had a path coverage of 0.8691 with a cluster size of 1 and 0.811 with a cluster size of 4. The cumulative critical path coverage (sum of $X - axis$ values) of the paths identified by STA was 0.9997 for a cluster size of 1 and 0.9955 for a cluster size of 4. This indicates that STA does a good job of predicting critical paths and we can optimize the circuit reasonably well if we consider the critical and top few near-critical paths given by STA rather than concentrating only on the most critical path. On the other hand, when we look at the ASIC implementation, the corresponding critical path coverage was only 0.3554. This number is much lower than the FPGA implementation. This indicates that for ASICs, STA does a poor job of identifying critical paths and the impact of variations on ASICs are much higher and hence statistical optimization techniques are more critical to ASICs than FPGAs. One thing to note here is that different sets of CAD tools are used for the FPGA and ASIC design flows. As a result, the studies show an interraction between the CAD tools and the underlying



(a) Path Criticalities for S38417,
$\sum Path\_Criticality = 0.3554$



(b) FPGA Path Criticalities for S38417,
$\sum Path\_Criticality = 0.9997$



(c) FPGA Path Criticalities for S38417,
$\sum Path\_Criticality = 0.9955$

Figure 3.7: Path Criticalities of FPGA Vs. Standard-Cell Designs

architecture and the results may have a little bias caused by the tools. However, we believe that the studies are useful since each design platform is used with its own set of design tools in practice and our studies will give valuable insight into how variation affects each platform.

Though the discussion in this section makes a case that statistical optimization is more critical for ASICs, we do not discount the importance of using it for FPGAs.

From our studies it is clear that we have not yet reached the stage when statistical techniques are as critical for FPGAs as they are for ASICs. This gives us more time to explore various architectures and CAD techniques to address the problems associated with variations as they will have a more substantial effect in future technologies. In the rest of this chapter, we investigate CAD and architecture techniques to combat the effect of variations.

3.4 Variation-Aware Router

In this section we present a variation-aware router to optimize timing yield of FPGAs. We modify the timing driven router in [10] to optimize statistical criticalities of nets instead of their deterministic criticalities.

VPR uses a negotiated congestion-delay algorithm where timing critical nets are given a high priority to use the fastest routing resources available and less critical nets are forced to take detours if necessary to alleviate congestion. The routing process is iterative and every net is ripped up and re-routed after every routing iteration. This process continues until there are no over-congested regions. To minimize delay, all nets are marked as critical initially. After every routing iteration, timing analysis is performed to compute criticalities of nets. The cost function that the router tries to optimize consists of a criticality component and a congestion component. The congestion cost of routing resources are also updated after every routing iteration. The cost of using a routing resource n, as a part of routing a net (i,j) is given by [10]

$$Cost(n) = Crit(i,j)delay(n) + [1 - (Crit(i,j)]b(n) \cdot h(n) \cdot p(n) \tag{3.4}$$

where *Crit(i,j)* is the criticality of the net, *delay(n)* is the delay of routing resource n. The second term in the cost function captures the effect of congestion. *b(n)* is the base cost of using a resource and is set to *delay(n)* and *h(n)* is the historic congestion factor

and keeps track of the congestion levels of the resource in previous iterations and *p(n)* monitors the congestion in the present iteration. To perform timing optimization in the presence of variations, we replace the criticality component in Equation 3.4 with statistical criticality. The cost function for using a routing resource n, now becomes

$$Cost(n) = Stat\_Crit(i,j)delay(n) + [1 - (Stat\_Crit(i,j)]b(n) \cdot h(n) \cdot p(n) \qquad (3.5)$$

For the first routing iteration, we set the criticality of all nets to 1.0. After the first routing iteration, we traverse the routing trees of all the nets and compute the expressions for the delays in the canonical form shown in Equation 3.3. We then perform SSTA and compute the arrival and required tightness probabilities. We use the criticality computation technique in [72] to compute the statistical criticalities of nets. For subsequent routing iterations, we use the cost function shown in Equation 3.5. The pseudo-code for our routing algorithm is shown in Figure 1 and is adapted from [10].

3.4.1 Experimental Setup

In this section, we present the experimental setup that we used to validate the benefits of using a variation aware router. The flow that we used is shown in Figure 3.8. For the deterministic flow, we adopt the standard practice of guard-banding the individual circuit elements with their 3σ delay values. The timing-yield with guard-banding depends on the value of the guard-band factor. Higher guard-band factors result in higher timing yields but this comes with a performance penalty since guard-banding imposes an extra margin on the critical path. This trade-off has been explored in [45] and we use the 3σ value since it gives the best yield in [45]. As in [45] we define the yield loss as the number of chips that fail timing specifications out of 10,000 chips. Once we obtain the delay distribution by a statistical static timing analysis, we compute the yield loss by finding the number of chips whose delays

Algorithm 1 Variation Aware Router

```
1: procedure VariationAwareRouter
2:     Stat_Crit(i,j) ← 1.0 ∀ nets i and sinks j
3:     while overused resources exist do
4:         for each net,i do
5:             Rip-up routing tree of net i
6:             Update affected components in Equation (3.5)
7:             for each sink j, of net(i) in decreasing Stat_Crit(i,j) order do
8:                 Find the least cost route of for sink
9:                 ∀ nodes in the path from i to j, Update affected Components in
   Equation (3.5)
10:                ∀ nodes in the path from i to j, Update Elmore delay of the route
11:            end for
12:        end for
13:        Update Historic_Congestion()
14:        Compute_NetDelay_mean_and_variance()
15:        Compute_Arrival_Tightness_Probabilities()
16:        Compute_Required_Tightness_Probabilities()
17:        Compute_Statistical_Criticalities()
18:        Update_Stat_Crit(i,j) ∀ i,j
19:    end while
20: end procedure
```

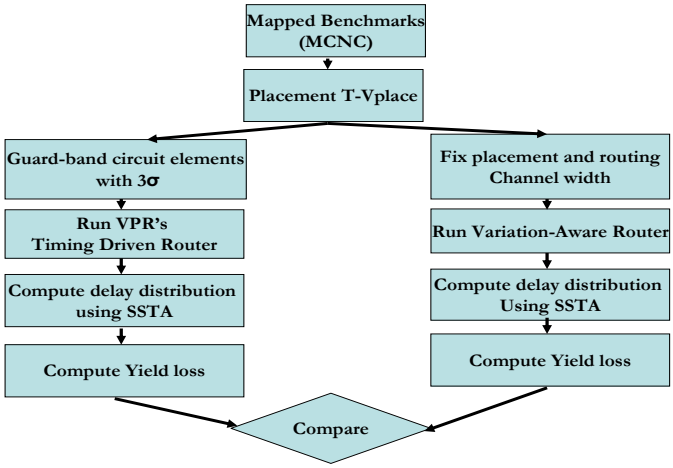


Figure 3.8: Experimental Setup

are greater than the timing specification. We first generate placements with VPR operated in its deterministic timing driven mode and then use the same placement for both the deterministic and variation-aware flows. We use 110% of the minimum routing channel width required and use the same number of tracks in both cases. We present our results and discussion in the next section.

3.4.2 Results and Discussion

In this section we first present the results of the experimental flow presented in Figure 3.8. Table 3.2 gives the effect of running VPR's timing driven router and our variation aware router on 20 MCNC benchmarks with a cluster size of 1. Once we have the delay distributions from both flows, we compute the yield loss in both cases with the 3σ guard-banded circuit delay as the cutoff delay. In the table, the column labeled T_{gb} gives the guard-banded circuit delay. Columns 3 and 4 of the table give the mean and standard deviation of circuit delay using the deterministic flow. Columns 6 and 7 give these numbers for our variation-aware router. The yield loss reported in columns 5 and 8 are the number of chips that fail out of 10,000 chips with the delay reported

Circuit	T_{gb}	VPR			Variation Aware Router			Y_l	ϕ(%)
	(ps)	μ(ps)	σ(ps)	Y_l	μ(ps)	σ(ps)	Y_l	Imp	Imp
ex5p	12144.80	9034.83	979.54	7.49	8754.44	869.81	0.33	22.70 x	6.58
alu4	12052.90	9203.27	913.50	9.06	9193.98	861.79	4.54	1.99 x	1.50
misex3	12057.00	9122.83	957.92	10.95	8728.27	879.07	0.76	14.41 x	5.76
apex2	12354.20	9668.55	851.08	8.01	9161.15	880.68	1.44	5.56 x	3.24
apex4	11259.40	8171.35	964.10	6.80	8297.57	845.53	2.30	2.96 x	2.57
pdc	20223.70	15347.10	1547.15	8.11	14596.10	1492.22	0.81	10.01 x	4.74
seq	11429.60	8991.70	763.89	7.08	8226.58	831.44	0.60	11.80 x	4.37
des	11403.54	8940.83	788.36	8.93	8107.74	898.94	1.23	7.26 x	3.73
spla	22604.60	17358.80	1690.36	9.57	16373.40	1662.72	0.89	10.75 x	4.82
ex1010	19595.50	14868.50	1354.97	2.43	14658.80	1315.03	0.87	2.79 x	1.83
frisc	16877.60	12547.50	1333.58	5.83	12283.90	1273.81	1.55	3.76 x	2.84
elliptic	14467.00	10925.50	1122.55	8.03	10867.35	1132.86	7.43	1.08 x	0.17
bigkey	6381.37	5039.69	453.72	15.53	5025.27	450.96	13.19	1.18 x	0.35
s298	18843.30	14518.60	1340.72	6.28	13371.70	1297.13	0.12	52.33 x	7.10
tseng	7797.39	5969.87	583.967	8.76	5654.55	546.49	0.44	19.91 x	5.93
diffeq	9052.21	6850.52	726.60	12.22	6752.71	648.09	1.94	6.30 x	4.16
dsip	9805.97	7857.53	640.05	11.66	7237.09	615.09	0.15	77.73 x	7.37
s38417	14437.40	11132.90	1075.88	10.65	10647.97	927.28	0.22	48.41 x	7.56
s38584.1	13387.60	10671.80	886.90	10.99	10047.16	860.87	0.52	21.13 x	5.43
clma	22793.60	16828.10	1866.62	6.97	17534.50	1722.72	11.34	0.61 x	-1.17
Mean		**10103.99**	**979.52**	**8.27**	**9731.51**	**941.98**	**1.09**	**7.61 X**	**3.95**

Table 3.2: Results of our Variation Aware Router with cluster size 1

in T_{gb} as the timing specification. The column labeled Y_l *Imp* gives the improvement in yield loss that can be achieved with a statistical flow. On average, our router reduces the yield loss from about 8.27 chips out of 10,000 chips to about 1.08 which is a gain of 7.61x for the benchmarks considered. The column labeled ϕ *Imp* presents the improvement in circuit delay that can be obtained with our router if we maintain the same yield as the deterministic router. To compute this value, we first obtain the yield loss for the deterministic router. Then we use this value, along with the probability distribution function (PDF) of the variation aware router and find the delay from the distribution corresponding to the required yield loss. The difference between this delay and the timing specification gives the delay improvement that can be obtained with our router when we keep the same yield loss as the deterministic router. We get an average improvement of about 3.95% in circuit delay. This is because the router reduces both the mean and the standard deviation of the circuit's delay distribution.

Table 3.3 gives results of our using our router with a cluster size of 4. Columns 1 and 2 give the delay mean and standard deviation for all circuits when a deterministic router is used. Columns 4 and 5 give the corresponding numbers for the variation aware router. On average, the variation aware router reduces the delay mean by about 5% and the standard deviation by about 8%. The σ/μ ratios for cluster sizes 1 and 4 are about 9.7% and 8.8% respectively and this indicates that the overall impact of process variations is slightly lower for a higher cluster size.

Though we report a reduction of 7.61x in yield loss, this does not translate to a very big improvement in timing yield of circuits. This is because when we use the 3σ guard-banded delay as the timing specification, only about 8 chips out of 10,000 chips failed the timing requirement to begin with. However, this is a very pessimistic timing constraint. The yield loss also depends on the amount of variation present

Circuit	VPR		Variation Aware Router	
	μ(ps)	σ(ps)	μ(ps)	σ(ps)
misex3	5542.45	493.27	5183.49	436.37
ex5p	6968.28	634.11	6578.96	572.75
alu4	5273.78	471.27	4983.72	409.77
apex2	5914.94	530.56	5530.5	505.22
pdc	8606.42	800.63	7943.43	709.64
seq	5313.29	480.85	5047.63	413.76
des	7010.01	625.41	6799.71	588.62
spla	7322.89	650.53	6986.32	602.38
ex1010	9100.70	829.61	8555.73	805.52
frisc	8782.64	821.89	8308.38	719.17
elliptic	6674.84	625.85	6631.07	604.75
bigkey	4344.85	350.36	4216.85	330.97
s298	10105.30	922.23	9145.68	877.98
tseng	5335.70	489.86	5103.88	454.24
diffeq	5671.73	520.64	5443.3	480.36
dsip	3008.95	290.37	2981.93	275.81
apex4	5301.98	466.57	4967.95	427.77
s38417	7594.57	686.54	6921.11	596.87
s38584	4773.45	417.67	4519.03	412.54
clma	9742.33	913.54	9372.13	844.43
Mean	**6619.45**	**601.09**	**6261.04**	**553.45**

Table 3.3: Results of our Variation Aware Router with cluster size 4

and since we do not have access to real foundry data, the actual yield loss may be higher even at the conservative timing constraint if the variation is significantly larger than our assumption. To further verify the effectiveness of our router, we first performed speed-binning experiments where we wanted to observe the effect of variation-aware routing on the distribution of chips into various delay bins. Speed binning is typically performed by implementing test structures on an FPGA and then testing its operating frequency to classify it into fast, medium, slow and failure bins. There may be more bins but for the purposes of this work, we assume these four delay bins. This is illustrated in Figure 3.9. In this context, variation aware design techniques will not change the number of chips in various delay bins since fixed test structures are used, regardless of the designs to be mapped, to bin chips. However, in customer-specific FPGAs such as Xilinx EasyPath FPGAs [40] where only one specific design submitted by the customer is implemented on the FPGA, variation aware design techniques can indeed change the contents of different delay bins. It is in this context that we present our studies on speed binning. Using the delay

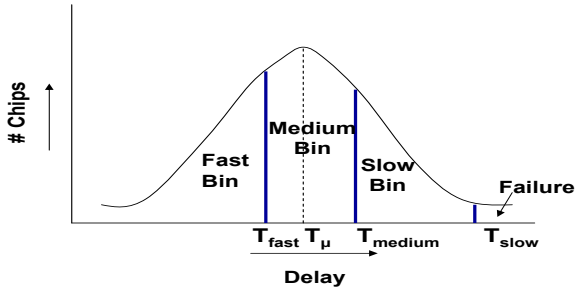


Figure 3.9: Speed Binning

distributions obtained for benchmark circuits for the deterministic router, we set the values of T_{fast}, T_{medium} and T_{slow} such that fast, medium and slow bins contain 40%, 30% and 29.90% of the chips respectively [45]. We then consider delay distributions obtained with our variation-aware router and bin chips into the same three delay bins.

The results of our speed-binning experiments are shown in Figure 3.10. The figure

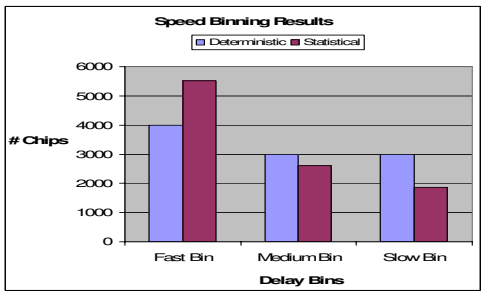
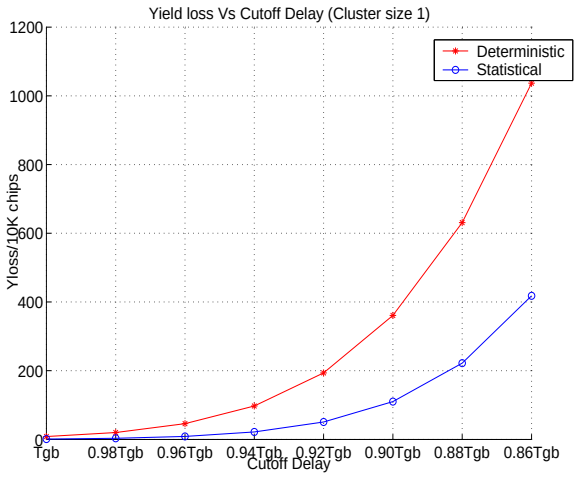


Figure 3.10: Speed Binning with Variation-Aware Router

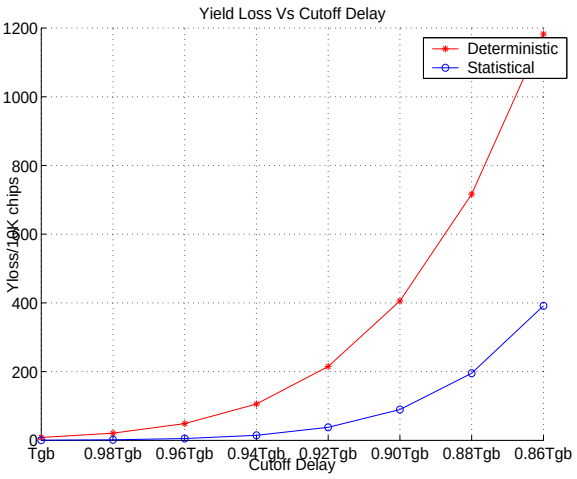
shows the average statistics of all benchmark circuits. It is clear that our variation-aware router significantly increases the number of chips in the fast bin. This increase is about 38%. This is beneficial since more number of chips can be sold at a higher price as fast chips. The number of chips in the medium and slow bins reduce but this is not a problem since some the chips that originally belonged to these bins are now shifted to the fast bins. Furthermore, the variation-aware router also helps to recover more chips from the failure bin since the yield loss reduces.

Since the yield loss numbers with the 3σ guard-banded delay (T_{gb}) as the timing specification was very low, we wanted to investigate the effectiveness of the router with more aggressive timing constraints. For this purpose, we varied the timing specifications from 86% of T_{gb} to T_{gb} with increments of 2% and computed the yield loss at every point. This information for cluster sizes 1 and 4 is shown in Figures 3.11(a) and 3.11(b) respectively. It is to be noted that for a cluster size of 4, the yield loss numbers are based on its guard-banded delay and hence the T_{gb} values are different for the two cases.

The Y-axis gives the yield loss per 10,000 chips and the timing specifications are shown in the X-axis. The yield loss numbers reported here are average numbers of all benchmark circuits. For both cluster sizes, the yield loss increases dramatically



(a) Cluster size : 1



(b) Cluster size : 4

Figure 3.11: Variation of Yield Loss with Timing Specification

as we tighten the timing constraints. With a cluster size of 1(4), the yield loss varies from 8.27(7.18) chips at T_{gb} to 1036.7(1182.6) chips at 86% of T_{gb} for the deterministic router. These numbers are 1.09(0.565) and 417.8(391.57) respectively for the variation aware router. This shows that although the conventional practice of guard banding circuit elements with the 3σ values results in low yield loss with conservative timing constraints, this number increases significantly when the aim is to design faster chips with more stringent timing requirements and variation-aware routing can lead to better timing yield. For instance, when the cutoff delay is 86% of T_{gb}, the timing yields using deterministic and variation aware routers are 89.64%(88.17%) and 95.82%(96.08%) respectively.

In this section, we investigated a CAD technique to mitigate the impact of variations by modifying the conventional deterministic router in VPR to optimize statistical criticality. In the third part of this chapter, we explore a combination of architecture and CAD techniques to further increase the timing yield of sequential circuits by performing clock skew assignment.

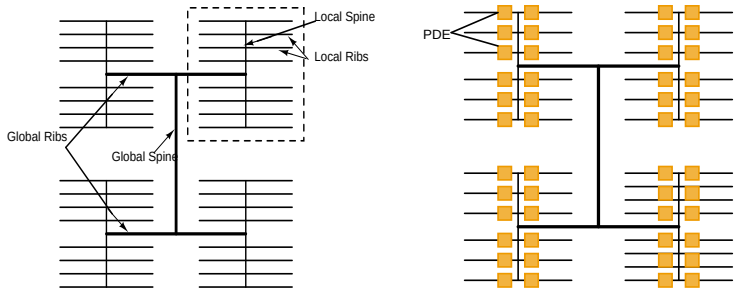
3.5 Skew Assignment for Improving Timing yield

As mentioned in Section 3.1, we can use the concept of time borrowing to improve timing yield of sequential circuits by distributing different clock skews to different flip-flops (FFs) to fix timing violations. In this section, we propose a skew distribution scheme with very little modification to the existing clock tree in FPGAs. This can be achieved by adding programmable delay elements (PDEs) to the clock network to deliver configurable skews to FFs. PDEs typically contain a number of delay blocks that have fixed delays. The delay of PDEs can be increased by making the clock signal pass through a desired number of delay blocks or alternately decreased by bypassing delay blocks. We perform extensive architecture exploration to identify

the number of delay blocks needed in each PDE to balance timing yield improvement with its associated area and power overhead. We also provide generic (performed before a design is implemented on FPGAs) and chip-specific techniques (performed after a design is implemented on FPGAs) to fix timing violations and conduct studies to observe how these techniques fare with both conservative and aggressive timing constraints. We further consider the impact of skew assignment on speed binning of chips. As in the previous section, we consider FPGAs with cluster sizes 1 and 4 in this work.

3.5.1 Architecture Design

FPGA clock networks are not typically designed with an H-tree topology since it is difficult to mesh it with a tile-based layout. They are designed with a spine-and-ribs topology where clocks are delivered to the spines which take it to various regions on the chip and distribute it to individual FFs using the rib structures. This is illustrated in Figure 3.12(a). We modify the clock architecture by adding PDEs on the local ribs.



(a) Spine and Rib topology (b) Tunable Clock Architecture

Figure 3.12: Proposed clock Network

Our proposed architecture is presented in Figure 3.12(b). We insert 4 PDEs in each row i.e. one for each local rib, giving us potentially 4 different skew values for every

row for a clock network with four ribs. It is to be noted that all the logic clusters that derive their clocks from a local rib would have the same skew so we should take care when we assign skews to avoid causing new timing violations. Though having only 4 distinct skew values per row does not give us very fine control of skews, it is still effective. This is because in practice paths that end up being critical in the design do not terminate on FFs that are within the same rib or quarter or a row. This is intuitive since paths that terminate on FFs within the same rib are short and would have lower delays than some of the other paths and hence will not be critical. We observed this in all of the studies we present in this section. We consider only one clock domain when presenting our approach for the ease of illustration. However, it can be easily extended to multiple clock domains.

Programmable Delay Elements

We borrow the programmable delay element that we use in this work from [78]. It is shown in Figure 3.13. The clock signal passes through a number of delay blocks (each containing a chain of 20 inverters) and the required skew is configured by storing appropriate values in the SRAM cells that control the output multiplexer. It is to be noted that available skews are discrete and depend on the delay of an individual delay block. We conducted SPICE simulations for a predictive 65nm technology and

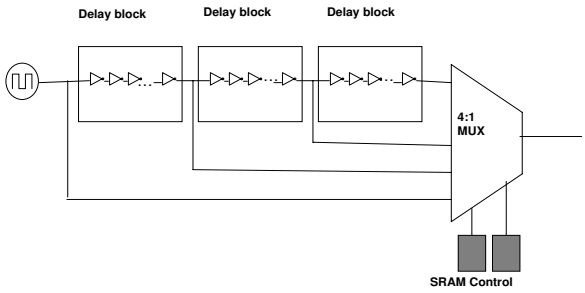


Figure 3.13: Programmable Delay Element

observed that the delay of a single delay block was around 150ps. To determine the optimum number of delay blocks needed in every PDE, we conducted experiments to study the tradeoffs involved in balancing yield improvement with the associated overhead. We present these studies in Section 3.5.1.

Architecture Exploration

The effectiveness of skew assignment techniques to fix timing violations depends on the amount of useful skew available. With the PDE design shown in Figure 3.13, this in turn depends on the number of delay blocks present in each PDE and the number of inverters used in each delay block. Our objective here is to maximize the yield improvement by providing enough delay blocks in the PDE while maintaining the area and power overhead to a reasonable level. In our study, we used the same delay block in [78] and tried to determine the optimum number of delay blocks needed in every PDE. To determine this number, we placed and routed sequential benchmarks and computed the 3σ guard-banded circuit delay (T_{grd}). We used T_{grd} as the timing specification to meet. We then performed Monte Carlo simulations with 10,000 samples on the circuits. For every chip that failed to meet the timing specification, we varied the number of delay blocks on every PDE from 0 to 6 to determine the number of chips that could be recovered with skew assignment. We plot the results in Figure 3.14. The y-axis shows the total number of chips in which the timing violations could be fixed with skew assignment for all sequential MCNC benchmarks we considered and the x-axis shows the number of delay blocks used in every PDE. From Figure 3.14 we observe that the number of chips recovered increases rapidly with the number of delay blocks initially and then the rate of increase reduces. The curve starts to flatten out when the number of delay blocks used is 3. This is because the short and long path constraints of circuits impose a limit on the actual critical path delay

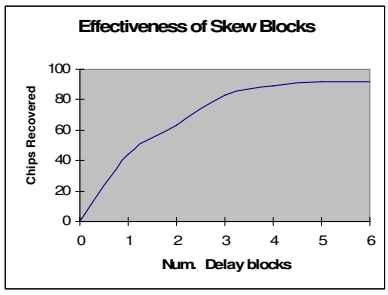


Figure 3.14: Architecture Exploration

reduction that can be obtained with skew assignment. Based on this observation and the need to reduce the area and power overhead, we decided to use 3 delay blocks in every PDE for this work.

In the next section, we state the skew assignment problem and present modifications to the problem to deal with layout constraints imposed by FPGAs and the discrete nature of available skews. We also present generic and chip-specific skew assignment techniques to fix timing violations.

3.5.2 Skew Assignment

We first present the unconstrained skew assignment problem and then augment it to consider layout constraints imposed by FPGAs and discrete skew constraints imposed by the PDE design. Consider Figure 3.15 from [60], which shows a part of a sequential circuit containing a combinational block and a pair of flip-flops (FF_i and FF_j) at the input and output respectively. The skews at these FFs are s_i and s_j respectively. The maximum and minimum combinational path delays are denoted by $d_{max}(i,j)$ and $d_{min}(i,j)$ respectively and T_{hold} and T_{setup} denote the hold and setup times of the FFs. If T_{clock} is the clock period of the circuit, we need the following conditions to be satisfied for correct circuit operation.

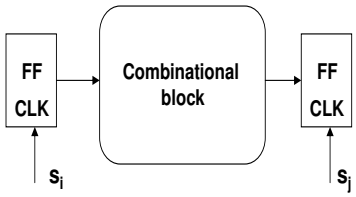


Figure 3.15: Setup for writing timing constraints

1. Long Path constraints : These impose a restriction that data launched from FF_i at a particular clock cycle should reach FF_j no later than T_{setup} before the next clock cycle. Mathematically,

$$s_i + d_{max}(i,j) \leq s_j + T_{clock} - T_{setup}$$

2. Short Path constraints : These impose a restriction that data launched from FF_i at a particular clock cycle should not reach FF_j earlier than the hold time for the previous data being clocked at FF_j. Mathematically,

$$s_i + d_{min}(i,j) \geq s_j + T_{hold}$$

These constraints are computed for each FF pair that are connected by combinational paths. Clock period, T_{clock} can be reduced by formulating the problem as a linear program with an objective of minimizing T_{clock}. Alternately, the problem can also be solved by performing a binary search on T_{clock}. A skew constraint graph is created and the Bellman-Ford algorithm is applied to it to test the feasibility of T_{clock} by checking for positive cycles at each step [60].

The formulation presented here assumes that we can individually control the skews of all FFs in the circuit and that available skews are unbounded. Neither of these assumptions are valid. This is because of layout constraints imposed by the architecture in Figure 3.12(b) and the discrete jumps in available skew due to the PDE design in

Figure 3.13. From Figure 3.12(b), it is clear that we have only 4 distinct skew values per row. To account for layout constraints, we first group FFs that are driven by the same PDE in every row of the chip. If we have an $n \times n$ FPGA and 4 PDEs per row, we will have a maximum of $4n$ FF groups. All FFs belonging to a group will have the same skew assigned to them and each group is given a unique identifier. It is to be noted that not all of the possible $4n$ groups will be used. The exact number of groups used depends on the design and where it is mapped on the chip. To generate long and short path constraints, we proceed by computing the maximum and minimum combinational delays between all FF pairs (i,j) that are connected by a combinational path. If i and j belong to different FF groups, we update the constraints. If i and j belong to the same group, the corresponding constraint is not included in the skew formulation. This is because i and j will have the same skew since they are driven by the same PDE. However, we still need to check whether paths that have their source and destination in the same FF group violate timing constraints. Skew assignment will not be able to fix these timing violations due to the layout restrictions in Figure 3.12(b). In our experiments we never encountered this problem.

Since we have only discrete levels of skew available, we introduce additional variables to account for the discrete skew problem. If there are L levels of skew available (S[1],$\cdots$,S[L]), we introduce L additional (0,1) variables x_1,x_2,$\cdots$,x_L for every skew variable i such that

$$\begin{aligned} x_{i1} + x_{i2} + \cdots + x_{iL} &= 1 \\ x_{i1}S[1] + x_{i2}S[2] + \cdots + x_{iL}S[L] &= s_i \end{aligned}$$

These additional constraints ensure that skews obtained by solving the mixed integer-linear program will satisfy the discrete skew constraints imposed by the PDEs.

The formulation presented here does not consider process variations. To fix timing violations, we need to consider both the variability in the combinational sub-circuits

as well as the clock distribution network. In Sections 3.5.2 and 3.5.2, we present two skew assignment schemes to fix timing violations considering process variations.

Generic Skew Assignment

We first present a technique to robustly assign skews considering process variations before the design is implemented on an FPGA. The skew assignment is performed after the place & route stage and is common to all chips (i.e. pre-fabrication optimization). We perform an SSTA as a first step and compute the statistical max, $d_{max}(i,j)$ and min, $d_{min}(i,j)$ for every FF pair (i,j). Let $(\mu_{max}, \sigma^2_{max})$ and $(\mu_{min}, \sigma^2_{min})$ be the (mean,variance) of the max and min delay distributions respectively. We introduce an *uncertainty factor* k to increase the variation tolerance in our skew formulation by adjusting d_{max} and d_{min} based on the mean and variance of the distribution. We set

$$\begin{aligned} \overline{d}_{max}(i,j) &= \mu_{max} + k \cdot \sigma_{max} \\ \overline{d}_{min}(i,j) &= \mu_{min} - k \cdot \sigma_{min} \end{aligned} \tag{3.6}$$

The uncertainty factor k tightens the long and short path constraints to make the skew assignment more robust to variations. In our work we chose the value of k to be 3. We use $\overline{d}_{max}$ and $\overline{d}_{min}$ instead of d_{max} and d_{min} in our formulation. To account for variations in the clock network, we introduce a *robustness factor*, R to the formulation and we try to maximize this parameter. Based on the technique in [72], we compute the maximum statistical criticalities of all combinational paths $p \rightsquigarrow q$, between all FF pairs (p,q) belonging to different FF clusters. We then use this criticality information $crit(p,q)$ along with the robustness factor and rewrite the skew assignment problem

as follows:

$$\begin{aligned} Maximize \quad & R \\ S.T. \quad \forall(i,j) & \\ s_i + \overline{d}_{min}(i,j) - crit(i,j)R &\geq s_j + T_{hold} \\ s_i + \overline{d}_{max}(i,j) + T_{setup} + crit(i,j)R &\leq s_j + T_{clock} \\ x_{i1} + x_{i2} + \cdots + x_{iL} &= 1 \\ x_{i1}S[1] + x_{i2}S[2] + \cdots + x_{iL}S[L] &= s_i \end{aligned} \tag{3.7}$$

Though we may assign specific skews to different FF clusters, the exact values may be slightly different due to variations in the clock distribution network. The R factor in the above formulation increases the tolerance of circuits to variations in skew. The component $crit(i,j)$ is used to provide more variation tolerance to those clusters that are connected by paths having a high statistical criticality since those are the paths that are affected most by variations in skew. We perform a binary search on T_{clock} to achieve the optimum clock period and at the same time maximize the robustness of the solution at each step by solving the linear program 3.7.

We evaluate the benefits of using this technique by first performing speed binning experiments. We then study its effectiveness in recovering failed chips with both conservative and aggressive timing constraints. For all studies in this section and in Section 3.5.2, we used deterministic place & route algorithms. This is because we wanted to study the effectiveness of the skew assignment technique to fix timing violations. In Section 3.6, we combine the variation-aware router with the skew assignment technique and study the interaction between the two techniques. We place and route sequential MCNC benchmark circuits and perform skew assignment based on the formulation in Equation 3.7. To study its impact on speed binning, we perform SSTA and set the values of T_{fast},T_{medium} and T_{slow} (from Figure 3.9) such that

the respective delay bins contain 40%, 30% and 29.9% of fabricated chips respectively. This is similar to the procedure described in Section 3.4.2. We then perform Monte Carlo simulations with 10,000 samples to gather statistics about the number of chips in each bin both with and without skew assignment. For all experiments we consider the variations in skew elements too. The results for a cluster size of 1 are shown in Figure 3.16. From the figure it is clear that skew assignment results in an increase

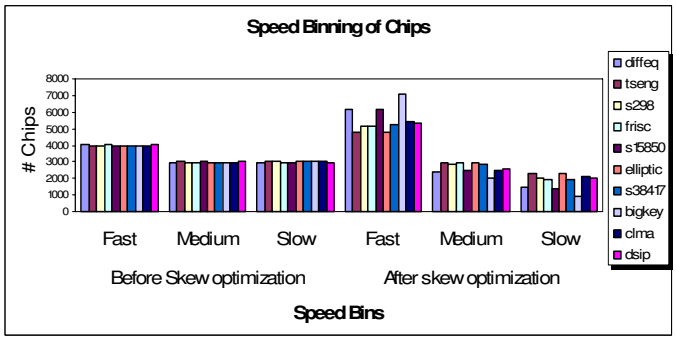


Figure 3.16: Speed Binning

in the number of chips in the fast bin. On average, the increase in the number of chips in the fast bin is 1.39x or 39%. Skew assignment provides a similar increase in the number of fast chips as the variation aware router (38% from Section 3.4.2). The number of chips in the medium and slow bins reduced to 0.88x and 0.61x respectively with skew assignment. Therefore performing skew assignment allows for more chips to be sold as fast chips. For the 10 benchmarks and 100,000 total chips, 561 chips belonged to the failure bin without skew assignment and this number reduced to 173 with skew assignment. This indicates that skew assignment is also beneficial in recovering more chips from the failure bin. We examine this in more detail by considering conservative and aggressive timing constraints.

We perform two sets of experiments to study the effectiveness of this technique in recovering chips from the failure bin. The first set of experiments were performed

with a conservative timing specification with the 3σ guard-banded circuit delay. For the second set of experiments we considered a more aggressive timing constraint and used the 2.5σ guard-banded delay as the timing specification to be met. In each case, we performed Monte Carlo simulations with 10,000 samples to determine the number of chips that could be recovered with skew assignment. These results are shown in Table 3.4. The number of chips that fail the timing specification and those

Circuit	GB Factor = 3 σ			GB Factor = 2.5 σ		
	$T_{spec}(ps)$	#Fail	#Rec	$T_{spec}(ps)$	#Fail	#Rec
diffeq	10878.3	11	9	10468.4	50	40
tseng	6824.36	11	10	6549.06	40	26
s298	21756.5	13	10	20952.9	32	15
frisc	13848.5	15	12	13326.6	42	29
s15850	9969.61	8	8	9554.62	38	25
elliptic	15712.5	17	10	15014.1	52	30
s38417	13039.6	12	11	12574	50	37
bigkey	6877.99	10	8	6638.99	40	38
clma	26442.1	10	6	25497.94	30	18
dsip	4901.54	12	10	4731.28	43	37
Geo. Mn.		**11.65**	**9.25**		**41.08**	**28.19**

Table 3.4: Generic Skew Assignment Scheme(cluster size 1)

that are recovered with skew assignment are shown in columns labeled #Fail and #Recover respectively. T_{spec} denotes the timing specification to be met. Columns 3 and 4 give the statistics for the conservative case with the 3σ guard-banded delay and columns 6 and 7 are for the aggressive case with 2.5σ guard-banded delay as the timing requirement. We observe that the skew assignment scheme is able to recover

about 80% of the failing chips with a conservative timing constraint and about 69% with an aggressive constraint. Results when using a cluster size of 4 is shown in Table 3.5. Here we get similar results with about 81% and 69% of failing chips being

Circuit	GB Factor= 3 σ			GB Factor = 2.5 σ		
	T_{spec} (ps)	#Fail	#Rec	T_{spec}(ps)	#Fail	#Rec
frisc	11248	14	11	10837	62	42
elliptic	8552.4	13	11	8239.5	54	35
bigkey	5395.9	11	9	5220.8	62	48
s298	12872	13	11	12411	41	28
tseng	6805.3	15	11	6560.3	57	35
diffeq	7233.7	12	11	6973.3	46	21
s38417	9654.2	13	10	9310.9	55	40
s15850	5994	14	10	5775.9	40	37
clma	12483	16	14	12026	53	34
dsip	3850.1	11	9	3734.9	59	49
Mean		**13.11**	**10.62**		**52.31**	**35.92**

Table 3.5: Generic Skew Assignment Scheme(cluster size 4)

recovered with conservative and aggressive timing constraints respectively. When timing constraints are tightened, the effectiveness of this scheme is slightly reduced. This can be attributed to the fact that layout constraints and the uncertainty in Equation 3.6 both have a bigger impact with tighter timing constraints. This effect however is marginal and the scheme performs well even with tight delay constraints.

Chip Specific Skew Assignment

The skew assignment scheme we presented in the previous section assigns the same set of skews to all chips. Since the technique is applied before implementing the design on a chip, there are uncertainties in delay estimation. In this section, we present a chip specific skew assignment scheme to fix timing violations.

The problem with a chip-specific scheme is that we need to know the exact delays of all nets and logic blocks to perform skew assignment. This is infeasible since we cannot accurately measure these delays on chip. Our approach to overcome this problem is to measure the delay of certain designated paths and use them to predict the delay of other circuit elements. This is very similar to the approach in [61]. We divide the chip into 8 grids and instantiate ring oscillators with a chain length of 11 in each of them. The exact number of grids and chain length of oscillators depends on the level of accuracy needed. The reason for dividing the chip into 8 grids is to extract spatial correlation information from the frequencies of the ring oscillators. Based on their frequencies, we predict the delay of circuit elements. For each grid i, we introduce a delay *prediction factor* p_i defined as

$$p_i = \frac{T_{act} - T_\mu}{T_\sigma}$$

where T_{act} is the delay of the oscillator in grid i, T_μ and T_σ are its delay mean and standard deviation obtained from SSTA. Once the p_i's for all grids are computed, we guard band circuit elements in each grid with its corresponding p_i value. We use a weighted average of p_i's for nets that span multiple grids. We did not expect this technique to be exact since the ring oscillators only approximately capture correlation information. To validate this technique, we conducted Monte Carlo simulations with 10,000 samples to compare the *actual* and predicted delays. To emulate the *actual* delay, we assumed we had access to all the individual net and logic delays in the

simulation and to emulate the predicted delay, we assumed we had access only to the ring oscillators' delays. Figure 3.17 shows the normalized *actual* and predicted delays for various circuits. On average, the delay prediction error was less than 1%. In the chip-specific scheme we perform skew assignment only for failing chips. We

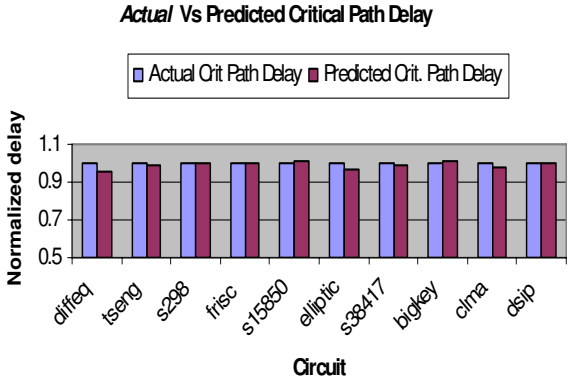


Figure 3.17: Delay Prediction Accuracy

first predict logic and net delays and use them in the skew formulation presented in Equation 3.7 to compute skews to be assigned to different clusters. Once these skews are obtained, the design can be re-implemented on the chip and tested to verify timing. Alternately, we can redo the physical design based on the estimated logic and routing delays. This however, increases the design time for every failing chip. So, we decided to use the same physical design for all chips and use only skew assignment to fix timing violations. Similar to Table 3.4 in Section 3.5.2 we test the effectiveness of this technique with conservative and aggressive timing constraints. The results are presented in Table 3.6. The cutoff delays and the naming convention for columns remain the same for Tables 3.4 and 3.6. We observe that the chip-specific skew assignment scheme performs marginally better than the generic scheme. About 82% of the failing chips are recovered with conservative constraints and about 77% with more aggressive constraints. The reason this performs slightly better is because the

Circuit	GB Factor = 3σ		GB Factor = 2.5σ	
	# Fail	# Recov	# Fail	# Recov
diffeq	11	7	52	32
tseng	11	10	59	58
s298	6	4	51	37
frisc	12	11	48	40
s15850	9	9	34	27
elliptic	9	6	34	27
s38417	8	8	40	35
bigkey	12	11	64	56
clma	5	3	31	16
dsip	8	8	47	40
Geo Mn.	**8.77**	**7.15**	**44.77**	**34.67**

Table 3.6: Chip-Specific Skew Assignment Scheme

delay prediction scheme does a good job of reducing uncertainty in delay estimation used in the problem formulation leading to a better skew assignment. This however comes at the cost of increased test time. The generic skew assignment scheme however is better for performing speed binning.

In this section, we have made modifications to the clock network and made use of statistical skew assignment techniques to compensate for variations. It would be interesting to study the contribution to the reduction in yield loss by the architecture and the statistical techniques in isolation. So we performed a purely deterministic skew optimization where we remove the robustness factor, uncertainty factor and statistical criticalities from the problem formulation and perform deterministic optimization to minimize the clock period. We perform Monte Carlo simulations with

this skew assignment to determine the percentage of chips that are recovered. This would give us the contribution of the PDEs in the clock network to reducing the yield loss without any statistical optimization. We present the results in Table 3.7. The

Circuit	GB Factor = 2.5σ	
	#Fail	#Recov
diffeq	56	41
tseng	48	15
s298	34	27
frisc	52	31
s15850	36	17
elliptic	51	36
s38417	54	40
bigkey	53	39
clma	35	14
dsip	40	30
Mean	45.12	27

Table 3.7: Deterministic Skew Assignment

timing specification that we used for the circuits in Table 3.7 is the same as in Table 3.4. We observe that about 59.8% of the chips that fail timing are recovered with a deterministic skew assignment scheme. This is the contribution of the architectural modifications. When we use the statistical skew assignment techniques presented in this section we can recover about 77%(refer to Table 3.6) of the chips that fail timing. The additional 17% reduction in yield loss comes from the statistical techniques. We believe that though the architectural modification gives the most benefit in reducing yield loss, statistical skew assignment techniques can be used to better leverage the

architecture to reduce the yield loss even further.

In the next section, we combine variation-aware routing with skew assignment to evaluate their effectiveness. We also study the interaction between these techniques and quantify the contribution of each technique to the overall timing yield improvement.

3.6 Interaction between variation-aware routing and skew assignment

In the previous sections we presented a variation-aware routing methodology and skew assignment techniques and studied their effectiveness in isolation. We conducted experiments to study the impact of each technique individually. In this section we combine the variation-aware routing and skew assignment techniques and study their interactions. We use the flow shown in Figure 3.18 for this purpose. There are four

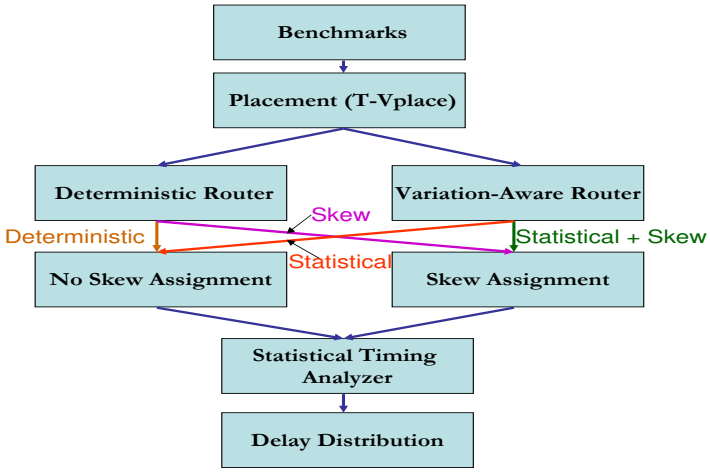


Figure 3.18: Interactions between different techniques

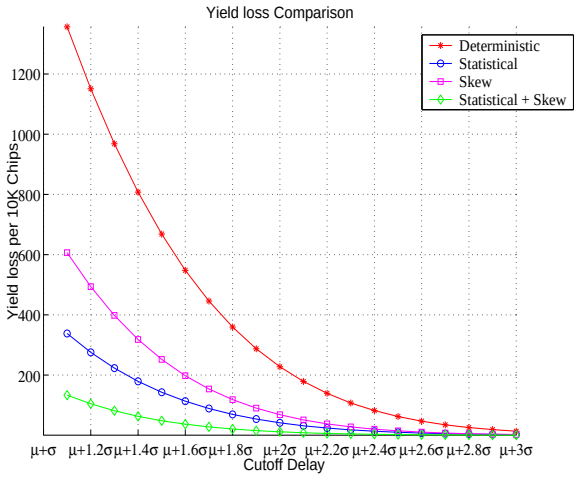
separate design flows that we consider. To capture the interactions accurately, we use the same placement and routing channel width for all 4 flows. The first is a purely

deterministic flow that makes use of the conventional router with all circuit elements guard banded by their 3σ values. In the second flow, we use a deterministic router but perform skew assignment. This is the same as in Section 3.5. For the third flow, we use our variation-aware router without any skew assignment. This is the same as the flow in Section 3.4. Finally, we use both variation-aware router and skew assignment to perform timing yield optimization. For each of these flows, we compute the output delay distribution by performing Monte Carlo simulation. We present the the results in Table 3.8. The table gives the average delay means and standard deviations of the sequential benchmarks for the four flows. The column labeled Det gives the results for the deterministic flow and like wise columns labeled Stat., Skew and Stat+Skew give the results for the variation aware routing only, statistical skew assignment only and the combined flows.

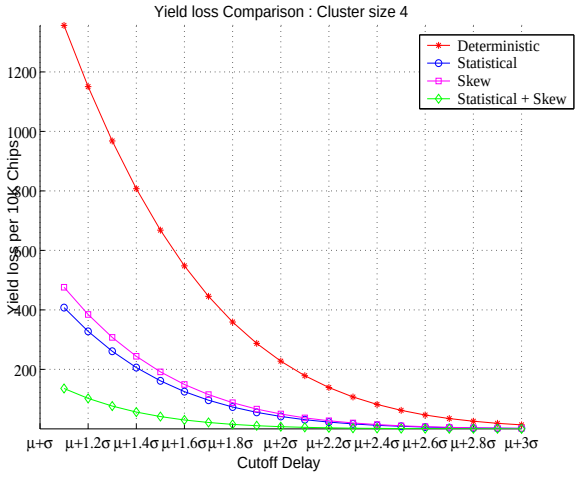
Cluster	Det.		Stat		Skew		Stat+Skew	
size	μ (ps)	σ(ps)	μ(ps)	σ(ps)	μ(ps)	σ(ps)	μ(ps)	σ(ps)
1	10048.17	965.01	8985.74	892.24	9705.01	944.65	8691.23	874.05
4	6594.67	605.73	6206.97	557.23	6308.47	573.29	6005.38	516.20

Table 3.8: Average delay distributions for sequential benchmarks

To evaluate the interaction between these techniques we perform two sets of experiments. First, for each of the design flows we compute the yield loss expressed as the number of chips that fail timing requirements out of 10,000 chips. We choose the fully deterministic flow to be our base line case. To observe how the other yield improvement techniques behave with various timing constraints, we vary the cutoff delay from a maximum cutoff delay of $\mu + 3\sigma$ of the deterministic flow to a minimum of $\mu + 1.1\sigma$. The results are shown in Figure 3.19. The X-axis shows the cutoff delays and the average yield loss per 10,000 chips of benchmark circuits is shown in the Y-axis. We observe that for both cluster sizes the curves follow sim-



(a) Cluster size 1



(b) Cluster size 4

Figure 3.19: Yield loss Comparison of the 4 flows

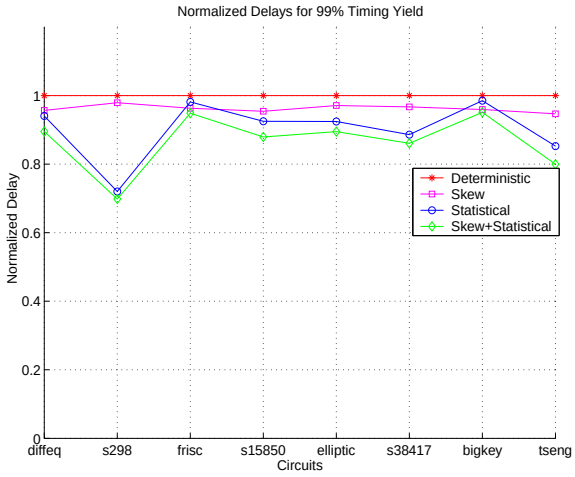
ilar trends. With conservative timing constraints, the yield loss for all 4 flows are very low. For instance, at the $\mu + 3\sigma$ point in Figure 3.19, the yield loss numbers for purely deterministic (D), variation-aware routing (Stat), skew assignment(Skew), and variation-aware routing + skew assignment (Stat+Skew)flows for a cluster size of 1(4)are 13.49(13.49), 2.79(1.84), 2.22(1.82) and 0.36(0.12). This shows that with conservative timing constraints both skew assignment and variation aware routing perform equally well. The yield loss numbers from combining the two techniques indicate that it has potential for reducing the yield loss further than the individual techniques applied in isolation. As we tighten the timing constraints, the yield loss numbers start to increase rapidly. This is expected since process variation plays a bigger role when delay variation is a larger fraction of the clock period. Variation aware routing performs slightly better than the skew assignment flow particularly at tighter timing constraints because of the limitation in the available useful skew in the PDEs. Combining the two flows however performs significantly better since the gain from skew assignment applies incrementally to the gain from variation-aware routing. When we take a look at our most aggressive timing constraint, $\mu + 1.1\sigma$, the average yield loss for D, Skew, Stat. and Stat+Skew flows are 1356.7(1356.7), 637.3(518.67), 411.5(471.35) and 147.76(146.67) chips respectively with a cluster size of 1(4). Combining variation aware routing with statistical skew assignment provides about a 9.2x improvement in yield loss. The yield loss numbers are reported for 10,000 chips. So, the timing yield numbers D, Skew, Stat. and Stat+Skew flows with a cluster size of 1(4) are 86.43%(86.43%), 93.63%(94.81%), 95.88%(95.28%) and 98.52%(98.53%) respectively. We observe that combining the two variation compensation techniques results in about a 12% improvement in the timing yield of circuits.

The discussion in the previous paragraph presents the potential benefit of the techniques proposed in this work when aggressive timing constraints are used. In

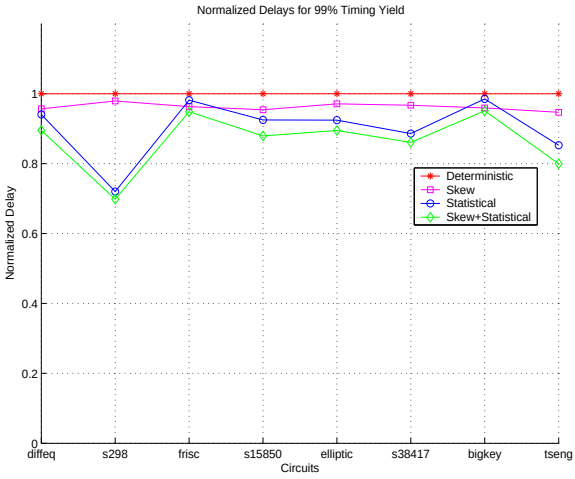
practice however, the timing yield is fixed to be a certain number and the speed of chips are decided based on the desired timing yield. This is done so that we do not end up throwing away chips because they do not meet aggressive timing constraints. We fixed the desired timing yield to be 99% or the yield loss to be 100 chips out of 10,000 chips for all flows to determine the speed of chips. We present these results in Figure 3.20. We plot the delays of all flows normalized to the delay of the purely deterministic flow. On average, the delay improvements in using Skew, Statistical and Skew+Statistical flows with a cluster size of 1(4)are 4.15%(4.86%), 8.41%(6.09%) and 12.25%(10.23%) respectively. Though the improvement provided by just the Statistical flow is higher than the Skew only flow, we note that for some of the circuits, the Skew-only flow performs better than the Statistical flow. For all circuits, the Skew+Statistical flow performs the best and further reinforces our belief that the skew assignment technique can be used to augment the gain from variation-aware routing.

3.7 Overhead

In this section we discuss the overhead associated with each of the techniques proposed in this work. Variation-aware routing presented in Section 3.4 involves changing the routing algorithm to optimize statistical criticality instead of deterministic criticality. The computational complexity of variation aware routing is asymptotically the same as the deterministic router and is $O(E+V)$, where E is the number of nets and V is the number of logic blocks. However, the constants involved with SSTA are higher. If we divide the chip into k grids, there will be k principal components for every process parameter and if we consider m process parameters there will be a total of mk coefficients in Equation 3.3. Hence the total time complexity of a single SSTA is $O(mk(E+V))$. We observed that on average a single routing iteration of our variation



(a) Cluster size 1



(b) Cluster size 4

Figure 3.20: Normalized delays for the 4 flows

aware router was about 1.39x slower than the deterministic router. However, this does not give the entire picture. The total time taken for routing is also dependent on the number of routing iterations used for both the routers. The number of routing iterations depends on the cost function used in the routing algorithm. We observed that the geometric mean of the total time for routing all benchmarks with our router was only about 1.02x slower than the deterministic router. This indicates that the variation aware router needed a lower number of routing iterations to converge to a solution. Using statistical criticality as a parameter in the cost function helps in reducing the number of iterations needed by the router.

For the skew assignment schemes proposed, we analyze computational complexity issues and the area and power overhead involved in adding PDEs to the clock tree. The major runtime overhead in the skew assignment scheme is in solving the mixed integer-linear program shown in formulation 3.7. Integer programs are notoriously hard to solve because its worst case complexity is exponential in the number of integer decision variables in the program. However, we found that the run-times for the benchmark circuits that we used were acceptable. The largest benchmark that we used was S38417[2] which had 6541 LUTs and 1463 FFs. The number of FFs is more critical in the formulation since the skews of these FFs form the decision variables. We implemented our code in C++ and used a 3.06GHz machine with 2GB of main memory and the run time for solving the MILP for S38417 was 24.72 seconds. In comparison, the time taken for routing the design was 49.96 seconds. This shows that the run times are reasonable. However, as the number of flip-flops and the number of available distinct skew values increase, the problem could get intractable. To avoid this, we can relax the MILP problem and remove the discrete skew constraints from the formulation and solve the relaxed linear program. We can then map the solution

[2]clma had 8383 LUTs but only 33 FFs

obtained to the set of available skews. Another parameter that can blow up the computational complexity is the criticality component used in the formulation. The component $crit(i, j)$ used in formulation 3.7 gives the maximum statistical criticality of paths between FFs. If we were to do it on a path-by-path basis this will have exponential complexity. To counter this, we make the same assumption as in [72] and calculate the criticality of a path as the product of arrival tightness probabilities of the edges on the path. We first compute the arrival tightness probabilities of all edges in the timing graph. Once we have the tightness probabilities of the edges, we then create a new graph which is topologically the same as the timing graph but with edge weights as the logarithms of the arrival tightness probabilities. Now the problem of finding the maximum criticality between a pair of FFs is transformed to a problem of finding the longest path in the new graph. This is exactly the same problem as timing analysis and can be computed efficiently.

In addition to the computational complexity issues, the proposed skew assignment schemes also have a hardware overhead. Similar to VPR, we estimate the area overhead by counting the number of additional minimum width transistors needed. The PDE shown in Figure 3.13 consists of 150 transistors. If the FPGA chip contains 10,000 4-LUTs arranged in a 100×100 array, the area overhead in modifying the clock network is about 3.5%. We modified the clock architecture in *powermodel* [56] to compute the power overhead. Adding PDEs to the clock network increased the average power dissipation in the clock network by about 5.6%. It is to be noted that the overall increase in power is lower and depends on the fraction contributed by clock power to the total power. We can reduce this further by using energy efficient delay elements that work by increasing the switching resistance rather than the capacitance [73].

3.8 Summary

In the first part of this chapter, we studied the difference in impact of process variations on designs mapped to ASICs and FPGAs. We observed that for our assumption of process parameters, STA surprisingly did a much better job of identifying critical paths in FPGAs than ASICs in the presence of variations. This showed us that for conservative timing constraints, deterministically optimizing the top few critical paths for FPGAs would have a similar effect as statistical optimization techniques. However, this may not be the case in future technologies or when timing constraints are tighter. So the rest of the chapter focussed on variation compensation techniques.

We proposed a variation-aware router in the second part of the chapter that optimizes statistical criticality instead of deterministic criticality. This approach resulted in a 7.6x improvement in yield loss. We then proposed a modification to the clock network to deliver programmable skews to different FFs and presented two robust skew assignment schemes to compensate for variations. We obtained an yield loss improvement of about 80% and 69% for conservative and aggressive timing constraints with a generic skew assignment scheme. The corresponding numbers for a chip-specific scheme was 82% and 77% respectively. Finally, we combined the two techniques and studied the contribution of each to improving yield loss and their interaction. When both techniques are used, we observed an improvement of 9x in yield loss which corresponds to a 12% improvement in the timing yield. When the desired timing yield is 99%, using both our techniques resulted in an improvement of about 10% over the purely deterministic flow.

Future work would involve combining the CAD techniques proposed in [44] with the CAD and architecture techniques presented in this work to create a unified flow for the FPGA community to address issues related to process variations.

Chapter 4

Reliability Estimation and Optimization Techniques for Digital Circuits Subject to Transient Errors

4.1 Introduction

The continued scaling of silicon-based systems in the deep nanometer regime presents numerous technological challenges. Issues such as thermal fluctuations, quantum effects and radiation strikes manifest themselves as transient errors. These are beginning to affect the functionality and reliability of devices [31]; [32]; [51]. The impact of transient errors on combinational circuits is projected to be as severe as that on memory elements in the near future [62]. To make matters worse, the failure rates of emerging technologies such as quantum dots and molecular devices are expected to be significantly higher than those of CMOS devices. We will soon be at a point where circuit reliability will be a dominant parameter in the design of circuits. To address this concern, we need fast and accurate techniques for estimating reliability. We need to incorporate these techniques into efficient strategies for optimizing circuits for reliability.

Previous methods that have considered reliability estimation with transient errors are computationally intensive [11, 31, 41]. Some of the work in the area has had a technology-dependent focus, relying on the electrical characteristics of circuit

elements [51, 81]. This kind of technology dependence has limited use in logic-level synthesis. Our contributions in this work are twofold. First, we develop an efficient reliability estimation technique using a new signal probability propagation method. Second, we develop an ATPG based rewiring framework and gate sizing to improve the robustness of circuits. We have presented our reliability estimation technique and our rewiring framework in [66]. We further analyze the impact of these techniques on circuit delay, area and power consumption to determine the most effective approach to improve circuit reliability. We achieve a 10% improvement in circuit reliability with a 15% improvement in area and an 18% improvement in the power-delay product with our rewiring framework. Our combined rewiring and gate sizing based optimization results in a 17% improvement in reliability with an area and power-delay overhead of 2.5% and 7% respectively.

4.2 Fault Modeling

Our aim is to develop logic-level algorithms to estimate and optimize circuit reliability. Accordingly, we adopt the technology-independent fault model used in [11,31,32,41]. Transient faults are modeled as bit-flips at the outputs of gates and are assumed to last for exactly 1 clock cycle. The probability with which a gate's output is flipped is its failure probability.

We make the following assumptions in this work.

1. Only bit-flip faults occur in the circuit.

2. All gate failures occur independently of each other.

3. Only the effects of logical masking are considered. Logical masking occurs when failure at a gate is suppressed by a controlling value at the inputs of one of its

dominators. It is technology independent and depends only on the logic being implemented.

To model a gate G that has a failure probability of ξ, we connect a dummy XOR gate at the output of G. The second input of the dummy XOR gate is connected to a signal, e_i that has a signal probability of ξ. This ensures that the output of G is flipped with a probability of exactly ξ. G and the dummy XOR gate together model the faulty gate.

To estimate circuit reliability, first we transform the original circuit by adding the dummy XOR gates mentioned above and then we compare the primary outputs of the transformed faulty circuit to those of the original fault-free circuit. The setup that we use for reliability computation is shown in Figure 4.1. In this figure, the

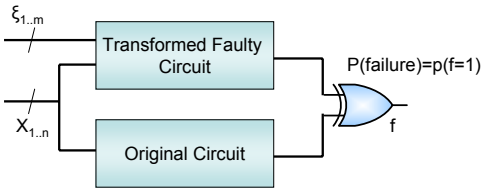


Figure 4.1: Setup to Estimate Circuit Reliability

X_i's form the inputs to the circuit. Each gate g_i has a dummy input ξ_i with a signal probability equal to its failure rate. After transforming the circuit, we compute signal probabilities of the outputs of the fault-free and faulty versions of the circuit. The signal probability of the final dummy XOR gate, f in Figure 4.1, gives the failure rate and consequently the reliability of the circuit. For multi-output circuits, we have a dummy XOR gate for each output and perform the disjunction of the XOR outputs to compute the overall circuit failure rate. In the next section we present a technique for computing circuit reliability using this setup.

4.3 Estimating Circuit Reliability

As explained in the previous section, we convert the problem of estimating a circuit's reliability into the problem of calculating signal probabilities of all of its internal nodes. Many techniques have been proposed to compute signal probabilities in combinational circuits [53].

At one end of the spectrum, there are simulation-based methods which apply large numbers of input vectors. At the other end of the spectrum, there are probabilistic methods that entail propagating probabilities based on simple rules. Probabilistic methods are, in general, more efficient than simulation-based methods. Unfortunately, they suffer from low accuracy due to signal correlations. Several studies have addressed this issue [27, 35, 39]. Although promising, these methods are not accurate enough to be used practically [48]. In [55], the authors propose an exact technique to evaluate signal probability. However, for large circuits their exact method is intractable. [25] partially applies the exact technique to circuits by considering signal reconvergences up to a specified logic depth; beyond this depth, all signals are assumed to be uncorrelated. This results in inaccuracies. In [21], the authors use the approach proposed in [27] to consider signal correlations and develop a scalable technique to estimate circuit reliability. However, the approach in [27] to tackle signal correlations results in inaccuracies. In Section 4.4, we illustrate this issue and modify the approach to improve the accuracy.

Our ultimate objective is to estimate reliability of nodes in a circuit and use that information in an optimization framework to improve overall circuit reliability. The optimization procedure may involve several iterations where reliability is evaluated to achieve good results. In this context, it is crucial that we develop fast and accurate techniques to estimate reliability. To this end, we have developed a hybrid approach that combines the exact approach with probabilistic techniques.

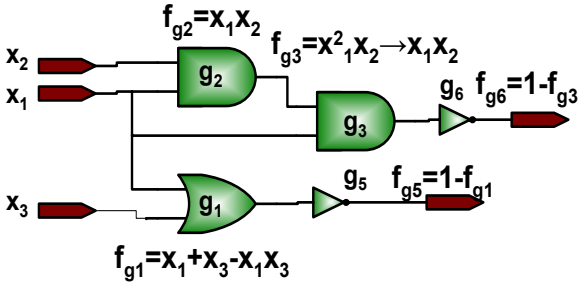


Figure 4.2: Illustration of probability propagation

4.3.1 Preliminaries

To better understand our hybrid approach, in this section we review background information about exact and probabilistic techniques.

Exact Approach

The exact method proposed in [55] evaluates signal probability of a node as a polynomial function of the signal probabilities of its inputs. Consider a logic circuit with n inputs, each associated with signal probability variables, $x_1, x_2, \cdots x_n$. Let G be an AND gate with 2 inputs. If $f_1(x_1, \cdots, x_n)$ and $f_2(x_1, \cdots, x_n)$ represent the functions of the signal probabilities of its inputs, the output signal probability is given by the function f_g, such that

$$f_g(x_1, \cdots, x_n) = f_1(x_1, \cdots, x_n) \times f_2(x_1, \cdots, x_n)$$

The corresponding expression for an OR gate is $f_g = f_1 + f_2 - f_1 f_2$ and for an inverter is $f_g = 1 - f_1$. Signal dependencies appear as exponents in these polynomials and are suppressed to achieve accurate results. This is illustrated in Figure 4.2.

In this figure, x_1, x_2 and x_3 denote signal probability variables corresponding to the inputs. Note that for gate g_3, application of probability propagation rules yields the function $x_1^2.x_2$. This indicates that g_3 acts as a site of reconvergence for x_1. The

correct function at g_3 is obtained by converting f_{g3} to x_1x_2 by suppressing the extra exponent of x_1. In [55], these functions are represented using BDDs and finally signal probabilities of all nodes are obtained by performing simulations on these polynomials by assigning Boolean values to $x_1 \cdots, x_n$ according to the input signal probabilities and evaluating the mean values of the polynomials.

In our framework for reliability estimation, shown in Figure 4.1, in addition to the primary inputs, we also have dummy inputs($\xi_i's$) modeling the failure rates of gates. These can be easily embedded into probability propagation rules presented in [55]. For example, let g_i be a two-input AND gate with signal probabilities at its inputs given by f_1 and f_2. The signal probability of its output is given by $f_1f_2(1-\xi_i)+(1-f_1f_2)\xi_i$, where ξ_i is the failure probability of the gate.

The ξ_i variables are real numbers and are typically fixed for gates. As a result, unlike [55], we require a convenient way to represent a mapping from the Boolean domain to the real domain ($\Re_+$). So, we can adopt the approach in [41] and use algebraic decision diagrams (ADDs) to represent the polynomial function at each node. Using this method, the evaluation of signal probabilities is exact. Unfortunately, the method is computationally infeasible for large circuits, especially if it is applied in the inner loop of an optimization framework.

Probabilistic Approach

To mitigate the cost of the analysis, probabilistic techniques have been widely used to estimate signal probabilities. In this work, we apply the idea proposed in [27] to account for signal dependencies. The authors introduced the notion of explicit correlation coefficients (interchangeably referred to as correlations in this work) between signals. The correlation coefficient, $C_{i,j}$ between two signals i and j is defined as

$$C_{i,j} = \frac{P(ij)}{P(i)P(j)} = \frac{P(i|j)}{P(i)} = \frac{P(j|i)}{P(j)} \tag{4.1}$$

Instead of propagating polynomial functions as is done in the exact method, probabilistic techniques work by propagating real numbers corresponding to the signal probabilities of nodes. Probability propagation rules are augmented to include correlation coefficients. For instance, signal probability at the output of a 2-input AND gate with input signal probabilities given by p_1 and p_2 is given by $p_1 p_2 C_{1,2}$, where $C_{1,2}$ is the correlation coefficient between the two inputs. For an OR gate, the output signal probability becomes $p_1 + p_2 - p_1 p_2 C_{1,2}$.

To propagate correlation coefficients, the authors assume that the circuit's primary inputs are independent of each other. A topological sort algorithm is first used to levelize the circuit. Correlations are computed for each pair of edges that cross a level (i.e. an edge cut-set). This information is used to compute probabilities of nodes in the next level according to the augmented probability propagation rules. This process continues until it reaches the output nodes. Signal correlations between a pair of edges is computed as follows. Let f denote the function implemented by a gate; i and j are its inputs and y is its output. The correlation between y and another edge m can be computed with the help of the following equations [27].

$$\begin{aligned} p(y) &= f(i,j) \qquad (4.2) \\ p(y|m) = p(y)C_{y,m} &= f(i|m, j|m) \\ C_{y,m} &= \frac{f(i|m, j|m)}{f(i,j)} \end{aligned}$$

Since we process edge cut-sets topologically, $f(i|m, j|m)$ is known when we compute $C_{y,m}$ in Equation 4.3.

In [21], the authors use this technique to account for signal correlations in their reliability estimation framework. This method may yield exact probabilities in certain cases; however, in general, it is not accurate since it considers only first-order correlations. Higher-order correlations such as two signals depending simultaneously on a third signal are ignored. This is illustrated in Figure 4.3. The circuit has 2 re-

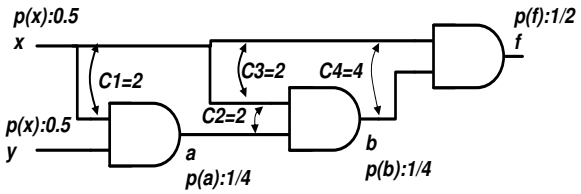


Figure 4.3: Inaccuracies with Correlations

dundant gates and only performs the AND operation of two variables but is useful to illustrate the underlying issues in modeling correlations[1]. The correlations between pairs of signals are annotated as the C_i values in the figure. They are computed using the technique presented in [27] and reviewed in this section. Assuming x and y are independent with a probability of 1/2 each and the gates to be fault-free, the probability propagation method described in this section computes the probability of gate f as 1/2. This however, is inaccurate since $f = xy$ and the exact probability of $f = 1/4$. The problem lies in computing the correlation between inputs of node f.

From the definition of correlation in Equation 4.1, C_4 should be computed as

$$C_4 = \frac{p(b|x)}{p(b)} = \frac{p(xy|x)}{p(xy)} = \frac{p(y)}{p(xy)} = 2 \tag{4.3}$$

In the approach presented in [27] however, $C_4(=C_3.C_2)$ is computed from C_3 and C_2 by considering only the pairwise dependence of signals. This results in an error in the value of C_4 leading to an inaccurate result for the probability of gate f. Intuitively this happens because this method of considering correlations does not remember how the signal was correlated after it passes a level but only propagates the correlation numbers to cuts in the next levels.

In the next section we present a hybrid approach where we combine the features of the exact and probabilistic techniques to develop a fast and accurate reliability

[1]Given that synthesis tools are not perfect, similar redundancies might happen in practical applications

estimation algorithm.

4.4 Hybrid Approach to Reliability Computation

In this section we develop a hybrid approach for reliability computation that provides the scalability of the probabilistic technique with the higher accuracy of the exact approach. It is clear that if we can partition the circuit into a set of independent regions, we can use the exact approach for smaller regions which may have signal reconvergences inside them and then combine the results using probabilistic techniques.

There are two potential challenges. The first is to identify a set of independent regions in the circuit that are small enough for the exact approach to be practical. The second is to develop an interface between the exact and probabilistic techniques to combine the results.

4.4.1 Identifying Independent Regions

Previous work such as [49] discussed techniques for partitioning a circuit into a set of independent regions called super-gates. These independent partitions would allow the exact technique to be applied for smaller regions of the circuit. However, from our studies as well as that presented in [25], we note that in real circuits, the sizes of these super-gates are still large. This adversely impacts the applicability of the exact approach. Also, the technique in [49] depends on finding articulation points of the circuit graph. However, after we apply the circuit transformation in Figure 4.1, the circuit does not have any articulation points.

Ideally, all sources of reconvergence at a node should be considered exactly in order to accurately evaluate the signal probability of the node. However, this is computationally infeasible. So, we relax this condition and use an approximation where

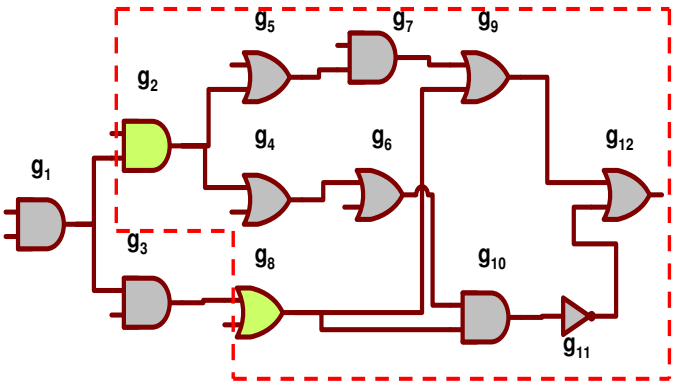


Figure 4.4: Illustration of Super-gates

we consider only the *sources of first reconvergence* for every site of reconvergence. The rationale behind this assumption is the observation that the effect of signal dependencies on a node decreases as we move away from the node in its transitive fanin-cone. The following definitions are useful in describing the notion of a *source of first reconvergence* for a node.

1. Source of Reconvergence : A node u is a source of reconvergence for v if there are at least two pairwise edge disjoint $u \rightsquigarrow v$ paths in the circuit graph

2. Sources of *first* Reconvergence : If S is the set of all sources of reconvergence for u, then the sources of first reconvergence for u is given by the set $S_{first,u}$ defined as

$$S_{first,u} = S - \{v_i | \exists v_i \rightsquigarrow v_j; v_i, v_j \in S; i \neq j\}$$

In our approach, we redefine the notion of a super-gate to represent a maximal pseudo-gate rooted at a site of reconvergence. It has the sources of first reconvergence as its inputs. Note that, unlike the traditional super-gate, the inputs to our super-gate may not be independent of each other. The definitions are illustrated in Figure 4.4.

Applying Definition 1 to gate g_{12} in the figure, $\{g_1, g_2, g_8\}$ is the set of all sources

of reconvergence at g_{12}. However, a path exists from g_1 to g_2 and g_8, so g_1 is removed from this list to form the set $S_{first,12}$. g_{12} is called the super-gate root. For each super-gate rooted at a reconvergent node(g_{12} in the figure), we include only the set of sources of first reconvergence as its input to reduce its size. We treat all gates on the set of pairwise edge disjoint paths from the sources of first reconvergence to the reconvergent node as a part of the super-gate rooted at the reconvergent node. The set $\{g_2, g_4, \cdots, g_{12}\}$ forms the super-gate rooted at g_{12} in Figure 4.4 (enclosed in a dashed line). g_2 and g_8, the sources of first reconvergence for g_{12} supply its inputs.

Every gate that has more than one fanout is a potential source of reconvergence for a downstream gate. We compute super-gates in two phases. We first perform depth-first graph traversals from candidate sources and collect gates where signals recombine. In the second phase, we identify the super-gates by collecting gates in a reverse topological order starting from sites of signal reconvergence identified in the first phase.

4.4.2 Combining Symbolic and Probabilistic Techniques

We apply the probabilistic approach for all non-reconvergent nodes and the exact approach for the super-gates rooted at reconvergent nodes. The treatment of non-reconvergent nodes is the same as in Section 3-A.2.The exact technique cannot be directly applied to super-gates for evaluating the signal probabilities of reconvergent nodes. The rest of the section describes how to adapt the exact and probabilistic techniques for this.

Representing Signal Probability of a Super-gate as an ADD

To capture the effects of signal correlation from the sources of first reconvergence for a super-gate, we assign a Boolean variable to each source of reconvergence. This is

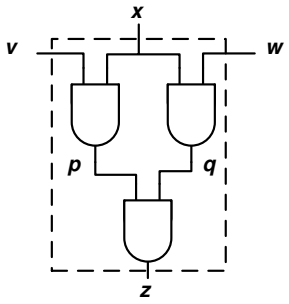


Figure 4.5: Pseudo-Inputs to a Super-Gate

similar to the treatment of primary input nodes in the exact approach. The difference in the hybrid approach is that we may have gates other than sources of first reconvergence that supply inputs to some gates inside the super-gate. We treat these inputs as pseudo-inputs. Instead of assigning separate Boolean variables for these inputs, we treat them as real numbers representing probability values. This minimizes the size of the decision diagram that is needed to represent the polynomial function representing the signal probability at the root of the super-gate. This is illustrated in Figure 4.5.

In the figure, z is a reconvergent node and x is a source of first reconvergence for z. The super-gate rooted at z consists of x, p, q and z. v and w are pseudo-inputs to the super-gate rooted at z.

Since we treat the pseudo-inputs as real numbers, we may lose some correlation information if they are correlated to some of the sources of first re-convergence at an upstream node(For instance, v and x may be correlated at an upstream node in Figure 4.5). To capture this effect, we embed the correlation between pseudo-inputs and signals present in the super-gate while building the decision diagram of the super-gate. Correlation information is stored as a part of the function being evaluated at the super-gate root. This function is stored as an ADD.

We use the example in Figure 4.6 to illustrate the process of building the ADD of

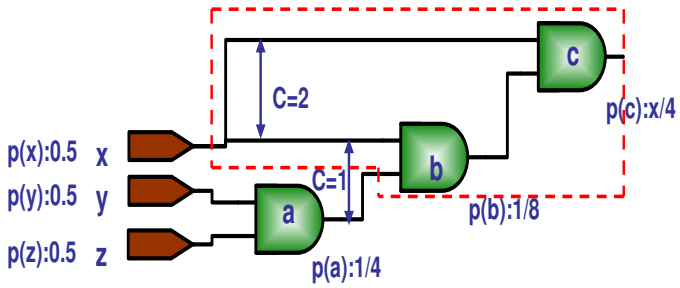


Figure 4.6: Evaluating Signal Probability using a Hybrid Approach

a super-gate, embedding correlation information and finding probabilities of reconvergent nodes. Nodes x, b and c constitute the super-gate rooted at c. Node x is a source of first reconvergence and is given a separate Boolean variable in the decision diagram. Here a is a pseudo-input and is treated as a real number. $C_{x,a}$ (=1 in the example since x and a are independent) is the correlation between the pseudo-input and a source of first reconvergence. Assuming the gates are fault-free, the probability of c is given by the function,

$$p(c) = xp(a)C_{x,a} = x.\frac{1}{4}.1 = \frac{x}{4}$$

We store this function ($\frac{x}{4}$) as an ADD at node C.

The entire procedure involved in representing the signal probability of a super-gate root as an ADD is described in Algorithm 2. Only inverters and 2-input AND gates are considered in the algorithm for ease of illustration. Other types of gates are handled similarly. We note that separate symbolic boolean variables are assigned only to the inputs of a super-gate. This is handled in Line 4 of Algorithm 2. Pseudo-inputs(eg. v, w in Figure 4.5) are handled in Line 18 of the algorithm.

Algorithm 2 Hybrid approach to represent signal probability of super-gate root as an ADD

1: **procedure** [ADD(G_1)]=SUPER-GATEROOTADD(G_1,S_{first,G_1})
▷ G_1 = Super-gate root, (Eg. Gate g_{12} in Figure 4.4)
▷ S_{first,G_1} = Super-gate rooted at G_1, (Eg. Gates enclosed in a dashed line in Figure 4.4)
2: *Queue* $Q \leftarrow \{\}$
3: **for** $g_i \in$ $Input(S_{first,G_1})$ **do** ▷ Eg. Gates g_2 and g_8 in Figure 4.4
4: $ADD(g_i) \leftarrow CreateADDVariable()$ ▷ Separate symbolic variables for super-gate inputs
5: **for** $g_j \in S_{first,G_1} \cap Fanouts(g_i) \&\& g_j \notin Q$ **do**
6: $Q \leftarrow Q \cup \{g_j\}$
7: **end for**
8: **end for**
9: **while** $Q \neq \{\}$ **do**
10: $g_j \leftarrow Q.pop()$
11: $V \leftarrow Fanins(g_j)$ ▷ Collect fanins of the gate being processed
12: **if** $g_j.type ==$ **INVERTER then**
13: $ADD(g_j) \leftarrow 1 - ADD(V[0])$
14: **else if** $g_j.type ==$ **AND then**
15: **if** $V[0], V[1] \in S_{first,G_1}$ **then**
16: $ADD(g_j) \leftarrow ADD(V[0]).ADD(V[1])$ ▷ Both fanins are part of the super-gate
17: **else**
18: $ADD(g_j) \leftarrow ADD(V[0]).P_{v1}.C_{v_0,v_1}$ ▷ V[1] is a pseudo-input (Eg. v,w in Fig 4.5)
▷ P_{v1} is signal probability of V[1]
▷ C_{v_0,v_1} is correlation between V[0] and V[1]
19: **end if**
20: **end if**
▷ Other types of gates are treated similarly
21: $FO \leftarrow Fanouts(g_j)$
22: **for** $g_i \in FO \cap S_{first,G_1}$ **do**
23: Q.push(g_i)
24: **end for**
25: **end while**
return ADD(G_1)
26: **end procedure**

Evaluating signal probability(ADD) of super-gates

To compute the signal probability of the root node of a super-gate after computing the ADD, we generate Boolean vectors for the sources of first reconvergence based on their probabilities (these are already known since we process gates in a topological order) and evaluate the ADD of the root node (computed by Algorithm 2). The mean of the decision diagram values at the root node gives its probability. For instance, in Figure 4.6, to evaluate the function represented as an ADD at node C, we apply Boolean values 0 and 1 to x with a probability of 0.5 each since $p(x) = 0.5$ and evaluate the decision diagram at $c(p(c))$. This gives the correct probability of c as 1/8.

If we revisit the example in Figure 4.3, nodes x,b, and f constitute the super-gate rooted at f. Node x is a source of first reconvergence and a is a pseudo-input. Assuming the gates are fault-free, the probability of f is given as

$$p(f) = xp(a)C_{x,a} = x.\frac{1}{4}.2 = \frac{x}{2}$$

We apply boolean values 0 and 1 to x with a probability of 0.5 each since $p(x)$=0.5 and evaluate $p(f)$. This gives the probability of f as 1/4 and this is the correct value. The probabilistic method presented in [27] and used in [21] would inaccurately compute this value as 1/2.

Evaluating the probability of the root node of a super-gate by assigning boolean values to sources of first reconvergence according to their probabilities will lead to inaccuracies if the sources of first reconvergence themselves are correlated (gates g_2 and g_8 for instance in Figure 4.4). We have developed a heuristic to correct this problem.

Suppose there are k sources of first reconvergence for a super-gate. Let $n_1 \cdots n_k$ be the k unique symbolic Boolean variables we assign them. We order these nodes

according to their levels in the circuit graph. When we evaluate the probability of the super-gate root, we start from n_1 and assign it either 0 or 1 based on its probability. If we have assigned values to m nodes, the probability for the $(m+1)^{st}$ node is adjusted as the conditional probability based on the previous m assignments as follows.

$$\begin{aligned} p_{adjusted}(m+1) &= p(n_{m+1}|(n_1 = b_1, \cdots, n_m = b_m) \\ &\quad , b_i \in \{0,1\}) \end{aligned} \tag{4.4}$$

If e denotes the event that $n_1 = b_1$,$n_2 = b_2$,$\cdots$ $n_m = b_m$, (4.4) reduces to

$$p_{adjusted}(m+1) = p(n_{m+1}).C_{n_{m+1},e} \tag{4.5}$$

We consider only first order correlations between sources of first reconvergence. So, we approximate the correlation term in (4.5) as

$$C_{n_{m+1},e} \approx C_{n_{m+1},n_1=b_1} \times \cdots \times C_{n_{m+1},n_m=b_m}$$

$C_{n_{m+1},n_i=1}$ is computed the same way as in Section 3-A.2.$C_{n_{m+1},n_i=0}$ is given by

$$C_{n_{m+1},n_i=0} = p(n_{m+1}).\frac{\left(1 - p(n_i)C_{n_{m+1},n_i=1}\right)}{(1 - p(n_i))} \tag{4.6}$$

We generate Boolean vectors for the sources of first reconvergence based on these adjusted probabilities and then evaluate the probability of the super-gate root by using the method described earlier in this section. The overall circuit reliability is obtained by applying the probabilistic approach for all non-reconvergent nodes and by applying the technique described in this section for reconvergent nodes in the circuit.

In the next section we describe our experimental setup to validate our work and compare our approach with the exact and probabilistic techniques described in this section.

Table 4.1: Results of Reliability Estimation

Circuit	Reliability				Runtime (sec)				Error(%)		
	Exact	Prob	Hyb	Sim	Exact	Prob	Hyb	Sim	Prob	Hyb	Sim
fulladder	0.7451	0.7642	0.7332	0.7482	0.1	0.1	0.1	18.98	2.56	1.59	0.42
C17	0.8315	0.8577	0.8519	0.8251	0.1	0.1	0.2	17.54	3.15	2.44	0.76
b1	0.8057	0.8068	0.7988	0.8015	0.1	0.1	0.1	18.61	0.14	0.99	0.66
cm42a	0.9085	0.8584	0.8913	0.9094	0.1	0.2	0.3	48.22	5.51	1.89	0.09
cm138a	0.9114	0.9078	0.9078	0.9152	0.1	0.4	0.2	52.23	0.39	0.39	0.41
tcon	0.8237	0.8561	0.7985	0.8312	0.1	0.5	0.5	56.61	3.92	3.06	0.90
count	0.6840	0.7801	0.7152	0.6903	0.4	0.77	1.49	317.33	14.05	4.56	0.93
c8	0.7601	0.7626	0.6969	0.7527	0.2	0.83	1.68	361.98	0.33	8.31	0.96
sqrt	0.5287	0.5881	0.5542	0.5295	0.2	0.48	1.86	343.79	11.21	4.82	0.14
term1	0.7392	0.7547	0.7429	0.7342	8.06	5.38	10.82	949.02	2.09	0.50	0.68
alu4	0.4436	0.4786	0.4498	0.4494	16.14	14.06	44.76	1589.01	7.88	1.38	1.29
C432	x	0.3543	0.3835	0.4136	x	4.11	8.23	745.43	14.33	7.27	0
too_large	x	0.1861	0.1709	0.1588	x	22.76	45.16	1994.92	17.16	7.59	0
Mean					**2.33**	**3.83**	**8.87**	**501.05**	**6.37**	**3.45**	

4.4.3 Validating the Hybrid Approach

To validate the accuracy and efficiency of our hybrid approach, we implemented reliability estimation techniques based on the exact, probabilistic and hybrid signal probability evaluation approaches and compared them to a Monte Carlo-based fault injection framework. For each input vector sample in the Monte Carlo simulation, we can perform fault simulations and inject faults at gates based on their failure probabilities. We assume that the primary inputs are all independent[2] with a probability of 0.5 and failure rate of gates, ξ is 0.005. In our symbolic and hybrid reliability computation techniques, we sample the input space of the function being evaluated until the final probability converges.

We implemented all techniques presented in this work in C++ and used benchmark circuits from the LGSynth93 benchmark suite. The results are presented in

[2]We can easily extend our work to include the case where inputs are correlated

Table 4.1. Exact, Prob, Hyb and Sim refer to exact, probabilistic, hybrid and Monte Carlo approaches to reliability estimation, respectively. From the results, it is clear that the exact approach works well for small circuits but is infeasible for larger circuits due to the exponential increase in the size of the decision diagrams involved. So, the exact approach cannot be used in an optimization framework.

We use the reliability estimated using the exact approach as the baseline to compare the accuracy of probabilistic and hybrid approaches. For circuits in which the exact approach fails, reliability obtained from Monte Carlo based fault injection is used as the baseline. The errors of the probabilistic and hybrid approaches are about 6.4% and 3.5% on average, respectively, when compared with results from Monte Carlo-based fault injection. The maximum error of the hybrid approach is 8.31% as opposed to 17.16% for the probabilistic approach indicating that the hybrid approach is more consistent overall. The errors of both techniques increase with the size of circuits. For the 4 largest circuits in Table 4.1, the average errors are 10.4% for the probabilistic approach and 4.2% for the hybrid approach. An increase in the estimation error for large circuits is intuitive because of a greater likelihood of signal reconvergences in the circuit. This may lead to more approximations in the computation. Both techniques still perform reasonably well.

From Table 4.1, it can be seen that on average, the exact technique performs very well for small circuits since the decision diagrams involved are smaller and evaluating them is easier than propagating correlations across edge cut-sets. Over all circuits used, the probabilistic approach was the most efficient. This is expected since our hybrid approach involves all the computation used in the probabilistic approach in addition to identifying and evaluating super-gates. When compared to the Monte Carlo approach, the probabilistic and hybrid approaches offer average speedups of $131X$ and $56X$, respectively.

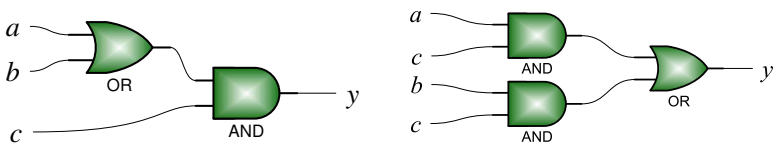
In the next section, we explore two techniques to optimize circuit reliability. We first present a rewiring-based optimization framework followed by a selective gate sizing technique to improve robustness of circuits to transient errors. Both techniques make use of our hybrid reliability estimation method to efficiently evaluate different circuit configurations.

4.5 Reliability Optimization Techniques

Most approaches ([8,41]) that optimize circuit reliability involve selective gate sizing to harden gates that are sensitive to transient errors. In this section, we discuss two approaches to optimize reliability. The first approach involves restructuring logic by performing incremental rewiring to improve reliability. The second approach involves selective gate sizing to improve circuit resilience to transient errors. We study the effectiveness of both approaches in improving reliability and their associated area, power and delay overhead.

4.5.1 Rewiring based Reliability Optimization

We observe that circuit topology has a significant impact on reliability. This is demonstrated in Figure 4.7, which shows two circuit realizations of the same logic function. The failure probability of an individual gate in each case is 0.05. The difference in their reliabilities is about 5%.



(a) Circuit A, Reliability : 0.927 (b) Circuit B, Reliability : 0.884

Figure 4.7: Reliability for different circuit topologies

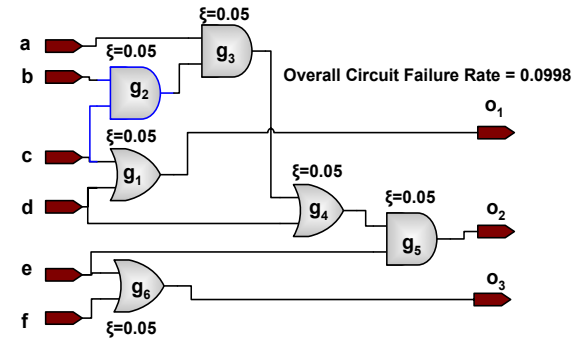
Rewiring is an effective technique to alter circuit topology. It uses a series of transformations that involve removing some wires in the circuit and adding new wires, while still maintaining the same functionality. Intuitively, this is readily applicable to reliability optimization. We can improve circuit reliability by replacing wires having low reliability with more reliable wires.

The potential of rewiring-based approaches to optimize reliability is illustrated in Figure 4.8. Wire $c \Rightarrow g_2$ is replaced by $g_1 \Rightarrow g_5$ after rewiring. This makes g_2 redundant; it is removed from the circuit. We observe that the failure rate of the circuit falls from 0.0998 to 0.0611 after the rewiring transformation. There are two main reasons for this. Firstly, rewiring reduces the number of gates through which c passes through before reaching O_2. This reduces the chances of errors on the path. Secondly, rewiring reduces the number of gates in the circuit. This automatically reduces the failure rate. Additionally, the area that is saved by rewiring can be used to further reduce the failure rate by selective gate up sizing.

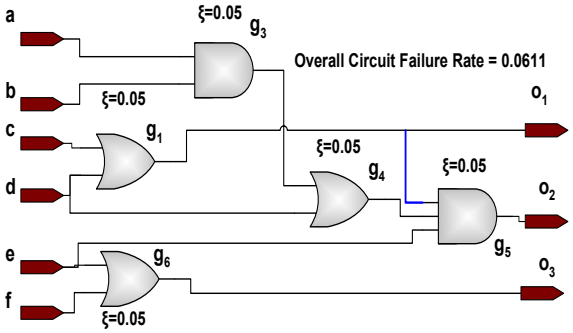
Rewiring Framework

We adopt an ATPG-based rewiring approach for incrementally restructuring a circuit to improve its reliability. The rewiring engine first identifies a set of mandatory assignments to nodes to propagate a signal on the wire to be replaced to the outputs. Then, a set of candidate wires which, when added to the circuit, would make the target wire redundant are identified. Finally, the target wire is replaced by one of the candidate wires. The ability to rewire signals and the choice of available candidate connections depend on the number of mandatory assignments that can be identified.

Our rewiring engine is similar to [17], which uses direct implication to uncover mandatory assignments. Direct implication is a well established technique in which mandatory logic assignments to signals are determined by simple forward and back-



(a) Original Circuit



(b) Circuit after Rewiring

Figure 4.8: Reliability for different circuit topologies

ward traversals of the circuit graph. To uncover more mandatory assignments we provide a parameterized backtracking procedure where we recursively perform dynamic implication to a few logic levels as specified by the user. Implicants uncovered this way are verified by casting the circuit as an instance of a SAT problem and checking the implicants uncovered by backtracking for consistency. We also use the blocking clauses generated by the SAT solver to uncover more implicants. This is again controlled by a user parameter in order to trade off runtime with the number of implicants uncovered.

We can optimize reliability by first identifying nodes in the circuit that have low reliability. We can then rewire signals feeding these nodes with alternate wires driven by more reliable sources in the circuit. If we select the nodes and thereby the wires to replace intelligently, the overall circuit reliability can be improved. We have developed cost metrics to help us identify target wires to be replaced as well as their alternate wires. We explain these metrics in Section 4.5.1.

Cost metrics for rewiring

Our approach is based on identifying nodes with low reliability. We modify the reliability estimation setup shown in Figure 4.1 to insert dummy XOR gates for every node in the original circuit to obtain the reliability of each node in addition to the overall reliability. We also use the concept of *observability* of a node, which is a measure of the likelihood that the logic value at the node is visible at the output. We use the approach in [27] to evaluate observabilities of all nodes in the circuit as probabilistic measures.

We use the reliabilities and observabilities of internal nodes in the circuit to select a target node(sink of the wire we want to remove) according to the following cost

measure:

$$C_t(N) = (1 - \alpha_t - \beta_t).(p_{fail}(N)) + \alpha_t.obser(N) + \beta_t.\frac{l_N}{L_{max}} \tag{4.7}$$

where C_t is the cost of the target node, $p_{fail}(N)$ is its failure probability, $obser(N)$ is its observability, l_N is its level in the circuit and L_{max} is the maximum depth of the circuit. α_t and β_t are parameters that control the relative importance of these three factors and are tuned experimentally. The terms containing observability and level of the node give importance to nodes that are highly visible and close to the output. This is important to ensure that significant effort is not wasted on nodes that may have poor reliabilities but whose errors are unlikely to propagate to the outputs. We select the target node based on this cost measure and try to rewire the inputs to this node in decreasing order of their failure rates.

We select the source of the candidate alternative wire to be added based on the following cost measure.

$$C_s(N) = (1 - \beta_s).(p_{rel}(N)) + \beta_s(1 - \frac{l_n}{L_{max}}) \tag{4.8}$$

Where $p_{rel}(N)$ is the reliability of gate N, i.e. $1 - p_{fail}(N)$. The other terms have a similar meaning to the target node described in Equation 4.7. We prefer source nodes candidate wires that have high reliability and are close to the inputs. This will reduce the chances of an error occurring on the source of the candidate wire. After selecting a candidate source for the alternate wire, we select a sink by giving preference to nodes that are closer to the output. Adding a new wire to a gate closer to the output reduces the number of gates that the signal passes through. This limits the chances of accumulating errors along the path. It has the added benefit of reducing the delay on the path by reducing its logic depth.

After selecting the target and candidate wires, we replace the target wire by the candidate wire and update the circuit. For performance reasons, we re-evaluate circuit

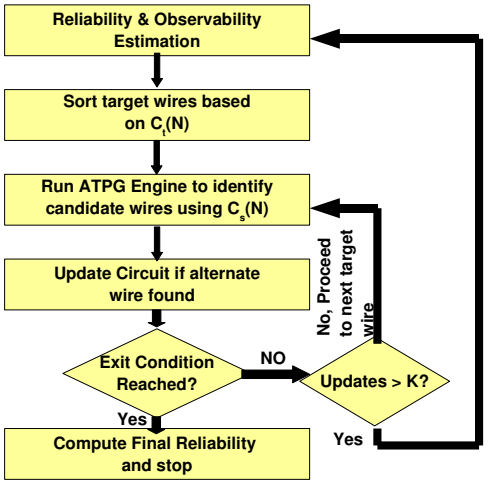


Figure 4.9: Rewiring flow

reliabilities and observabilities after every k updates. We repeat this process until either there are no more rewiring options or until the overall reliability stops improving for a few iterations. Our rewiring flow is shown in Figure 4.9. Our experimental framework and results are discussed in Section 4.6.

4.5.2 Reliability Optimization by Gate Sizing

Gate sizing is a widely adopted technique([8,82]) to improve the resilience of circuits to transient errors. The key idea is that larger gates require more energy from the source of transient errors to cause bit-flips. This effectively reduces failure rates of larger gates leading to an increased overall circuit reliability.

Typically, a few gates are selected to be up-sized according to the impact they have on circuit reliability. Gate observability is a common selection criterion. The rationale behind using this metric is the observation that reducing error rates of highly visible gates would lead to an improvement in circuit reliability. Our intention in this section is to determine how a traditional observability based optimization

approach compares to our rewiring scheme and to study the interactions between the two approaches. Similar to the work in [8, 82], we increase the size of the selected gates to 4x their original size to make them 100% resilient to transient errors.

While gate sizing can increase the reliability of circuits, it may adversely affect other metrics such as delay and power if they are not considered in the optimization process. [8] addresses this by performing sizing on only those gates that have a positive timing slack. This will result in a circuit whose delay after sizing would be no worse than the initial delay.

In our work, we give equal importance to observability and timing cost associated with each gate that we consider for resizing. The criterion that we use for selecting a gate to be up-sized is given by

$$Cost(G_i) = Observability(G_i).\frac{\Delta Delay_{G_i}}{\Delta Area_{G_i}} \tag{4.9}$$

Where, Observability(G_i) is computed using the technique presented in [27]. $\Delta Delay_{G_i}$ gives the change in circuit delay when gate G_i is up-sized to 4x its original value and $\Delta Area_{G_i}$ is the increase in circuit area. The second term in Equation 4.9 gives the delay sensitivity of the gate and tries to optimize circuit delay. For each gate G_i, $Cost(G_i)$ has equal contribution from both the observability cost that is geared towards optimizing reliability and delay sensitivity that targets circuit delay.

In our work, we first compute the observability of all gates in the circuit. We then perform timing analysis to obtain the initial circuit delay. We compute delay sensitivities by performing incremental timing analysis on each gate. All gates (G_i's) in the circuit are ordered in descending order of Cost(G_i). When the cost of two gates are equal, we give preference to the gate having a higher observability. In the optimization phase, we select the gates from this list and up-size them under a given area constraint.

4.6 Reliability Optimization Results

In this section we describe our experimental framework and analyze the effectiveness of rewiring- and sizing-based reliability optimization approaches. We also analyze the impact of the techniques on metrics such as area, delay and power of circuits. In all our experiments, we use a simple 100nm standard cell library containing two and three input NAND, NOR gates and inverters to evaluate circuit delay, area and power.

We assume the failure rate of all gates to be 0.005 before sizing and 0 after they are sized to 4x their original sizes. Circuit area is reported as the number of minimum width transistors used in the circuit. Delay is obtained by performing timing analysis on the optimized circuit. We use SPICE simulations and fit a quadratic gate delay model for all gates in the library for different capacitive loads. We perform extensive circuit simulations to estimate switching activities of each gate and use this data to compute the total dynamic power of the circuit.

Figure 4.10 illustrates the flow we used in our study. We use three different optimization flows - rewiring based reliability optimization, gate sizing based optimization and a combination of rewiring and gate sizing based optimization. A Monte Carlo simulation based reliability estimation approach is used to evaluate the pre- and post-optimization reliabilities. Our hybrid reliability estimation approach is used in the rewiring framework shown in Figure 4.10.

Figure 4.11 shows the reliability of circuits for the various optimization flows that we studied. The X-axis shows different circuits from the LGSynth93 benchmark suite and the Y-axis gives circuit reliability. The graph labeled *Orig* gives the reliability of the original circuit without any optimization and is used as the baseline for comparison. The bar labeled *Rewire* gives the reliability when only rewiring optimization is performed. *Sizing* gives the reliability when only gate sizing is performed and

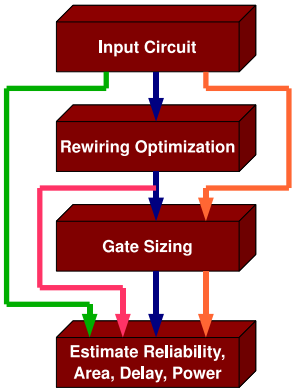


Figure 4.10: Optimization Flows

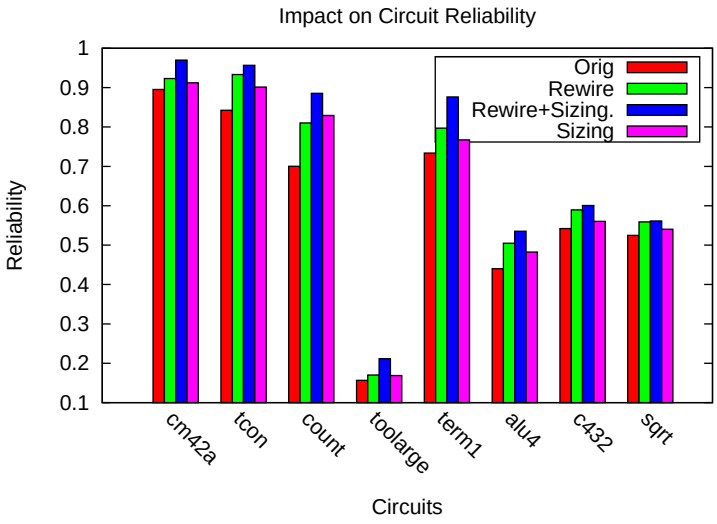
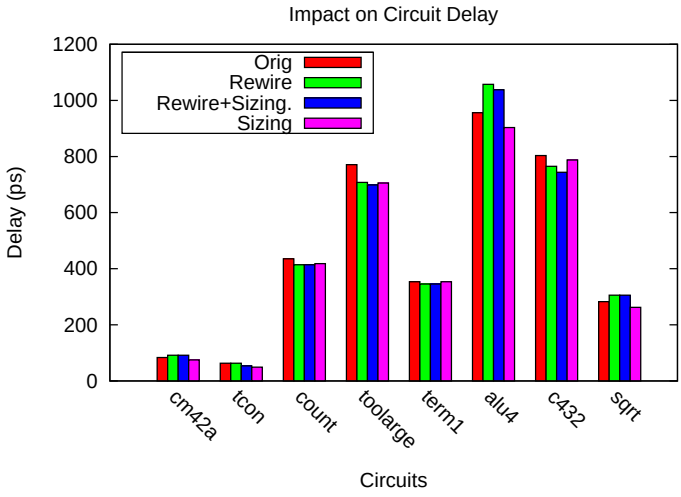


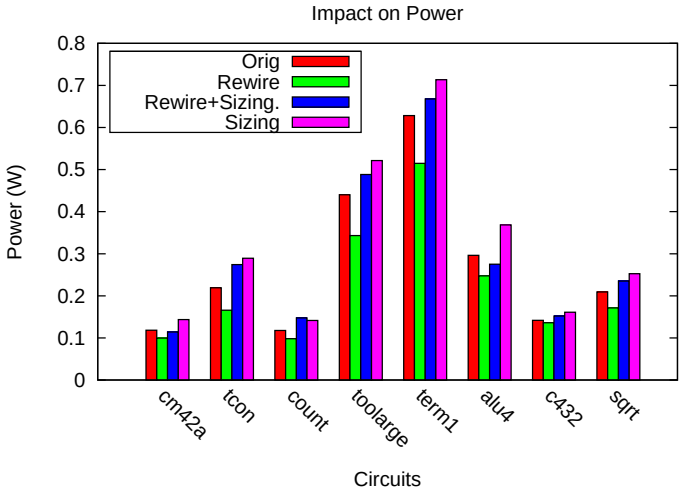
Figure 4.11: Circuit Reliability

Rewire+Sizing gives the reliability when a combination of rewiring and gate sizing is performed. On average, we achieve a 10% improvement in reliability when rewiring optimization is performed. When only gate sizing is performed, the reliability improves by about 8%. This supports our belief that circuit topology plays a vital role in determining circuit reliability and rewiring is a good tool to restructure logic circuits to improve reliability. We note here that we limited the maximum area overhead to 10% of the un-optimized circuit area for the gate sizing approach. As expected, the best results are obtained with a combination of the two techniques(*Rewire+Sizing* in the figure) giving a 17% improvement on average in reliability. The maximum improvement in reliability achieved by *Rewire,Sizing* and *Rewire+Sizing* were 16%, 26% and 18% respectively.

Figure 4.12(a) shows the impact of our reliability optimization flows on circuit delay. We observed that on average, the rewiring approach increases circuit delay by about 0.75%. However, for 5 benchmarks rewiring based reliability optimization improved the circuit delay by an average of about 4% and for the remaining benchmarks, the delay worsened by about 9%. While it is expected that the delay would worsen if we optimize for reliability, we can explain the reason behind delay improvement for most of the benchmarks by taking a closer look at our rewiring cost metric in Equation 4.8. The candidate source for an alternate wire to be added is chosen to be close to the primary inputs (lower $\frac{l_n}{L_{max}}$). Likewise, the candidate sink for the alternate wire is chosen to be close to the primary outputs. This leads to a reduction in the overall depth of the path leading to reduced delays. For the *Rewire+Sizing* optimization flow, circuit delay improves by 2.3% on average with a maximum improvement of about 15%. This is expected since we use delay sensitivities of gates in addition to their observabilities when performing gate sizing. The *Sizing* optimization flow improves delay by about 8.4% on average and the delay improves for all the



(a) Circuit Delay



(b) Total Power

Figure 4.12: Impact of different optimization approaches

benchmarks we considered. This is expected since most circuits have several gates having positive delay sensitivities. Since we do not change the topology of the initial circuit, there is more scope to improve circuit delay by gate sizing.

Impact of the reliability optimization approaches on power is shown in Figure 4.12(b). We observe that the *Rewire* optimization flow reduces power consumption by about 18% on average. Rewiring removes several gates in the circuit leading to a reduction in the total capacitance switching every cycle. This reduces power consumption significantly. *Sizing* and *Rewire+Sizing* flows increase power consumption by 21% and 9.5% respectively. This is expected since up-sizing gates increases switching capacitance leading to more power consumption. Since the delay for each flow is different, we use the power-delay product (PDP) as the metric to compare the overhead to ensure fair comparison. The PDP of the *Rewire* optimization flow is about 18% better than the original circuit. However, *Rewire+Sizing* and *Sizing* increase the PDP and the overhead was about 7% and 11% respectively.

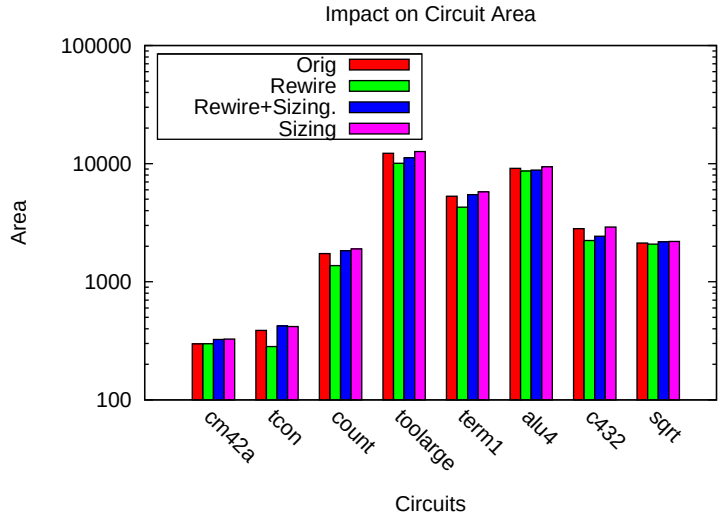


Figure 4.13: Circuit Area

Figure 4.13 shows the effect of different optimization flows on circuit area. Area is reported as the number of minimum width transistors in the design. The *Rewire* optimization flow reduces circuit area by about 14% on average. Area improvement is achieved by removing gates from the design during the rewiring process. In the *Rewire+Sizing* flow, area is first saved during the rewiring phase of the flow. In the gate sizing phase, we allow a maximum area overhead of 10% of the original circuit area. We further restrict sizing to gates that have non-negative delay sensitivities to avoid an adverse impact on timing. The overall area overhead for the *Rewire+Sizing* flow was about 2.5%. The *Sizing* flow incurs an area penalty of 7%, which is the largest among all the approaches.

In summary, we observe that a rewiring based reliability optimization approach performs marginally better(10% VS 8%) than a gate sizing technique in improving circuit reliability. Furthermore, a rewiring based approach has significant benefits in reducing the area(15%) and power consumption(18%) of circuits. The gate sizing approach on the other hand significantly improves the timing(8.4%) of circuits while incurring a PDP and area overhead of 11% and 7% respectively. A combination of rewiring and gate sizing based approaches provides the best results in terms of improving circuit reliability (17%) with an acceptable PDP and area overhead of 7% and 2.5% respectively.

Chapter 5

Fast Thermal Simulation of Single-Processor and Chip-Multi Processor Systems

5.1 Introduction

As technology scales deep into the nanometer regime, power density is increasing exponentially. It is becoming increasingly harder for modern packages and cooling systems to keep up with the thermal demands imposed by current designs. The problems are only getting worse with the integration of many processor cores on a single die. Elevated chip temperatures and temperature variations across the die significantly increase cooling costs. Leakage power is exponentially dependent on temperature and rising temperatures create a positive feedback mechanism that could create thermal runaway. High temperatures also create performance issues as circuit delay increases with temperature potentially causing functional failures. Large temperature gradients (spatial and temporal) create reliability problems due to effects such as electro-migration [69], negative bias temperature instability and hot carrier injection [42].

Dynamic Thermal Management(DTM) techniques, [12, 24, 34] have gained traction over the past few years as a means to mitigate problems associated with rising temperatures and to maintain performance at an acceptable level. DTM techniques rely on the knowledge of temperatures across the chip to guide their decisions. When designing chips, we need to evaluate a huge number of potential solutions to come up

with the most optimum DTM approach.

Given the vastness of the solution search space, temperatures across the chip need to be evaluated several times when performing microarchitecture simulations. We have to rely on thermal simulations instead of hardware thermal sensors to estimate module temperatures during design time. Consequently, efficient thermal simulation plays a vital role in optimizing the floorplan and selecting a suitable DTM technique. This becomes even more critical when we move to chip multi-processors as both the size of the system as well as the solution space increase dramatically. We address this issue by developing a fast thermal simulation approach that works for both uniprocessor and chip-multiprocessor systems.

Our technique is based on moment matching and moves most of the computation involved in determining module temperatures offline resulting in speedups of two orders of magnitude when compared to conventional thermal simulation approaches. Our technique suffers from a minor loss in accuracy, approximately 2.7°C, which is comparable to the estimation errors of thermal sensors [2]. During temperature computation, if we limit the impact of lateral heat conduction of a block to a few surrounding blocks, we achieve an average speedup of $1900X$ with the same loss in accuracy.

The rest of the chapter is organized as follows. In Section 5.2, we briefly describe related work and highlight the contributions of our work. Section 5.3 describes the basics of thermal modeling and simulation. In Section 5.4, we describe our technique for speeding up thermal simulation using moment matching. Section 5.5 describes our simulation framework. We present results on single core and chip-multi processors in Section 5.6. Finally, Section 5.7 describes the influence of lateral heat conduction on estimating module temperatures and leverages this information to further speedup thermal simulation.

5.2 Related Work

There are several approaches to thermal simulation such as solving the heat transfer equation by the finite difference method [19] and using Green functions [80] among others. At the architecture level, a thermal modeling and simulation tool called HotSpot [68] was developed to investigate various thermal management techniques and has become the de facto standard in the research community.

HotSpot makes use of the processor floorplan and package information and abstracts a thermal model in the form of thermal resistance and capacitance matrices. Temperatures of various modules in the processor are then determined by solving the system repeatedly for a given power trace of the modules. To evaluate different DTM techniques, HotSpot is coupled to a cycle accurate architecture simulator such as Wattch [13] or PTScalar [43] that estimate powers of various micro-architecture blocks. The average power of modules over regular intervals are fed to HotSpot and thermal simulations are performed to estimate the temperature trace of all modules. The runtime overhead of HotSpot comes from using numeric techniques such as Backward Euler or Runge-Kutta methods to solve the thermal system. As the size of the system increases with more cores being integrated on a single die, runtime penalty may become unacceptable.

Similar to our work, the authors in [46] also use a moment matching based approach to speed up thermal simulation but there are several key differences with our work. [46] does not model leakage power, which is becoming an increasing fraction of the total power with every new technology generation. Leakage power needs to be taken into account in thermal simulations to accurately estimate module temperatures. [46] relies on the power traces of modules being periodic. In multi-core and multi-threaded systems, tasks are migrated between different cores and threads may be swapped in and out of a core by the scheduler to maintain a desired thermal

profile and to balance performance. The unpredictable workload makes power traces non-periodic. [46] would have to perform moment matching when the power profiles of the tasks being executed change. In contrast to [46], our approach is independent of the tasks being run and depends only on the processor floorplan and package information. This allows us to move most of the computation offline, resulting in a significant speedup.

5.3 Thermal Modeling

This section briefly describes the thermal modeling and simulation steps used in HotSpot, which acts as the baseline for comparing the accuracy and efficiency of our approach. HotSpot uses the well known duality between heat diffusion in a system and the flow of current in an electrical network to model and simulate the temperature profiles of modules. Each module in the processor floorplan is represented by its equivalent thermal resistance and capacitance. Heat spreader layers and heat sinks are also modeled this way. The entire thermal system is represented by creating a mesh of thermal resistances and capacitances. Heat is conducted vertically from the active layer through the heat spreader to heat sinks. Lateral heat conduction between modules in the active layer is also modeled by thermal resistances between blocks. The entire thermal system is modeled as,

$$GT + C\frac{dT}{dt} = P \tag{5.1}$$

where G and C are thermal conductance and capacitance matrices, T and P are the temperature and power vectors of each node in the system. The power of each block can be further expressed as, $P = P_{dyn} + P_{Leak}(T)$, where P_{dyn} is the dynamic power and P_{Leak} is the leakage power of the block at temperature T.

The system represented by equation (5.1) is solved in time domain by reducing it

into a linear system

$$\left(G + \frac{C}{h}\right).T(t) = P_{dyn} + P_{Leak}(T(t-h)) + \frac{C}{h}.T(t-h) \tag{5.2}$$

using the Backward Euler technique with a small time step h. At every time step, the right hand side of equation (5.2) is evaluated and the equation is solved to get the values of $T(t)$. The time, t is then incremented by h and the process is repeated. This continues till we have time-stepped to the end of the desired simulation interval. We use the Backward Euler technique since the matrix on the left hand side(LHS), $\left(G + \frac{C}{h}\right)$, remains constant. This enables us to preprocess it once and use it to solve for temperatures in every time step. The computation involved in solving an N node system is thus $O(N^2)$ for each time step. We do not include the complexity of processing the LHS matrix since we can perform that computation offline. The power trace for the modules in the floorplan are obtained from simulators such as Wattch or PTScalar that have hardware access counters to monitor the activity of each block and report the average dynamic power of the blocks in the simulation interval. The entire flow is illustrated in Figure 5.1. We use this technique as the baseline to evaluate the efficiency and accuracy of our method. In the next section, we describe our fast moment matching based thermal simulation technique

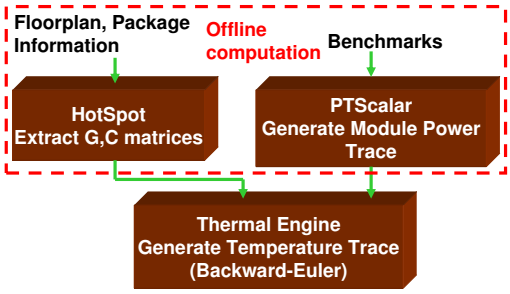


Figure 5.1: Conventional Thermal Simulation

5.4 Fast Thermal Simulation using Moment Matching

In this section, we describe how we apply moment matching techniques to estimate the temperature profiles of different modules in the system. For ease of illustration, we first assume zero initial conditions and no leakage power in the blocks(Sec. 5.4.2). Later on, we modify the technique to account for leakage power and non-zero initial conditions(Sec. 5.4.3). In Section 5.4.1, we review some basics of moment matching.

5.4.1 Moment Matching Preliminaries

Large linear systems can be analyzed quickly through the use of model order reduction methods in which a higher order system is approximated by a lower order system that characterizes the original system. The moments of the transfer function of the original system can be used to compute an equivalent lower order system. Consider a linear system whose transfer function is represented by $f(t)$. The i^{th} moment of the transfer function is given by

$$m_i = (-1)^i \int_0^\infty \frac{t^i}{i!} f(t)dt \tag{5.3}$$

The Taylor series expansion of the Laplace transform of the transform function can be expressed as

$$F(s) = m_0 + m_1 s + m_2 s^2 + m_3 s^3 + \cdots$$

Once the moments of the transfer function are computed, an equivalent lower order system can be generated by computing an [L/M] Padé approximant, where L denotes the degree of the numerator polynomial and M denotes the degree of the denominator polynomial of the transfer function of the reduced system [60]. For instance, if we wanted to represent an arbitrary transfer function as a [0/1] Padé approximant,

$$m_0 + m_1 s + m_2 s^2 + \cdots \equiv \frac{a_0}{1 + b_1 s} \tag{5.4}$$

We match the power of s on both sides of equation (5.4) and obtain the values of a_0 and b_1 as m_0 and $\frac{-m_1}{m_0}$ respectively [60]. In this way, we can represent any linear system as an equivalent lower order system. The process of matching the powers of s of the moments of the transfer function with a lower order system is known as moment matching and is useful to efficiently analyze complex linear systems.

5.4.2 Temperature Computation using Moment Matching

We can use the moment matching approach discussed in Section 5.4.1 to calculate the temperatures of different modules in the floorplan. In this section, we describe a technique in which we compute the thermal transfer functions of different modules once and use them to compute module temperatures for any power trace. Our technique significantly speeds up thermal simulation by eliminating the need to perform moment matching every time the power trace changes.

Thermal Characterization of the Floorplan

To efficiently compute the temperature vector of all modules for an arbitrary power trace, it is essential to estimate the thermal characteristics of the underlying system. We can find the thermal transfer function of the system by computing the impulse response of the linear time invariant system. Once we have estimated the transfer function of the system, the response to any power trace (treated as step inputs) can be efficiently computed.

We need to find the thermal impact of the power consumed by a module on itself and other modules. For instance, if a module a has a power of P_a, it would result in a rise in temperature of module a as well as its neighboring modules due to lateral conduction in the active layer. To estimate this effect and to compute the impulse response of the system, we iterate through all the modules in the floorplan exciting

the modules one at a time with a unit power source. We then compute the 0^{th} and 1^{st} order moments of the temperature response at all modules as [60]

$$\begin{aligned} G \cdot M_0 &= P \\ G \cdot M_1 &= -C \cdot M_0 \end{aligned} \tag{5.5}$$

where, G is the thermal conductance matrix of the floorplan, C is the thermal capacitance vector and P is the vector of module powers. We note here that G and C matrices are extracted using HotSpot. When we are performing thermal characterization of module i, $P[i]$ is 1 and the rest of the entries in P are 0. M_0 and M_1 are vectors containing the moments of the temperature response of all modules. After computing the moment vectors M_0 and M_1, we compute the [0/1] Padé approximants from equation (5.4). Similar to the modified nodal analysis of electric circuits, we drop the ambient node from the system shown in Equation (5.5) to make the computation more efficient [60]. So, the M_0 and M_1 vectors, characterize the increase in temperatures with respect to the ambient temperature of the system.

Performing this computation once would characterize the impact of one block on the rest of the system. We repeat this process for all blocks in the floorplan to characterize the full system. The entire procedure is described in Algorithm 3. $A_0[i][j]$ and $B_1[i][j]$ together describe the thermal response at module j due to a unit power excitation in module i. Intuitively $A_0[i][j]$ models the sensitivity of module j due to changes in the power profile of module i and $B_1[i][j]$ models the speed at which module j responds to changes in module i.

Temperature Computation

Once we completely characterize the thermal behavior of the system, we use it to estimate the temperature of all modules in the floorplan for a given power trace of the modules. Let the thermal transfer function in the s domain at module i due to

Algorithm 3 Thermal Characterization of the Floorplan

1: **procedure** $[A_0, B_1]$=ThermalCharacterize(Φ_p,G,C) ▷ Φ_p = Set of blocks in the floorplan
▷ G = Thermal Conductance matrix
▷ C = Thermal Capacitance vector
▷ A_0, B_1 = Matrices storing Padé approximants
2: $G_1 \leftarrow G^{-1}$
3: P = Vector of Module Powers
4: **for** $B_i \in \Phi_p$ **do**
5: Reset Module Powers
6: $P[B_i] \leftarrow 1$
7: $M_0 = G_1 \cdot P$ ▷ 0^{th} order moment
8: $M_1 = -G_1 \cdot CM_0$ ▷ 1^{st} order moment
9: **for** $k \in \Phi_p$ **do**
10: $A_0[i][k] \leftarrow M_0[k]$
11: $B_1[i][k] \leftarrow -M_1[k]/M_0[k]$
12: **end for**
13: **end for**
14: **end procedure**

module j computed by Algorithm 3 be

$$H(s) = \frac{a_{0ji}}{1 + b_{1ji}s}$$

Where a_{0ji} and b_{1ji} correspond to the entries $A_0[j][i]$ and $B_1[j][i]$ from Algorithm 3. If module j has power P_j, the thermal response at module i due to this excitation is given by

$$\Delta T_{ji}(s) = \frac{P_j \cdot a_{0ji}}{s \cdot (1 + b_{1ji}s)} \tag{5.6}$$

We have $\Delta T_{ji}(s)$ instead of $T_{ji}(s)$ because the response estimates the change in module temperatures from the ambient. Splitting Equation (5.6) into partial fractions and taking the inverse Laplace transform we get the effect module j has on module i in time domain for a given power. For a time interval t_{int}, over which the average module power is P, the change in temperature can be evaluated very efficiently.

$$\Delta T_{ji}(t) = P_j \cdot a_{0ji} \left(1 - e^{-t/b_{1ji}}\right) \tag{5.7}$$

Equation (5.7) computes the increase in temperature in module i due to module j. The net change in temperature of module i is obtained from superposition as follows

$$\Delta T_i(t) = \sum_{j=1}^{N} P_j \cdot a_{0ji} \left(1 - e^{-t/b_{1ji}}\right) \tag{5.8}$$

The absolute temperature of module i can be obtained as $T_{ambient} + \Delta T_i$. This process is illustrated in Figure 5.2

5.4.3 Modeling Non-zero Initial Conditions and Leakage Power

The discussion in Section 5.4.2 does not consider nonzero initial conditions when estimating the temperature responses of modules. However, in order to accurately compute the temperatures of modules in a processor executing a set of tasks, we need to consider nonzero starting temperatures at the beginning of every time interval except the first.

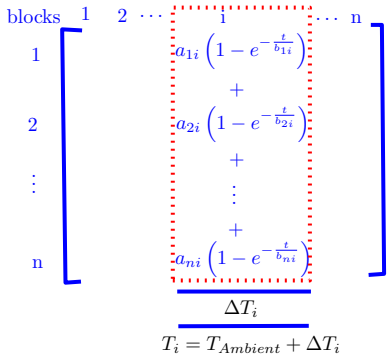


Figure 5.2: Computing Module Temperatures

We address this issue by observing that the overall system response can be obtained as the sum of a zero-input response and a zero-state response. The zero-input response corresponds to nonzero initial conditions and the zero-state response corresponds to zero initial conditions. In Section 5.4.2, Equation (5.7) describes the zero-state response. The zero-input response for the same system is similar to that of a simple RC circuit with nonzero initial charge and is given as

$$\Delta T_{ji}(t) = \Delta T_{ji}(t-h) \cdot e^{-t/b_1 ji} \tag{5.9}$$

Where $\Delta T_{ji}(t-h)$ denotes the initial value at the start of time interval t. The overall response is now obtained by adding the zero-input response to Equation 5.7 and is given as

$$\Delta T_{ji}(t) = P_j a_{0ji} + \left(\Delta T_{ji}(t-h) - P_j a_{0ji}\right) e^{-t/b_1 ji} \tag{5.10}$$

Leakage Power Modeling

Modeling sub-threshold leakage in the framework of moment computation is a challenge. The leakage power of a module $P_L(T)$ at temperature T with area proportional to A and supply voltage V_{dd} is given by [68],

$$P_L(T) = AT^2 e^{-\frac{\alpha V_{dd}+\beta}{T}} \tag{5.11}$$

where α and β are empirical constants depending on whether the module is a functional unit or a memory unit. Due to the non-linear dependence on temperature, leakage power in Equation 5.11 cannot be modeled in the framework of moment computation.

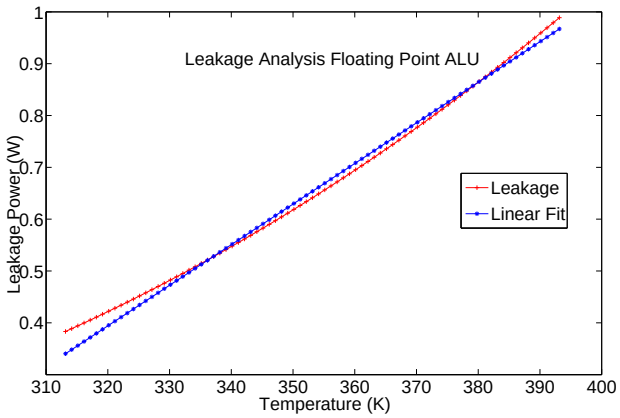


Figure 5.3: Leakage Model

To overcome this problem, we approximate the leakage power by a linear model[1], $P_{Leak} = C_2T + C_1$. This approximation allows us to incorporate leakage power in the computation framework described in Section 5.4.2. For each module in the floorplan we generate the approximate linear leakage power model by using information regarding the size of the module and the α and β parameters from [68]. Figure 5.3 shows the leakage power and the linear fit for a floating point ALU in the floorplan. We observe that the linear model follows Equation 5.11 quite closely in the temperature range (313.15 K - 395 K) that we are interested in. The models for other modules in the floorplan behave similarly. Figure 5.4 shows a part of the system in which block i is augmented by its leakage model. It is connected to several neighboring blocks, $N_j's$ through lateral conductances. In the vertical direction, it has some conductance

[1]We use the exponential leakage model for the conventional technique

to the heat spreader layer which in turn connects to the heat sink. P_{dyn} represents the average dynamic power of block i in the current time interval and $P_{Leak} = C_2T$ from our linear model.

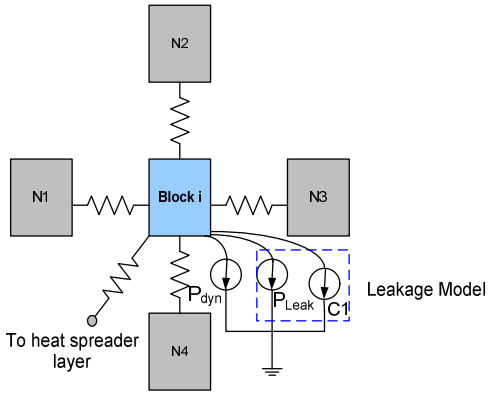


Figure 5.4: Modeling Leakage in the Thermal System

Revisiting the duality between thermal and electric systems, we observe that power and temperature in thermal RC circuits are equivalent to current and voltage in electric RC circuits respectively. The linear leakage model is clearly equivalent to a voltage controlled current source in electric circuits. These models are then stamped into the thermal conductance matrix (G) using techniques presented in [60]. Once leakage models are stamped into the conductance matrix, the techniques described in Section 5.4.2 to characterize the floorplan are still valid.

Algorithm 4 describes the procedure of computing module temperatures considering both leakage power and non-zero initial conditions. If there are N_B blocks in the system, the time complexity of Algorithm 4 is $O(N_B^2)$. We do not include the time complexity of Algorithm 3 in the overall complexity of our approach since it is run offline and does not create any additional overheads in computing module temperatures. In comparison, the time complexity of the conventional transient simulation presented in Section 5.3 is $O(N_{steps} \cdot N^2)$. N_{steps} is the number of steps required

in the transient simulation. It is to be noted that $N > N_B$ since N includes the entire system including the blocks, heat spreader layers and heat sinks (e.g., in our experiments $N_B = 20$ and $N = 71$ for a single core processor).

Algorithm 4 Computing Module Temperatures

1: **procedure** T=ComputeTemperature(P,Φ_p,ΔT,A_0,B_1) T : Module Temperatures; P : Set of module powers; Φ_p : Set of blocks in the floorplan; ΔT : Matrix s.t. $\Delta T[i][j]$ is $\Delta T_{ij}(t-h)$ in Equation 5.10; A_0,B_1: obtained from Algorithm 3
2: **for** $B_i \in \Phi_p$ **do**
3: $\Delta T_{local} \leftarrow 0$
4: **for** $k \in \Phi_p$ **do**
5: Update $\Delta T[k][i]$ according to Equation 5.10
6: $\Delta T_{local} \leftarrow \Delta T_{local} + \Delta T[k][i]$
7: **end for**
8: $T[B_i] \leftarrow T_{ambient} + \Delta T_{local}$
9: **end for**
10: return T
11: **end procedure**

5.5 Simulation Framework

We validate our fast thermal simulation technique on a microarchitecture similar to the Alpha 21264 design scaled to 65nm running at 2.0GHz. The architecture configuration is shown in Table 5.1. Figure 5.5 shows the floorplan of the processor core that we use. The *L2* cache (not shown) is wrapped around the core. We use PTScalar [43] to generate power traces for all modules in the microarchitecture. We profile a set of benchmarks from the Spec2000 [3] benchmark suite

Table 5.1: Architecture Details

Block	Configuration
Register Update Unit (RUU)	16 Instructions
Load Store Queue (LSQ)	8 Instructions
Fetch queue	4 Instructions
Issue Width	4 Instructions
Decode Width	4 Instructions
Commit Width	4 Instructions
Integer ALU	3 Adders, 1 Multiplier
Floating point ALU	1 Adder, 1 Multiplier
Branch predictor	Combined, Bimodal 2K entries, 2 level with 1K entries and 8 bit history
Instruction TLB	64 entry fully associative
Data TLB	128 entry fully associative
L1 Data Cache	16KB, 4-way set associative with 32B blocks
L1 Instruction Cache	16KB, direct mapped with 32B blocks
L2 Cache	256KB, 4-way set-associative with 64B blocks

(gcc,gzip,vpr,bzip2,mgrid,art,equake) to generate power traces. Similar to [24], we use a $100ms$ sampling period for our thermal simulations. So, in our profiling runs, we gather traces of dynamic power averaged over each sampling period for the entire simulation length of benchmarks. We run the benchmarks for 10 billion cycles and generate power traces once and use them in all subsequent experiments.

After generating power traces, we use the flow illustrated in Figure 5.1 for the conventional transient simulation. The speed and accuracy of the conventional technique depends on the step size used in the Backward-Euler method. As alluded to in Section 5.4.2, the time complexity of the conventional technique is $O(N_{steps} \cdot N^2)$. So, the choice of step size is critical in achieving accurate results within reasonable runtimes. We performed full transient simulations on a few benchmarks to determine the best step size. We used a step size of $1ns$ as our baseline to compute the

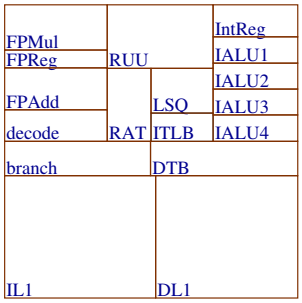


Figure 5.5: Processor Core

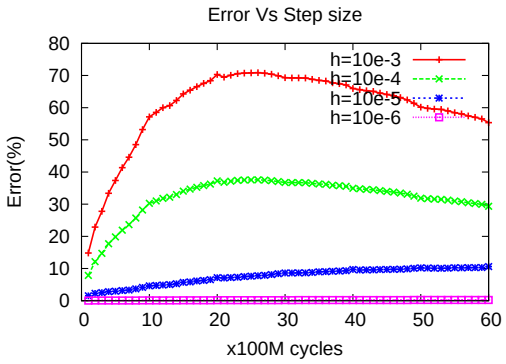


Figure 5.6: Error-Step size Tradeoff

estimation error. Figure 5.6 shows the results of our studies. Based on the results, we decided to use a step size of $0.01ms$ (h=10e-6 in the figure) for the conventional technique.

The overall flow of our methodology is illustrated in Figure 5.7. We first characterize the thermal behavior of the processor offline and use power traces generated from our profiling runs to compute module temperatures. We compare the temperature traces obtained by our technique with the traces generated by the conventional technique in terms of accuracy and runtime.

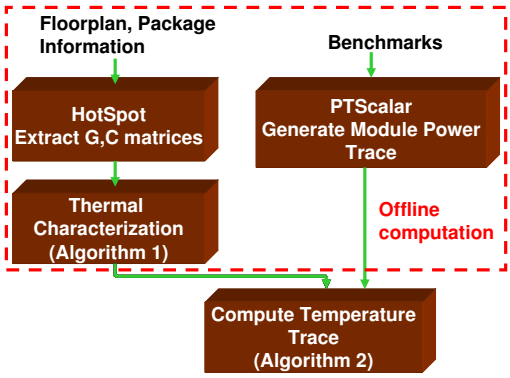


Figure 5.7: Our Thermal Analysis flow

5.6 Results

We conducted experiments by creating several job schedules for the processor with the power traces we generated; and analyzed the accuracy and speedup achieved by our technique. We present our findings in this section. We first describe the results for a single-core processor system in Section 5.6.1. Results of multi-core systems are discussed in Section 5.6.2.

5.6.1 Single-Core Systems

We use the floorplan in Figure 5.5 for our experiments. In Figure 5.8, we present the results for a job schedule that we used. In each of the schedules, we launch a task and completely simulate its power trace before moving on to the next task. The plot labeled FastTr gives the temperature trace of our technique and the one labeled FullTr is the trace obtained by the conventional technique. Temperature traces give the variation of temperatures of the register files, which are typically the hottest regions on the chip. From Figure 5.8, it is clear that the temperature trace obtained from our technique closely follows the conventional technique. The performance and accuracy comparisons between our technique and the conventional

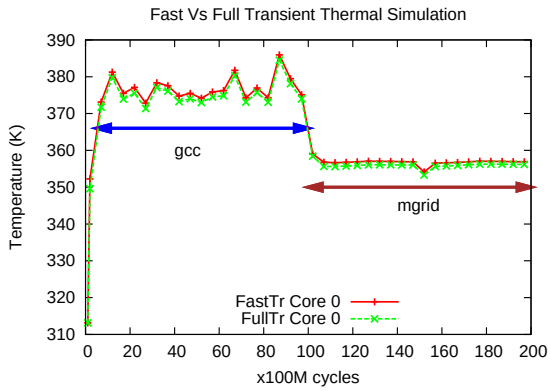


Figure 5.8: Single-Core system: Core 0 running gcc followed by mgrid (Workload 1)

approach are described in the row noted by 1 core in Table 5.2 (more details are given in Section 5.6.2). On average, our technique was **312X** faster than the conventional technique with an error in estimation of 2.7°C.

5.6.2 Multi-Core Systems

We tested the applicability of our technique on multi-core systems under different scheduling scenarios with tasks launched on different processor cores at different times. We conducted experiments on 2, 4, 8 and 16 core microarchitectures and present our results here.

We did not have access to real floorplans and cycle accurate simulators for the multi-core systems that we studied. To test our technique, we had to make the following modeling assumptions.

- **Choice of Floorplans:** We used the floorplan in Figure 5.5 for each core in the system. To generate the multicore floorplan, we estimate the number of cores to be placed in a row to maintain an aspect ratio close to 1. The individual cores are then placed side-by-side. Once a row of cores has been placed, we

move to the next row and repeat this process. The $L2$ cache is then wrapped around the cores.

- **Processor Communication:** Since we did not have access to architectural simulators for multi-core systems, we could not model the communication between cores. For all the schedules that we ran on the multi-core architectures, we assumed that the memory accesses of tasks are completely independent creating no coherency related issues.

  We want to evaluate our thermal analysis technique under different scenarios where tasks may start and end on different cores at different times. Our interest is limited to the thermal aspects of the system, which depend only on the floorplan and the power trace. We can easily incorporate communication between cores if we had access to architecture simulators capable of handling multi-core systems. The communication unit will be modeled as an additional block in the floorplan with its associated power.

Temperature Graphs

We first present graphs of temperature traces from our multi-core architecture experiments to demonstrate that our thermal analysis technique works well. We provide a detailed description of the accuracy and performance benefits of our approach compared to the conventional approach later in the section.

Figure 5.9 shows the results for a schedule on a dual-core architecture. The scheduler launches gzip on Core 0 on startup. Core 1 is assigned gcc after 3 s. We observe from the figure that our technique works well even when one Core switches while another core is active.

Sample temperature traces for 4, 8 and 16 core systems obtained by running a set of tasks on each is shown in figures 5.10, 5.11 and 5.12 respectively. Due to space

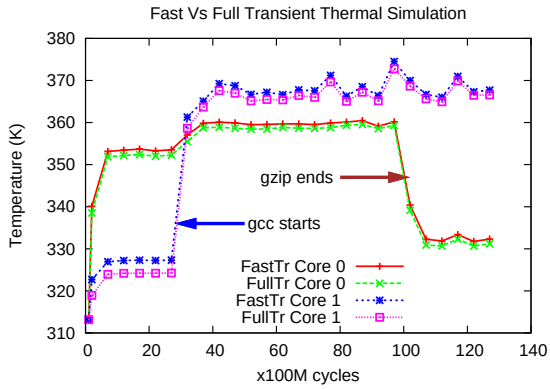


Figure 5.9: Dual-Core System: (Workload 4)

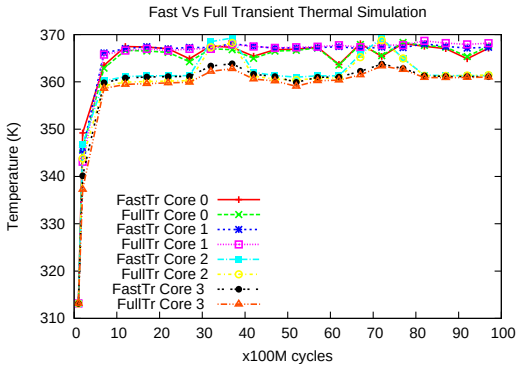
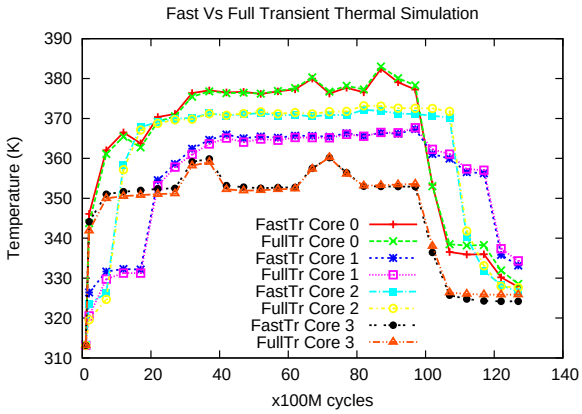
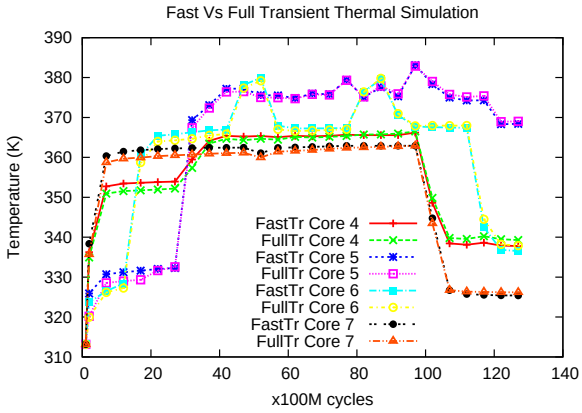


Figure 5.10: Four-Core System: (Workload 6)

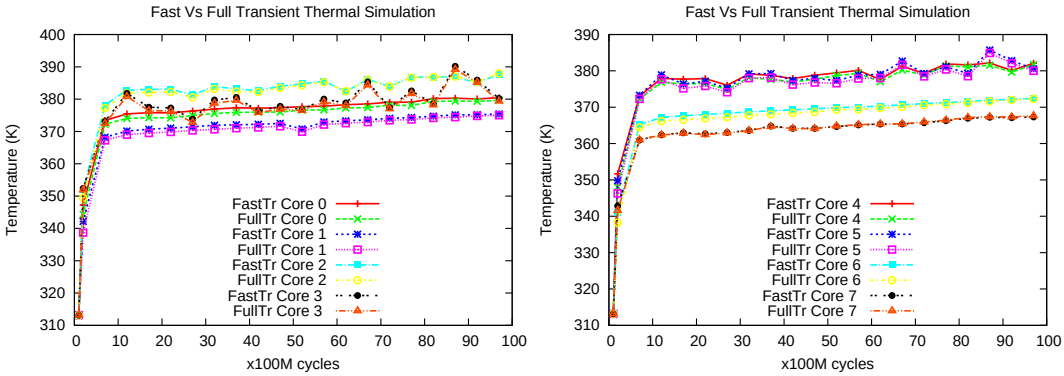


(a) Cores 0-3



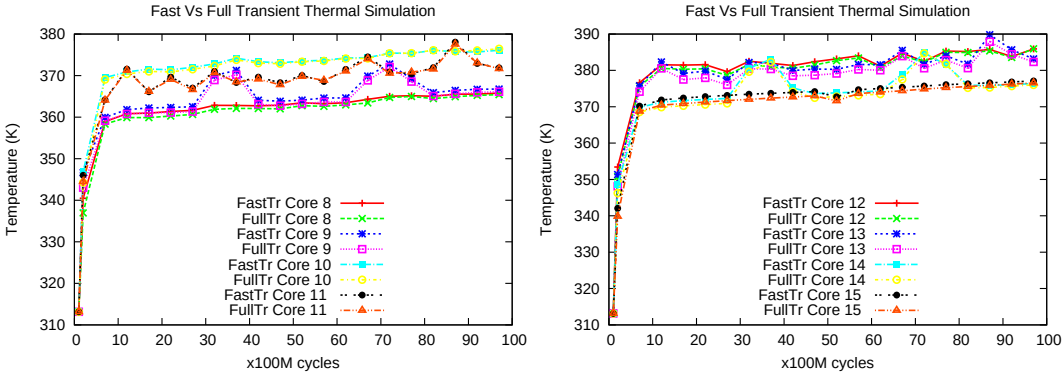
(b) Cores 4-7

Figure 5.11: 8-Core System: (Workload 9)



(a) Cores 0-3

(b) Cores 4-7



(c) Cores 8-11

(d) Cores 12-15

Figure 5.12: 16-Core System: (Workload 11)

limitations, we have shown results for only one of the schedules we implemented on each system. More details about the error and performance are given in Section 5.6.2.

From the graphs, we observe that the temperature traces computed with our technique closely follow the conventional technique. To quantify the effectiveness of our technique, we measured the performance and accuracy of our technique and compared it with the conventional technique. We summarize the results in the next section.

Accuracy and Performance Comparisons

The column labeled *Cores* in Table 5.2 gives the number of processor cores in the architecture, *Workld* gives the workload used. Each workload in the table corresponds to a set of tasks and their associated start times on each core (e.g. in workload 1, the processor executes gcc for 100M cycles immediately followed by mgrid). C_{ON} gives the average error of our technique in estimating the temperatures for cores that are actively executing tasks. C_{OFF} gives the average error for cores that are idle. Errors in C_{OFF} are important because lateral heat conduction between cores is becoming more significant and we need to make sure that our technique accurately models this effect. $Max.Err.$ gives the maximum estimation error of all cores over the complete schedule. The runtimes of our technique and the conventional approach (both implemented in C++) are given in columns $FastTr$ and $FullTr$ respectively. The speedup of our technique expressed as a ratio is given in column $Impr$.

We observed that over all our simulation configurations, the average estimation error for active cores was less than 1°C. For idle cores, the error was about 1.5°C. The maximum error was expectedly higher and was about 2.7°C. We performed further analysis of the experimental data and determined that the maximum error occurs during two scenarios. The first is during system startup when all cores are at am-

Table 5.2: Accuracy and Performance Comparison of our Technique Vs. the Traditional Method

Cores	**Workld**	**Error (°C)**			**Runtime (s)**		
		C_{ON}	**C_{OFF}**	**Max. Err**	**Fast Tr**	**Full Tr**	**Impr.**
1	1	1.14	-	2.64	0.08	26.92	336.5
	2	1.23	-	2.92	0.06	18.35	305.83
	3	1.09	-	2.80	0.06	17.72	295.33
2	4	1.22	2.21	2.50	0.09	32.28	358.67
	5	0.82	1.99	2.60	0.14	49.89	356.36
4	6	0.59	2.19	2.30	0.21	79.43	378.24
	7	0.88	0.66	2.40	0.31	178.47	575.71
	8	0.68	1.02	2.83	0.80	574.75	718.43
8	9	0.75	1.39	2.79	0.68	357.74	526.09
	10	0.69	1.35	2.86	0.90	553.37	614.85
16	11	1.05	2.83	3.12	2.86	1758.38	628.80
Avg.		**0.89**	**1.55**	**2.69**			**441.29**

bient temperature and tasks launched on some cores cause temperatures to ramp up rapidly. The second scenario is when a core completes executing a task and is allocated a new task by the scheduler. When the power profiles of these tasks vary significantly, we observe the maximum error at the instant a core switches tasks. In both cases, the maximum error occurs only momentarily and the estimation error quickly settles to a value closer to the average error. We note here that the error of our technique compares favorably to hardware thermal sensors [2]. Across all our simulation configurations, our technique resulted in a 441X speedup over the conventional approach on average. This is expected from our complexity analysis of Algorithm 4 in Section 5.4.2. We observe that as we increase the number of cores, the speedup of our technique also increases.

In the next section, we analyze the thermal impact of modules on their neighbors and leverage this information to further speedup thermal simulation.

5.7 Influence of Blocks on their Neighbors

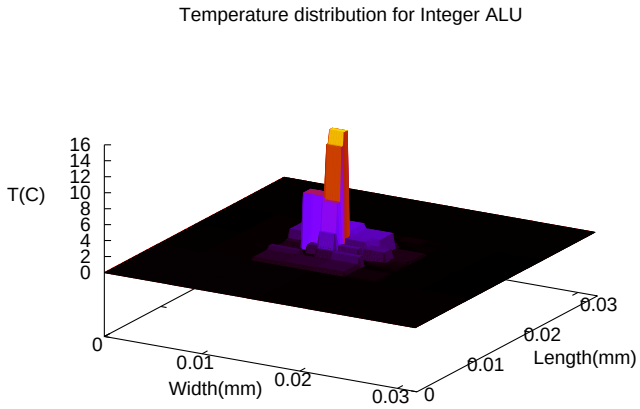
Due to lateral heat conduction, every block in the floorplan contributes to the temperatures of its surrounding blocks. However, the impact of lateral conduction through the active layer and substrate reduces rapidly with distance. Consequently, every block has a sphere of influence beyond which other modules have no impact on its temperature. When performing thermal simulations, we can take advantage of this and achieve further speedup over that reported in Section 5.6 by considering only the sphere of influence for each module instead of all modules in the floorplan when computing its temperature.

We studied the thermal influence of a set of neighboring blocks on a block by running power intensive tasks on the processor and profiling the temperatures of the top 10 hottest blocks. For each selected block, we study the temperature contribution from all blocks in the processor. This will provide us with information about the sphere of influence around each block. If the temperature of block j is influenced by k surrounding blocks, its temperature is computed by adding the temperature contribution from each of the k blocks as

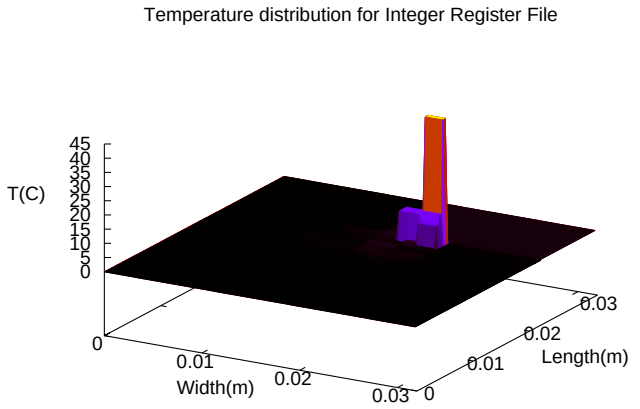
$$T_j(t) = T_{ambient} + \sum_{i=1}^{k} \left(P_i a_{0ij} + (\Delta T_{ij}(t-h) - P_i a_{0ij}) e^{-t/b_{1ij}} \right) \qquad (5.12)$$

If we do not limit the sphere of influence around a block, k will be equal to the total number of blocks in the floorplan. The rest of the terms in Equation 5.12 are identical to Equation 5.10.

Figures 5.13(a) and 5.13(b) illustrate the result of our profiling studies. We show the results for an Integer ALU and register file, which were the hottest modules in our



(a) Integer ALU



(b) Integer Register File

Figure 5.13: Sphere of Thermal Influence around Modules

studies. The X and Y axes show exact (x, y) coordinates on the chip and the Z axis gives the temperature contribution in °C from a specific (x, y) location to the block being analyzed. In essence, we are visualizing Equation 5.10. The Z axis gives the value of ΔT for each (i, j) location. The temperature of the block can be obtained by adding the temperature contribution from all blocks (the Z values in the plot) to the ambient temperature.

The highest temperature values in Figures 5.13(a) and 5.13(b) correspond to the locations of the Integer ALU and register file respectively. This indicates that the maximum temperature contribution to a block is from itself. This is because the impact of lateral heat conduction on a block is dominated by the temperature increase due to power consumed by the block itself. Furthermore, it is evident that the impact of lateral conduction decreases rapidly with distance and considering the impact of only a few neighboring blocks for each block should be sufficient to accurately compute the temperature of a module. This could potentially result in a significant reduction in the value of k used in Equation 5.12 leading to further speedup of thermal simulation especially for multi-core systems with a large number of blocks in the floorplan.

5.7.1 Accuracy and Runtime Tradeoffs

From Figures 5.13(a) and 5.13(b), we observe that we can speedup thermal simulation further by considering a limited number of neighbors (k in Equation 5.12) when computing the temperature of each block in the floorplan. In this section, we study the accuracy and runtime tradeoffs involved in varying k. The purpose of this study is to identify the optimum value of k to give the most improvement in runtime at a minimal loss in accuracy.

We use workloads from Table 5.2 for this study. For each workload, we vary the number of neighboring blocks used to evaluate the temperature of every block

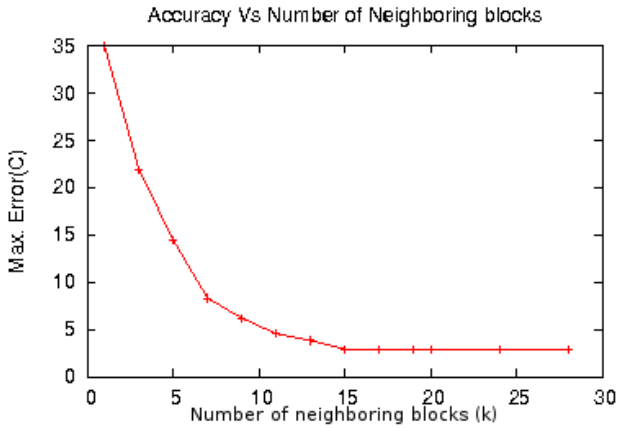


Figure 5.14: Temperature Estimation Errors

and perform thermal simulation using the fast moment matching based approach presented in Section 5.4. Full transient thermal simulations presented in Section 5.3 are used to generate the accurate baseline temperatures of modules. We then gather statistics of the maximum error in estimating module temperatures for each workload. Figure 5.14 shows the average of maximum estimation errors over all workloads. The X-axis shows the number of blocks whose contribution is considered for calculating the temperature of each block (k in Equation 5.12) and the Y-axis shows the maximum estimation error. From the graph, we observe that the error drops rapidly initially. However, as we increase the number of neighboring blocks considered beyond fifteen, the maximum error stabilizes and shows negligible improvement. The value at which the error stabilizes is equal to the error reported in Table 5.2, where lateral heat conduction from all blocks in the floorplan is considered.

Based on this study, we set the number of neighboring blocks to be considered for each block to be fifteen and repeat our experiments from Table 5.2 to analyze the speedup and accuracy tradeoffs involved. We present our results in Table 5.3.

The columns labeled C_{OFF}, C_{ON}, *Workld* and *Max. Err* are similar to those in

Table 5.3: Accuracy and Performance Comparison of our Technique Vs. the Traditional Method

Cores	**Workld**	**Error (°C)**			**Runtime (s)**		
		C_{ON}	**C_{OFF}**	**Max. Err**	**Fast Tr with 15 Neighbors**	**Fast Tr from Table 5.2**	**Impr.**
1	1	1.16	-	2.67	0.06	0.08	1.33
	2	1.25	-	2.93	0.05	0.06	1.20
	3	1.10	-	2.82	0.05	0.06	1.20
2	4	1.23	2.23	2.52	0.04	0.09	2.25
	5	0.82	2.02	2.61	0.05	0.14	2.80
4	6	0.60	2.21	2.33	0.04	0.21	5.25
	7	0.89	0.69	2.41	0.06	0.31	5.17
	8	0.70	1.04	2.86	0.15	0.80	5.33
8	9	0.76	1.40	2.81	0.05	0.68	13.60
	10	0.70	1.38	2.86	0.05	0.90	18.00
16	11	1.07	2.86	3.14	0.14	2.86	20.42
Avg.		**0.91**	**1.58**	**2.86**			**4.32**

Table 5.2. The column *Fast Tr from Table 5.2* gives the runtime of our fast thermal simulation technique when lateral heat conduction from *all* blocks in the floorplan is considered. The column *Fast Tr with 15 Neighbors* gives the runtime of our thermal simulation technique when the lateral conduction from only fifteen(based on Figure 5.14) neighboring blocks is considered. The error numbers listed in the table are computed with respect to a full transient simulation. The column *Impr* gives the speedup obtained by considering lateral conduction from only fifteen blocks when compared to considering lateral conduction from all blocks in the floorplan. We note here that the underlying thermal simulation technique is still our fast moment matching based technique presented in Section 5.4. The only difference is the number of blocks for which we consider lateral conduction.

We observe that on average, the estimation errors for both idle cores and active cores are nearly identical to the results in Table 5.2. The error for active (idle) cores was 0.89 (1.55)°C when lateral heat conduction from all blocks is considered and about 0.91(1.58)°C when lateral conduction from only 15 blocks is considered. The maximum errors were also nearly identical (2.69 and 2.86°C). This shows that considering lateral conduction from all blocks is an overkill and we can use this information to improve the runtime further.

On average, when we consider the lateral conduction from fifteen blocks, we achieve a speedup of $4.3X$ over our technique presented in Section 5.4. We observe that as the number of processor cores increase, more significant speedups are obtained. For a floorplan with sixteen processor cores, limiting the impact of lateral heat conduction to fifteen blocks results in a 20X speedup when compared to including the lateral heat conduction from all blocks. If there are N_B blocks in the system, the time complexity of computing module temperatures using Algorithm 4 is $O(N_B^2)$. When we consider the impact of lateral conduction from only fifteen blocks, the complexity of Algorithm 4 becomes $O(N_B)$ explaining the speedup we obtained in Table 5.3. We note here that this speedup is in addition to the $443X$ speedup reported in Table 5.2. The overall speedup we can achieve over a conventional transient simulation is the product of the speedups reported in Tables 5.2 and 5.3 and is $1900X$.

We observe that our technique, while being comparable to the accuracy of hardware sensors, offers three orders of magnitude speedup over a conventional technique making it well suited for the thermal analysis of highly integrated multi-core architectures. Our belief is that the speed of our technique can help in exploring the vast solution space of DTM techniques to meet the thermal and power challenges of the future.

Chapter 6

Conclusions

At the outset of this dissertation, we wanted to answer the following questions.

1. How can we bridge the gap between FPGAs and ASICs?
2. With continued process scaling, are process variations important for FPGAs and how do we modify tool flows to account for them?
3. How do we efficiently estimate and optimize the resilience of circuits to transient errors?
4. Can we efficiently estimate thermal stress in circuits to reduce the occurrence of permanent failures?

We addressed the first question in Chapter 2 by trading-off some flexibility of FPGAs to achieve improvements in performance, power and area. Our idea consisted of removing some programmable switches and replacing conventional switch boxes in the routing architecture with switch boxes that contain a mixture of programmable switches and hard wired connections between routing tracks. The routing tool is thus given an option to use faster hard-wired segments for performance critical wires. We could design circuits that operate about 18% faster than conventional designs on average, effectively reducing the gap between FPGAs and ASICs.

In Chapter 3, we analyzed the impact of process variations on circuits implemented on FPGAs. We also developed CAD algorithms to modify the routing tool to optimize for statistical timing criticality and improve timing yield in the presence of process

variations. We were able to achieve a 7.6X reduction in timing yield loss with our algorithm. We proposed a minor change in the clock distribution network of FPGAs that will enable designers to use clock skew to compensate for process variations. We combined the two techniques and improved timing yield loss by 9X or equivalently a 12% improvement in timing yield. We also achieved a 10% improvement in circuit performance over a purely deterministic CAD flow.

Chapter 4 addresses the third question we posed above. We proposed a novel reliability estimation technique that combines the efficiency of probabilistic techniques with the accuracy of symbolic techniques. While probabilistic and symbolic reliability estimation techniques are at two ends of the spectrum in terms of speed and accuracy, our technique offers a viable middle ground. In addition, we presented reliability optimization approaches that improved the resilience of circuits to transient errors by about 17%.

Finally, in Chapter 5, we proposed a novel approach to speed up thermal simulation of micro-processors. We noted that it is critical to efficiently estimate on-chip temperatures in order to schedule tasks to prevent the build up of thermal stress in digital systems. This is critical to avoid catastrophic permanent failures. The technique we proposed was based on moment matching and pre-characterized the thermal behavior of the processor floorplan. As a result of pre-characterization, we were able to achieve speedups of three orders of magnitude over a conventional thermal simulation technique. We generated several workloads and tested the speed and accuracy of our proposed approach. Based on extensive testing, we determined that the loss in accuracy of our technique was minor and comparable to modern hardware thermal sensors.

While we have touched upon all questions listed at the beginning of this section, our proposed solutions are by no means complete. We can extend our work in Chapter

2 by making the placement tool aware of the changes in the routing architecture. Doing this will lead to better quality results and help in achieving timing closure faster.

When we were developing our ideas presented in Chapter 3, there were other efforts in the research community that dealt with process variations in the technology mapping and placement phases of circuit design. We can combine all these research efforts and present a single unified design flow to the FPGA community. Also, the placement and routing algorithms are currently unaware of the ability to tune clock skews. Further changes to the algorithms are needed to leverage this information to achieve better quality results.

We plan to extend our work in Chapter 4 by exploring the effects of electrical masking on transient error propagation in digital circuits. Our work currently ignores this effect and we deal with worst case circuit reliability when performing optimization. However, considering electrical masking will provide a better global view of the problem and will open up new venues for optimization. Another interesting extension is to incorporate reliability as an optimization criterion during physical design.

While we have developed a fast technique to estimate thermal stress in microprocessors in Chapter 5, we have not performed task scheduling or floorplan optimization. Future work in this area will include optimizing floorplans by using the fast temperature estimation technique we have proposed. We also intend to explore different task scheduling options to alleviate thermal stress. This in turn would lead to reduced permanent failures.

In summary, we believe that the techniques we have proposed to address present and future challenges in circuit design hold promise and we look forward to exploring these topics in the future.

Bibliography

[1] Harp architecture generator and p&r tool url `http://www.ece.umn.edu/users/kia/` (click on the download link).

[2] *National Semiconductor, LM19 Temperature Sensor data sheet.*

[3] *SPEC. Standard Performance Evaluation Corporation CPU2000 Benchmarks.*

[4] Vpr pattern finder url, `http://www.ece.ucsb.edu/~express/software.html`, 2004.

[5] Charles J. Alpert, Dinesh P. Mehta, and Sachin S. Sapatnekar, editors. *Handbook of Algorithms for Physical Design Automation.* Auerbach Publications, 2008.

[6] C.J. Alpert, J. Hu, S. Sapatnekar, and P.G. Villarubia. A practical methodology for early buffer and wire resource allocation. In *Design Automation Conference*, pages 189–194, 2001.

[7] J.H. Anderson, F. Najm, and T. Tuan. Active leakage power optimization for FPGAs. In *International symposium on Field Programmable Gate Arrays*, 2004.

[8] Hossein Asadi and Mehdi B. Tahoori. Soft error hardening for logic-level designs. In *International Symposium on Circuits and Systems*, 2006.

[9] Berkeley-Device-Group. Predictive technology model. http://www.eas.asu.edu/ptm.

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

[11] D. Bhaduri and S. Shukla. Nanoprism: A tool for evaluating granularity vs. reliability trade-offs in nano-architectures. In *Great Lakes VLSI symposium*, pages 109–112, April 2004.

[12] David Brooks and Margaret Martonosi. Dynamic thermal management for high-performance microprocessors. In *International Symposium on High Performance Computer Architecture*, 2001.

[13] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimization. In *International Symposium on Computer Architecture*, 2000.

[14] Capo. A large scale fixed-die placer from UCLA. http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement.

[15] A. Chakraborty, K. Duraisami, A. Sathanur, P. Sithambaram, L. Benini, A. Macii, E. Macii, and M. Poncino. Dynamic thermal clock skew compensation using tunable delay buffers. In *International Symposium on Low Power Electronics and Design*, 2006.

[16] H. Chang and S. Sapatnekar. Statistical timing analysis under spatial correlations. *IEEE Transactions on Computer Aided Design if Integrated Circuits and Systems*, 24(9):1467–1482, 2005.

[17] S. Chang, L.P.P.P. van Ginneken, and M. Sadowska. Circuit optimization by rewiring. *IEEE Transactions on computers*, 48:962–970, 1999.

[18] Y.W. Chang and Y.T. Chang. An architecture-driven metric for simultaneous placement and global routing for FPGAs. In *Design Automation Conference*, pages 567–572, 2000.

[19] Yi-Kan Cheng, Prasun Raha, Chin-Chi Teng, and Elyse Rosenbaumand Sung-Mo Kang. Illiads-t: An electrothermal timing simulator for temperature-sensitive reliability diagnosis of cmos vlsi chips. *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, 17:668–681, 1998.

[20] K. Chopra, S.Shah, A. Srivastava, and A. Blaauw amd D. Sylvester. Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation. In *International Conference on Computer-Aided-Design*, 2005.

[21] Mihir R. Choudhury and Kartik Mohanram. Accurate and scalable reliability analysis of circuits. In *Design Automation & Test in Europe*, 2007.

[22] C. Clark. The greatest of a finite set of random variables. *Operations Research*, 9:85–91, 1961.

[23] B. Cline, K. Chopra, D. Blaauw, and Y. Cao. Analysis and modeling of cd variation for statistical timing. In *International Conference on Computer-aided Design*, 2006.

[24] Ayse Kivilvim Coskun, Tajana Simunic Rosing, and Kenny C. Gross. Proactive temperature balancing for low cost thermal management in mpsocs. In *International Conference on Computer aided Design*, 2008.

[25] J. Costa, L. Silveira, and S. Devadas. Power estimation using probability polynomials. In *Design Automation for Embedded systems*, 2004.

[26] A. Dehon. *Reconfigurable architectures for general-purpose computing.* PhD thesis, Massachusetts Institute of Technology, 1996.

[27] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Riccó. Testability measures in pseudorandom testing. *IEEE Transactions om Computer-Aided Design*, 11:6:784–800, 1992.

[28] P. Froedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos. Modeling within-die spatial correlation effects for process-design co-optimization. In *International Symposium on Quality of Electronic Design*, pages 516–521, 2005.

[29] A. Gayasen, K. Lee, N. Vijayakrishnan, M. Kandemir, M.J. Irwin, and T. Tuan. A dual-v_{DD} low power fpga architecture. In *International conference on Field Programmable Logic and its applications*, 2004.

[30] M.R. Guthaus, N. Venkateswaran, C.Viswesariah, and V.Zolotov. Gate sizing using incremental parameterized statistical timing analysis. In *International Conference on Computer-Aided-Design*, 2005.

[31] Jie Han, Erin Taylor, Jianbo Gao, and Jose Fortes. Faults, error bounds and reliability of nanoelectronic circuits. In *International Conference on Application-Specific Systems, Architecture and Processors*, 2005.

[32] J. P. Hayes, I. Polian, and B. Becker. A model for transient faults in logic circuits. In *International Workshop on Design and Test*, 2006.

[33] N. Jayakumar and S.P. Khatri. A metal and via maskset programmable VLSI design methodology using PLAs. In *International Conference on Computer Aided Design*, 2004.

[34] Ramkumar Jayaseelan and Tulika Mitra. Temperature aware task sequencing and voltage scaling. In *International Conference on Computer Aided Design*, 2008.

[35] B. Kapoor. Improving the accuracy of circuit activity measurement. In *Design Automation Conference*, 1994.

[36] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. Predictable routing. In *International Conference on Computer Aided Design*, 2000.

[37] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. Pattern routing: use and theory for increasing predictability and avoiding coupling. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(7):777–790, 2002.

[38] M. Khellah, S. Brown, and Z. Vranesic. Minimizing interconnection delays in array-based fpgas. In *IEEE Custom Integrated Circuits Conference*, pages 181–184, 1994.

[39] B. Krishnamurthy and I. G. Tollis. Improved techniques for estimating signal probabilities. *IEEE. Transactions on Computers*, 38:1041–1045, 1989.

[40] G. Krishnan. Low-cost easypath fpgas offer promise to assp companies. Technical report, XilinX Inc., www.xilinx.com/publications/xcellonline/xcell_53/xc_easyassp53.htm.

[41] Smita Krishnaswamy, George F. Viamontes, Igor Markov, and John P. Hayes. Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In *Design Automation and Test in Europe*, 2005.

[42] H. Kufluoglu and M.A. Alam. A computational model of nbti and hot carrier injection time-exponents for mosfet reliability. *Journal of Computational Electronics*, 3:165–169, 2004.

[43] W. Liao, L. He, and K.M. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *IEEE Tran. Computer-Aided Design of Integrated Circuits and Systems*, 24(7):1042–1053, 2005.

[44] Y. Lin and L. Hei. Stochastic physical synthesis for FPGAs with pre-routing interconnect uncertainty and process variation. In *International Syposium on Field Programmable Gate Arrays*, 2007.

[45] Y. Lin, M. Hutton, and L. Hei. Placement and timing for FPGAs considering variations. In *International Conference on Field Programmable Logic and Applications*, August 2006.

[46] Pu Liu, Zhenyu Qi, Hang Li, Lingling Jin, Wei Wu, Sheldon X.D. Tan, and Jun Yang. Fast thermal simulation for architecture level dynamic thermal management. In *International Conference on Computer Aided Design*, 2005.

[47] P. Maidee, C. Ababei, and K. Bazargan. Fast timing-driven partitioning-based placement for island style fpgas. In *Design Automation Conference*, pages 598–603, 2003.

[48] R. Marculescu, D. Marculescu, and Massoud Pedram. Efficient power estimation for highly correlated input streams. In *Design Automation Conference*, 1995.

[49] H.B. Min and E.S. Park. Graph-theoretic algorithm for finding maximal supergates in combinational logic circuits. In *IEE Proceedings-Circuits, Devices, Systems*, 1996.

[50] Alan Mishchenko, Satrajit Chatterjee, Robert Brayton, and Maciej Ciesielski. An integrated technology mapping environment. In *International Workshop on Logic Synthesis*, 2005.

[51] N. Miskov-Zhivanov and D. Marculescu. Mars-c: Modeling and reduction of soft errors in combinational circuits. In *Design Automation Conference*, 2006.

[52] H. Moghal, H. Qian, S. Sapatnekar, and K. Bazargan. Clustering based pruning for statistical criticality computation under process variations. In *International Conference on Computer-aided Design*, 2007.

[53] F. Najm. A survey of power estimation techniques in vlsi circuits. *IEEE. Transactions on VLSI systems*, 2:4:446–455, 1994.

[54] OpenAccess. Library core of EDA algorithms and infrastructure for openaccess. http://www.openedatools.org/projects/oagear.

[55] K. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE. Transactions on Computers*, C-24, 1975.

[56] K. K. W. Poon, S. J. E. Wilton, and A. Yan. A detailed power model for field-programmable gate arrays. *ACM Transactions on Design Automation of Electronic Systems*, 10(2):279–302, 2005.

[57] S. Raj, S. Vrudhala, and J.M. Wang. A methodology to improve timing yield in the presence of process variations. In *Design Automation Conference*, 2004.

[58] Y. Ran and M. Marek-Sadowska. Designing a via-configurable regular fabric. In *Custom Integrated Conference*, 2004.

[59] R. Rebonato and P. Jackel. The most genereral methodology to create a valid correlation matrix for risk management and pricing purposes. *Journal of Risk*, 2, 1999.

[60] Sachin Sapatnekar. *Timing*. Springer-Verlag, 2004.

[61] P. Sedcole and P. Y. K. Cheung. Within-die delay variability in 90nm FPGAs and beyond. In *IEEE International Conference on Field Programmable Technology*, 2006.

[62] P. Shivakumar, S.W. Keckler, D. Burger, M. Kistler, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *International Conference on Dependable Systems and Networks*, 2002.

[63] D.P. Singh and S.D. Brown. Constrained clock shifting for field programmable gate arrays. In *International Symposium on Field Programmable Gate Arrays*, 2002.

[64] S. Sivaswamy and K. Bazargan. Statistical generic and chip-specific skew assignment for improving timing yield of FPGAs. submitted to International Conference on Field Progammable Logic and Its Applications, 2007.

[65] S. Sivaswamy and K. Bazargan. Variation-aware routing for FPGAs. In *International Symposium on Field Programmable Gate Arrays*, 2007.

[66] Satish Sivaswamy and Kia Bazargan. Estimation and optimization of reliability of noisy digital circuits. In *International Symposium on Quality Electronic Design*, 2009.

[67] Satish Sivaswamy, Gang Wang, Cristinel Ababei, Kia Bazargan, Ryan Kastner, and Eli Bozorgzadeh. HARP: Hardwired routing pattern FPGAs. In *International Symposium on Field Programmable Gate Arrays*, pages 21–29, 2005.

[68] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature aware microarchitecture. In *IEEE International Symposium on Computer Architecture*, 2003.

[69] Jayanth Srinivasan, Sarita V. Adve, Pradip Vose, and Jude A. Rivers. The case for lifetime reliability-aware microprocessors. In *International Symposium on Computer Architecture*, 2004.

[70] K.Y. Tong, V. Kheterpal, V. Rovner, L. Pileggi, and H. Schmidt. Regular logic fabrics for a via patterned gate array (VPGA). In *IEEE Custom Integrated Circuits Conference*, 2003.

[71] J.L. Tsai, D.H. Baik, C.C.P. Chen, and K.K. Saluja. An yield improvement methodology using pre- and post-silicon statistical clock scheduling. In *International Conference on Computer Aided Design*, 2004.

[72] C. Visweswariah, K. Ravindran, K. Kalafala, S.G Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Design Automation Conference*, 2004.

[73] Lei Wang. An energy-efficient skew compensation technique for high-speed skew sensitive signaling. In *International Symposium on Circuits and Systems*, 2005.

[74] H. Y. Wong, L. Cheng, Y. Lin, and L. Hei. FPGA device and architecture evalutaion considering process variations. In *International Conference on Computer-Aided-Design*, 2005.

[75] Kun-Cheng Wu and Yu-Wen Tsai. Structured ASIC, evolution or revolution? In *International Symposium on Physical Design*, pages 103–106, 2004.

[76] S. Yang. Logic synthesis and optimization benchmarks, version 3.0. Technical report, Microelectronics Centre of North Carolina, 1991.

[77] Andy G. Ye and Jonathan Rose. Using bus-based connections to improve field-programmable gate array density for implementing datapath circuits. In *International Symposium on Field Programmable Gate Arrays*, pages 3–13, 2005.

[78] Chao-Yeng Yeh and Malgorzata Marek-Sadowska. Skew programmable clock design for FPGA and skew-aware placement. In *International Symposium on Field Programmable Gate Arrays*, 2005.

[79] B. Zahiri. Structured ASICs: Opportunities and challenges. In *International Conference on Computer Design*, pages 404–409, 2003.

[80] Y. Zhan and S. Sapatnekar. High efficiency green function based thermal simulation algorithms. *IEEE Tran. Computer-Aided Design of Integrated Circuits and Systems*, 26:1661–1675, 2007.

[81] C. Zhao, S. Dey, and X. Bai. Soft-spot analysis: targeting compound noise effects in nanometer circuits. In *IEEE Design and Test*, 2005.

[82] Quming Zhou and Kartik Mohanram. Transistor sizing for radiation hardening. In *International Reliability Physiscs Symposium*, 2004.

[83] P.S. Zuchowski, C.B. Reynolds, R.J. Grupp, S.G. Davis, B. Cremen, and B. Troxel. Hybrid asic and fpga architecture. In *International Conference on Computer Aided Design*, pages 187–194, 2002.