Physically-Adaptive Computing via Introspection and Self-Optimization in Reconfigurable Systems

by

Kenneth M. Zick

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2010

Doctoral Committee:

Professor John P. Hayes, Co-Chair
Professor John H. Holland, Co-Chair
Professor Todd M. Austin
Associate Professor Robert Dick

© Kenneth M. Zick
All rights reserved
2010

To Jie, Mom, and Dad

ACKNOWLEDGMENTS

I wish to sincerely thank all who enabled this research. Professor Holland, thank you for bestowing the ways of adaptive systems and for our collaboration with NASA; the sky is no limit now. Professor Hayes, thanks very much for the motivation, extensive feedback, important teachings and support. I am extremely grateful to NASA and its Graduate Student Research Program for four years of funding. I benefited greatly from U-M's Center for the Study of Complex Systems; its one-of-a-kind graduate program enabled me to set out on a new path. The Santa Fe Institute Complex Systems Summer School helped foster some important new connections and a swarm of Mars Tumble-weeds. I appreciate the many opportunities provided over the years by CSE/EECS, CoE, Rackham, and the University. Thanks to Xilinx and Sun Microsystems for donations, and thanks to my instructors and colleagues including my committee members Robert Dick and Todd Austin, the CSE staff, Eric Chaisson, Rick Riolo, Jan Gottlin, Marty Waszak, Sandra Myers, Carey Buttrill, Celeste Belcastro, Kevin Somervill, and NASA Langley. Working with Andrea Pellegrini, Joe Greathouse, and Aaron Bramson has been enlightening and fun. Best wishes as well to my classmates Daeyoung Lee, Chien-Chih Yu, Smita Krishnaswamy, Sungsoon Cho, Armin Alaghi, the participants in the Adaptive Hardware & Systems Reading Group, and the many others I've encountered.

For the important roles they have played, special thanks to my brother Matt, my brother Jeff, the Zicks, the Pitrones, my grandparents, Peter Riopelle, Paul Riopelle, Gary

O’Brien, Jim Davies, John Hajkus, Dean Robinette, Tim Gilhool, Tim Strong, Kevin Brandon, Corey Garrow, Kurt Severance, Mike Munichiello, and friends from MI, VT, TX, and MA. I’d like to acknowledge my family and, broadly, my ancestors; I wouldn’t be here if not for every one of them across the billions of years. The fortuitous chains of life are worth celebrating, and in my case culminate with my heroic parents. Thanks Mom and Dad. Lastly, I am profoundly grateful to my wonderful wife, Jie; I’m very fortunate that our two paths through the universe have merged happily into one.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Digital electronic systems typically must compute precise and deterministic results, but in principle have flexibility in *how* they compute. Despite the potential flexibility, the overriding paradigm for more than 50 years has been based on fixed, non-adaptive integrated circuits. This one-size-fits-all approach is rapidly losing effectiveness now that technology is advancing into the nanoscale. Physical variation and uncertainty in component behavior are emerging as fundamental constraints and leading to increasingly suboptimal fault rates, power consumption, chip costs, and lifetimes. This dissertation proposes methods of physically-adaptive computing (PAC), in which reconfigurable electronic systems sense and learn their own physical parameters and adapt with fine granularity in the field, leading to higher reliability and efficiency.

We formulate the PAC problem and provide a conceptual framework built around two major themes: introspection and self-optimization. We investigate how systems can efficiently acquire useful information about their physical state and related parameters, and how systems can feasibly re-implement their designs on-the-fly using the information learned. We study the role not only of self-adaptation—where the above two tasks are performed by an adaptive system itself—but also of assisted adaptation using a remote server or peer.

We introduce low-cost methods for sensing regional variations in a system, including a flexible, ultra-compact sensor that can be embedded in an application and implemented on field-programmable gate arrays (FPGAs). An array of such sensors, with only 1% to-

tal overhead, can be employed to gain useful information about circuit delays, voltage noise, and even leakage variations. We present complementary methods of regional self-optimization, such as finding a design alternative that best fits a given system region.

We propose a novel approach to characterizing local, uncorrelated variations. Through in-system emulation of noise, previously hidden variations in transient fault susceptibility are uncovered. Correspondingly, we demonstrate practical methods of self-optimization, such as local re-placement, informed by the introspection data.

Forms of physically-adaptive computing are strongly needed in areas such as communications infrastructure, data centers, and space systems. This dissertation contributes practical methods for improving PAC costs and benefits, and promotes a vision of resourceful, dependable digital systems at unimaginably-fine physical scales.

CHAPTER 1

Introduction

Digital systems have advanced remarkably for the past half-century. Semiconductor chips have been manufactured with increasing numbers of components exhibiting near-uniform, predictable behavior. This has allowed system architectures and programs to be designed from the top-down using a "one size fits all" strategy, independent of specific hardware characteristics or lifetime changes. This separation of the logical and physical realms has been an essential aspect of the computer revolution, allowing a single design to be incorporated into millions of systems. However, computation is physical, and as technology advances to nanoscale feature sizes, this long-standing paradigm is under threat. It is time to reconsider the complete separation between "bits and atoms."

One emerging and fundamental problem is physical variation. The spread in physical parameters such as transistor threshold voltage is growing dramatically, leading to chips with unique characteristics. An example is illustrated in Figure 1.1 [17]. Technology forecasts indicate rapidly worsening variability over the next several years, as seen in Table 1.1. The threat worsens even further at the lower supply voltages needed for low-power operation [28]. Another form of variation, wearout that occurs over time, is increasing, as are transient fault rates as illustrated in Figure 1.2 [16]. These trends are adding to the enormous costs required to develop viable manufacturing technologies.

Upon manufacturing, chips are being discarded due to marginal regions and components that fail to meet specifications, increasing costs further. Moreover, the systems built using the remaining chips are increasingly suboptimal.

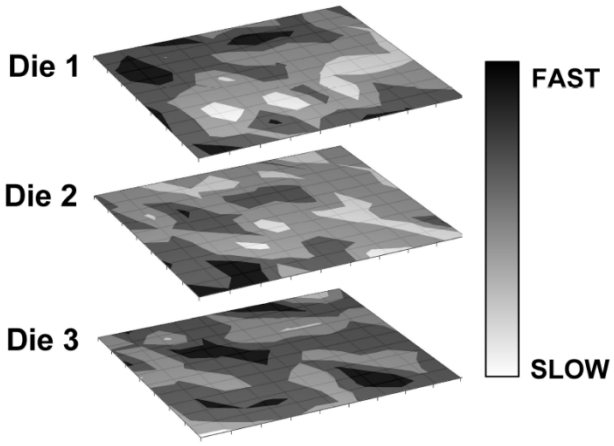

Figure 1.1: Delay profiles for three different 65 nm dies [28]

Table 1.1: Semiconductor technology outlook [17]

Date of high volume manufacturing	2006	2008	2010	2012	2014	2016	2018
Technology node (nm)	65	45	32	22	16	11	8
Billions of transistors	4	8	16	32	64	128	256
Variability	Medium		High		Very High		

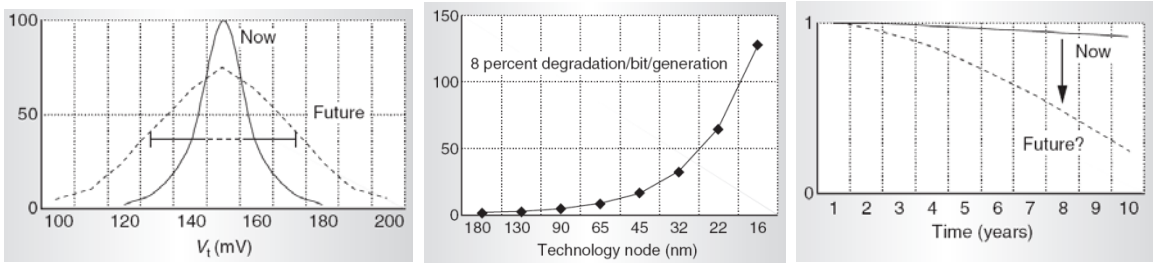

Figure 1.2: Threats from variability (left, threshold voltage), transient faults (middle), and wearout (right, degrading transistor current) [16]

Along with uncontrolled variation there is a second fundamental problem: growing uncertainty in physical parameters and component behavior. At fine scales it is harder to accurately model, predict, or observe subtle and complex physical interactions. During the design phase, the manufacturing process is often poorly understood and variation models are highly questionable. After years of metrology and process learning, the manufacturing models may improve, but often this is far too late to help designers [104]. Pre-manufacturing models and simulations can show considerable disagreement. One study of thermal gradients across a die suggests a difference of 8.6°C [66], while another sets the number at an alarming 20°C [102]. Even traditional models such as the heavily-used "bathtub" curve are becoming less effective; a large-scale study of memory reliability in Google data centers found that error rates rose unexpectedly after just 10–20 months of operation [89]. Increased integration has led to a widening knowledge gap. Thorough characterization of individual components and regions is costly. When there are over one billion transistors on a chip, the number of outliers is no longer trivial, but rather can number in the tens of millions.

These trends are impacting a wide range of semiconductor applications, especially those needing high levels of efficiency or reliability, which is to say, most applications. One domain is on-board processing in spacecraft [31][41], unmanned aerial vehicles (UAVs), and similar embedded systems. These applications often require low power, reconfigurability, well-understood soft error characteristics, and long lifetimes. Another domain is communications; routers from vendors such as Cisco often require high reliability since soft errors can have unbounded consequences [94]. High-performance computing (HPC) is another domain being impacted. As with embedded systems, there are

constraints on power and temperature. Additionally, the scale of HPC computations is too large to allow for extensive redundancy as protection against soft errors, and there is a large cost associated with frequent checkpointing, so new methods for characterizing and avoiding soft errors are needed [72]. Lastly, these trends are important not only for today's applications using CMOS-based semiconductors, but also for future applications based on emerging reconfigurable nanotechnologies [38].

Most computing systems are non-adaptive; the architectures employed and programs executed are fixed. This differs greatly from naturally occurring complex systems which are often highly adaptive. There are timely opportunities for using some principles, theory, and lessons from natural adaptive systems. At the same time, we should not expect to simply mimic nature, since computing challenges often differ in character from those faced by natural systems. Consider that computing systems are defined not only by *what* they can compute, but also *how* they compute. There is an important distinction between the former, which involves mathematical functions and abstractions such as instruction set architectures, and the latter involving physical properties such as noise susceptibility, power consumption, hot spots, and component lifetimes. Whereas the functional level of abstraction is often not amenable to adaptive approaches, there are enormous opportunities for adaptation at the underlying levels.

In this dissertation, we consider a paradigm we define as *physically-adaptive computing* (PAC). With PAC, systems learn about their physical circumstances and perform fine-grained self-adaptation in the field. In doing so, they improve their ability to meet their application-specific, physically-focused objectives, or in other words, they improve their fitness. Work addressing certain aspects of physically-adaptive computing has been

underway for many years, but significant progress has been lacking for a variety of reasons: the absence of significant physical variation in earlier semiconductor technologies, the lack of fine-grained reconfigurability, and the lack of design tool support. All of these reasons are rapidly disappearing, suggesting a prime opportunity for deploying PAC. Currently, one of the critical questions is how systems can adequately characterize fine-grained physical parameters, with little overhead. We contribute several novel, low-cost methods of sensing and self-test. Hardware experiments provide some surprising findings and previously unpublished data. A further question is how systems can feasibly optimize themselves using the information learned. We study a range of optimization mechanisms in two different physical contexts, and conduct case studies which quantify the possibilities. The contributions help to increase the benefits and decrease the costs of physically-adaptive computing, laying the groundwork for a significantly more adaptive, efficient, and reliable paradigm for digital electronic systems.

1.1 General Background and Problem Definition

Here we define the relevant elements of a computing system, including the physical substrate used for computation, the set of computational functions, and the set of system parameters. We then describe the key concepts of variations and adaptation, and define the PAC research problems that we address.

Substrate. We define a *substrate* as physical hardware able to support computation, such as an individual microchip or portion of a microchip. We use the notation Sub_j to represent a particular substrate. Substrates have an associated *platform type*, e.g., a chip model type. Each substrate, even among those of the same platform type, is physically unique. Certain computing platforms allow for fine-grained reconfiguration of the circuit

components and interconnections. Prime examples include field-programmable gate arrays (FPGAs); detailed background on FPGA-based systems is given in Section 1.2. A *configuration* specifies how to configure and connect the physical elements on a substrate. A particular configuration is denoted $Config_i$. Aside from FPGAs, there is a pronounced trend in microprocessor chips toward higher numbers of processor cores; these many-core platforms provide a form of reconfigurability but are relatively coarse-grained. A third type of reconfigurable platform uses nanoelectronics formed by self-organization rather than traditional lithography. An example is the nanoPLA, though the technology is still in the research stage [38]. Note that we refer to all of these as reconfigurable platforms, and we refer to a specific physical instance as a reconfigurable substrate.

Computational functions. A function that is to be computed will be represented by *function F*. It can be expressed mathematically in various ways such as with an algorithm or a finite state machine. Representations that are close to the hardware include register-transfer level (RTL) models and behavioral models using a hardware description language (HDL). We define a *function netlist FN* as a logical realization of *F* that lists the functional components and their connections. In many hardware-oriented design flows, this is a netlist generated by a synthesis tool. The function netlist in turn is implemented as a substrate configuration. There can be many functionally-equivalent configurations that implement *FN*. Re-implementation is a process of generating a new configuration that implements *FN*. Re-implementation can be performed on the original netlist *FN* or via direct modification of a configuration denoted $Config_1 \rightarrow Config_2$. Computations of a function *F* are performed by a *configuration-substrate pair* ($Config_i$, Sub_j). This pair is

an important entity that will be a focus of the adaptive process. Figure 1.3 illustrates a basic, traditional flow that occurs in the design and implementation of computing systems. Although such a flow is sometimes associated with FPGA-based "hardware" development, the configuration can be considered a type of low-level software (sometimes called firmware). In fact, the flow is general enough that it could be readily modified for "software"-oriented systems, e.g., when computational functions are expressed as instruction-based programs.

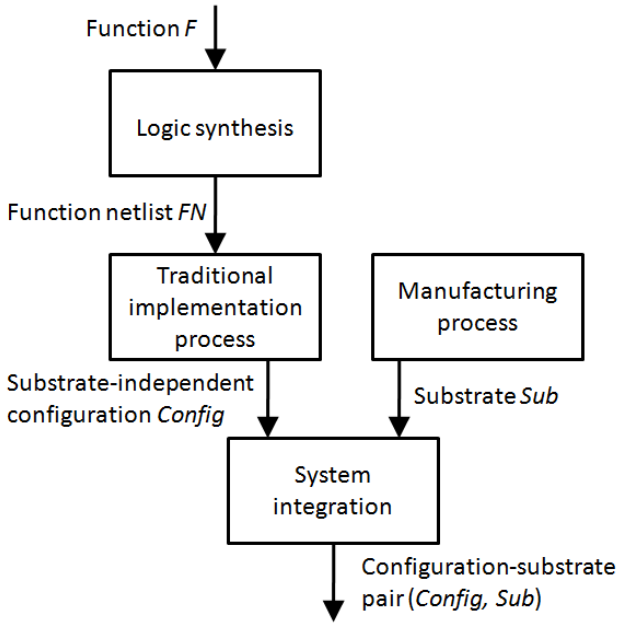

Figure 1.3: Conceptual view of a conventional system design flow

Parameters. A configuration-substrate pair has an associated set of physical and abstract *parameters* $\{Par_1, Par_2 \ldots\}$. We separate the parameters into three classes containing those specific to 1) the substrate, 2) the implemented logic, and 3) system operation. The main impetus for PAC is the deep uncertainty at design time of *substrate parameters* such as the inherent performance and reliability of individual transistors and wires. If there were no such uncertainty, then the traditional substrate-oblivious approach to com-

puter engineering would be sufficient. However, uncertainty is growing and is a consequence of technology advancing towards the atomic scale. The sources of variation in substrate parameters include imperfect lithography in manufacturing, fluctuations in the number of dopant atoms or gate oxide atoms, uneven stress during operation, and many others. *Logical parameters* are those primarily associated with the computational logic and are independent of the substrate. Examples include signal probabilities (i.e., how often a signal is in the 1 state), toggle rates, and error propagation probabilities. Logical parameters can exhibit high levels of variation, such as logical fanouts ranging from one to more than 30 [38], and order of magnitude differences in error propagation probabilities [130]. *Operational parameters* are those associated with the operation of a computing system. These emerge from the act of computing and depend upon the interaction of logical, substrate, and external environmental factors. Examples include power consumption, temperature, and fault/error rates. Operational parameter variations such as thermal hot spots increase the need for physical adaptation. Certain operational parameters are also important for assessing the overall fitness of a system.

Parameter variations. Many of the key parameters in a computing system exhibit some form of variation. There are spatial variations in substrate parameters such as delay, and in operating conditions such as temperature. In addition, parameters can exhibit temporal variations caused by wearout or changes in activity. Wearout mechanisms are manifold and becoming harder to ignore [99], such as negative bias temperature instability (NBTI), hot carrier injection, electromigration, and total ion dose (TID). Due to variations, only a certain number of chips in a group meet the desired parameter specifications; this is the *parametric yield*.

Regional variations are those that are spread over a physical area, for instance when all transistors in a region of a chip have low threshold voltages (V_t) due to imperfect manufacturing, or when a high-temperature region (hot spot) emerges due to uneven power consumption. These are also sometimes described as spatially-correlated or systematic variation. The *correlation distance* specifies the size of the region, e.g. a radius of 1 mm [24], for which parameters have a non-negligible correlation. *Inter-die* or *die-to-die* variation affects an entire die globally. It is sometimes considered a separate phenomenon from regional variation, but can also be thought of as a special case in which the correlation distance is much greater than the size of a die.

The second main type of spatial variation is local, spatially-uncorrelated variation, which we abbreviate as *local variation*. This occurs at a single component (transistor, wire, etc.), and has no correlation to parameter values at other locations. It is also sometimes called random variation. With nanoscale technologies, component properties can be dramatically affected by a small number of atoms. In the ubiquitous MOSFET-type transistors, electrical properties depend on the number of dopant atoms implanted during manufacturing. The number of atoms cannot be controlled precisely, leading to random dopant fluctuations that affect V_t and other parameters. Similar sources of local variation include uneven gate oxide thickness and line edge roughness. Local wearout is possible via electromigration, charge carriers becoming trapped in the gate oxide, atoms being displaced by radiation, and many other sources.

The amount of variation in a parameter is often modeled as being a sum of independent random variables representing the die-to-die, regional, and local variations:

$$\Delta Par = \Delta Par_{D2D} + \Delta Par_{reg} + \Delta Par_{local}. \quad (1.1)$$

Figure 1.4 illustrates both regional variations (e.g., the general dependence of frequency on the row location) and local variations (the spiky fluctuations), and how they combine in a single parameter.

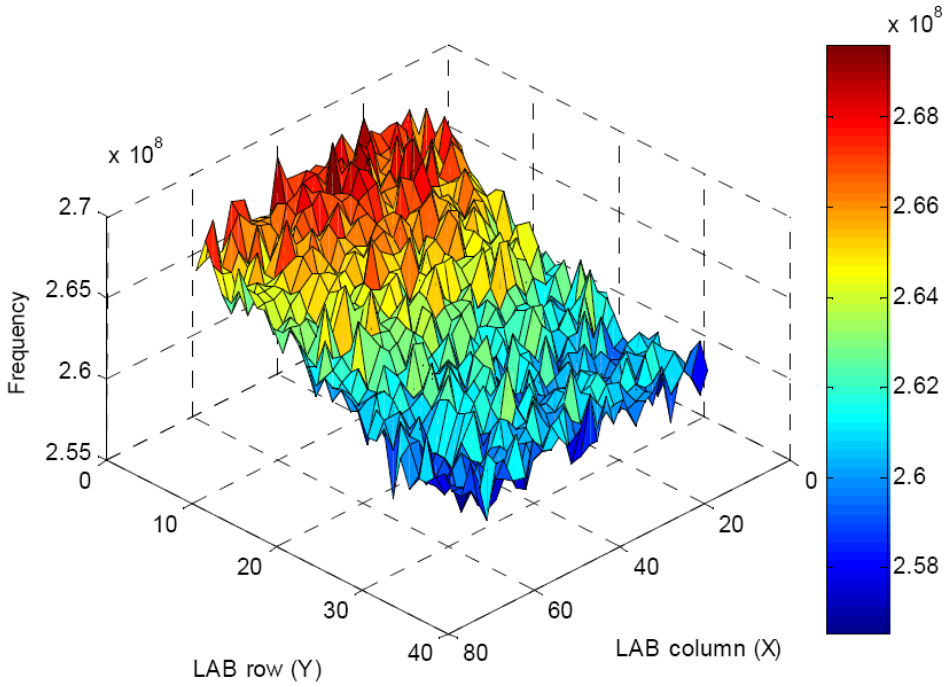

Figure 1.4: Regional and local variations in ring oscillator frequencies across an FPGA [90]

Transient Faults and Soft Errors. Advanced digital electronics are prone not only to physical variations but also to transient faults and soft errors. In this context, a *transient fault* is a temporary effect triggered by noise. The fault can lead to a deviation in the state trajectory of the machine; this is known as a *soft error* because it involves an error in information but no hard/permanent damage. A soft error may in turn cause an incorrect result to be returned after a computation, a scenario sometimes called a *system-level soft error* or a *failure*. One common source of transient faults is ionizing radiation, which can take the form of alpha particles, neutrons, solar protons, heavy ions, etc. A

transient current pulse is often called a *single event transient* (SET). A transient fault that changes the value of a state element is known as an *upset* or a bit-flip, and an upset of a single storage cell is often referred to as a *single event upset* (SEU).

Adaptation. Adaptation is a process by which a system undergoes structural or parametric changes and becomes better fitted to an environment over time. Adaptive systems are commonly found in nature, ranging from the simplest biological species to complex ecosystems, and including human-related systems such as language, economies, and society. The system environment itself is often dynamic, meaning adaptation is ongoing and the system never settles to an equilibrium.

The general mechanisms for adaptation can include evolution, learning, self-organization, or some combination thereof. Evolution involves key concepts such as natural selection, descent with modification, genetic variations via operators (cross-over, mutation, etc.), and differential reproduction. Learning on the other hand is usually considered a process occurring in a single individual through interaction with an environment over time. Self-organization is the notion of patterns emerging via the interactions of many agents; an example is the cell differentiation that occurs during biological development. Of course, nature is complex and there are many layers, variants, and hybrids of these phenomena.

An open question is the nature of adaptation in engineered/artificial systems [45]. To what extent can such systems adapt autonomously and in a productive manner? A few words need to be said about adaptation in artificial systems. As alluded to earlier, digital electronic systems in particular are defined not only by *what* they can compute, but also *how* they compute. There is an important distinction between the former, which

involves mathematical functions and abstractions such as instruction set architectures, and the latter involving *nonfunctional* properties. Much work in adaptive computing systems deals with the former. Examples of functional adaptation include adaptive decision making in robotic controllers, adaptive co-processing [47], some types of adaptive signal processing, and attempts at evolving digital functions in software or hardware. Enabling functional adaptation in computing systems is often a daunting challenge because much of the existing computing environment is very brittle and unforgiving of novelty. In this dissertation, we consider nonfunctional adaptation, and more specifically, adaptation that involves the physical properties of a system. A partial taxonomy of computing system properties is shown in Figure 1.5. A system has a set of functions $\{F_1, F_2, F_3 \ldots\}$ that it can compute, and also has nonfunctional properties that tend to involve physical phenomena such as power consumption, the area used by a circuit, the reliability of computations performed, and the lifetime reliability of the physical components. Even properties such as throughput and real-time performance involve a physical resource (time).

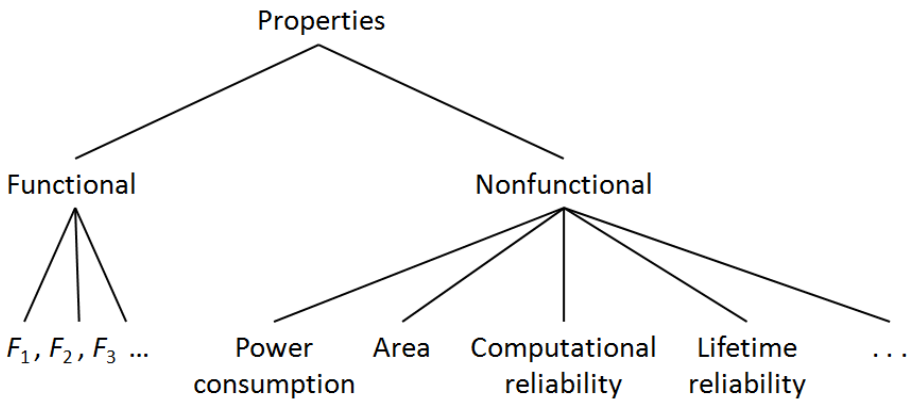

Figure 1.5: Taxonomy of computing properties

Note that adaptation is different than dynamic feedback control. With the latter, a controller generally accepts feedback and modifies a control variable, often to bring the system to a known state. An example in digital systems is dynamic control of a system's voltage and frequency based on temperature feedback. With adaptation, the fitness landscape is often uncertain, and a system generally finds new states with higher fitness through learning or evolution, carrying information forward via structural or parametric changes. The resources required to find better solutions can be extensive. In some cases the two concepts overlap, as in the fields of adaptive control and intelligent control.

Problem Definition. The main problem being addressed in this dissertation is how to foster the fittest digital systems, given spatially unique, time-variant, uncertain physical circumstances. Two of the critical research questions are: How can systems efficiently learn about their physical state and related parameters? How can systems feasibly optimize themselves and re-implement their designs on-the-fly using the information learned, given very limited system resources?

1.2 FPGA-based Systems

Field-programmable gate arrays (FPGAs) are currently the most widespread reconfigurable platform and are discussed often in this dissertation. Therefore, we provide some background information on FPGA-based systems including the hardware, reconfigurability, and design flows.

1.2.1 Hardware Platforms

FPGAs are highly-flexible semiconductor platforms that allow many useful algorithms to be implemented efficiently; thus they are popular for signal processing, communications, and a widening range of applications. A generic FPGA platform is illu-

strated in Figure 1.6, with a more detailed view of reconfigurable logic and interconnect in Figure 1.7. FPGAs contain vast arrays of reconfigurable components that can be used to implement logic functions. The logic is interspersed with networks of reconfigurable interconnect, on-chip memories, and I/O blocks. The basic unit of logic is a *lookup table* (LUT) which can implement any k-input function; typically the k inputs select among 2^k values stored in SRAM cells. For instance, a 6-input LUT contains 64 SRAM cells. Each LUT is paired with a flip-flop to form a *LUT-FF pair*. Logic is divided into small homogeneous *clusters* containing several LUT-FF pairs. Clusters are sometimes called slices in Xilinx terminology or logic array blocks (LABs) in Altera terminology. The amount of circuitry can be enormous, approaching one million LUT-FF pairs per chip.

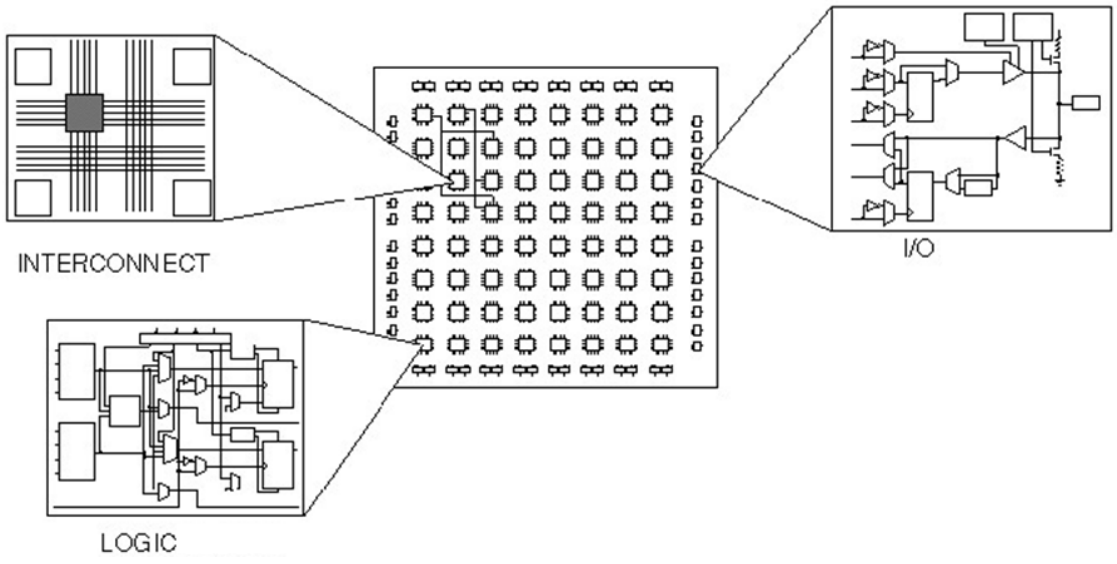

Figure 1.6: High-level diagram of an FPGA [76]

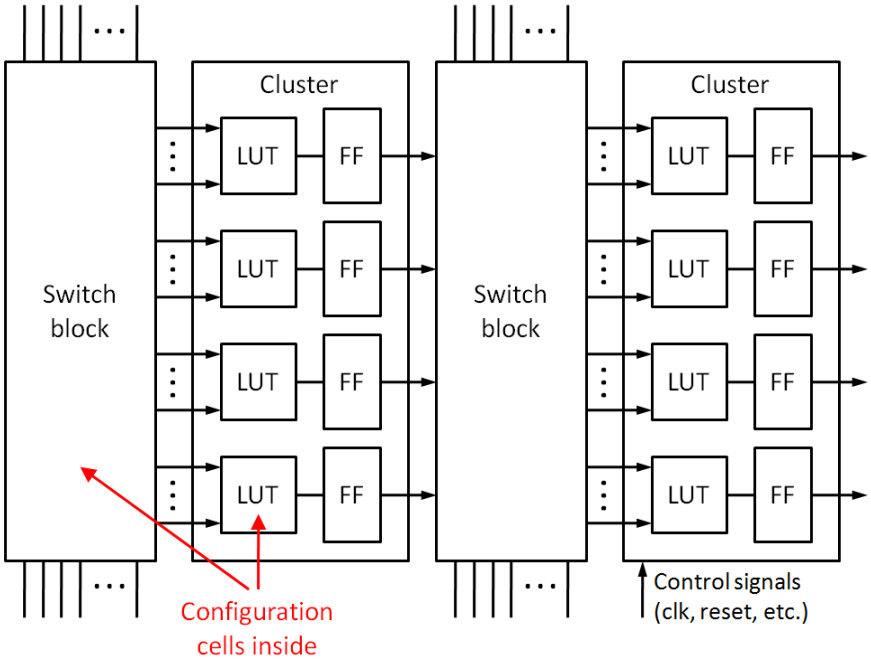

Figure 1.7: Diagram of FPGA reconfigurable logic and interconnect

An example of an FPGA type is the Xilinx Virtex-5, which is a fifth-generation, high-performance family. A specific model in this family is the Virtex-5 110T. All Virtex-5 FPGAs have clusters (slices) containing four LUT-FF pairs. Two clusters form what is called a *configurable logic block* (CLB). The interconnect consists of wires and programmable switches inside switch boxes and local connection boxes; many of the routing details are proprietary. In addition to the elements mentioned, the chips include hard macros such as DSP blocks, high-speed network interfaces, and sometimes entire processor cores.

1.2.2 Reconfigurability

FPGAs can be reconfigured by writing a new pattern into the on-chip configuration memory. Most FPGAs such as the Virtex-5 are SRAM-based, meaning the configuration bits are stored in volatile, on-chip static random access memory (SRAM). A configuration is streamed into the chip from an external nonvolatile memory whenever a chip is

powered up; for this reason a configuration is also sometimes referred to as a *bitstream*. Certain FPGAs include nonvolatile configuration memory (e.g., flash memory) right on the chip. A third type of platform uses anti-fuses to represent the configuration, but these platforms are only one-time programmable and of little interest for adaptive computing.

FPGA configurations can contain tens of millions of bits and are streamed into chips at bandwidths on the order of gigabits per second; thus the configuration time can be on the order of milliseconds. This delay has been a persistent roadblock to functionally-adaptive computing intended to operate at the time scale of individual computations and sub-computations. Support for *partial* reconfiguration is improving, creating new opportunities. In any case, low-overhead adaptation at coarser time scales—such as hours, minutes, or even seconds—can already be achieved, which bodes well for physical adaptation.

1.2.3 Design Flow

Figure 1.8 depicts a conventional FPGA design flow. The user provides an application description, such as a function *F* described in HDL, along with application-specific constraints for the various tools. The function is synthesized to a technology-independent netlist *FN* using an electronic design automation tool. In newer "high-level synthesis" flows, the function is described using a C-like language and synthesized with a separate tool. In any case, the remaining steps are normally performed by FPGA vendor tools. The netlist is mapped into the primitives of a targeted FPGA platform. After this "technology mapping," a placement tool assigns all elements to a physical location. The mapping and placement tools are also responsible for "packing" the logic into cluster-size groups [2]. After placement, a routing tool makes all the necessary connections using the

reconfigurable interconnect. Lastly, a configuration (i.e., bitstream) is generated and stored for eventual download to an FPGA chip of the appropriate type.

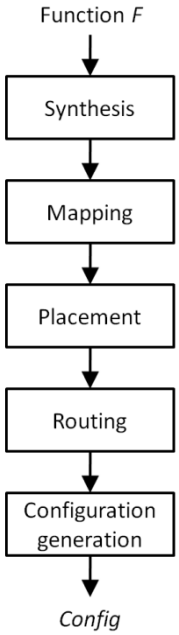

Figure 1.8: Example of a conventional FPGA design flow

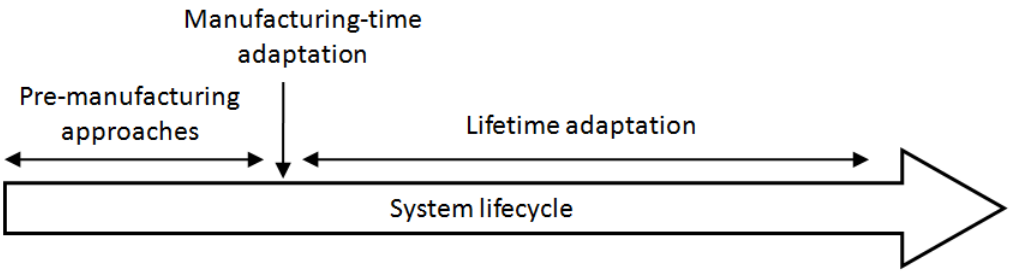

Figure 1.9: Addressing physical variations and uncertainty at different times in a system lifecycle

1.3 Related Work

The concept of physically-adaptive computing has connections to research in on-line adaptation of integrated circuits, reconfigurable computing, the general study of adaptation, and many others. We classify the previous work into three broad classes: pre-

manufacturing approaches, manufacturing-time adaptation, and lifetime adaptation (Figure 1.9).

1.3.1 Pre-Manufacturing Approaches

As a stopgap measure, there have been efforts to address variability in a nonadaptive way at pre-manufacturing design time. For instance, extra safety margin can be added to a design. The amount of margin required is estimated, based in part on a general model of the expected variability. A second approach involves timing analysis with statistical models of delay rather than worst-case models. The objective is to design a circuit that can be manufactured with higher yield at a given frequency, using the facts that delays are usually better than worst case and that local, normally-distributed variations may cancel out over many components. Two examples of statistical static timing analysis of FPGA applications are given in [91] and [59]. Such efforts are sometimes called "variation-aware design," though in actuality they are only aware that variability will be present and not aware of what the specific variations will be in a manufactured chip. They are better described as variation-tolerant.

Another approach that occurs before chip manufacturing is work to improve the manufacturing process itself. Advanced semiconductor processes are developed over a period of years and require extensive modeling, equipment development, test chips, and new fabrication plants. It is well known that the investments required are enormous and growing. Sizable efforts are needed to minimize parameter variation and uncertainty, since the traditional, non-adaptive paradigm depends on manufacturing consistent and well-behaved components. Part of the rationale for PAC is reducing these costs and making better use of inherently variable substrates.

Pre-manufacturing approaches are inherently oblivious to the unique characteristics of physical substrates. Moreover, they do not address changes that occur in the field to workloads, external environments, or substrate parameters. Therefore we focus on physically-aware approaches, both at manufacturing time and in the field.

1.3.2 Manufacturing-Time Adaptation

Some compensation for physical effects is possible at manufacturing time. At-speed testing of dies on a wafer allows worst-case timing to be determined and chips to be placed into a speed bin or discarded altogether. Testing is also performed for defects. Defects are permanent hardware faults, and can be thought of as an extreme type of highly localized variation. Faulty logic or memory can sometimes be permanently swapped out for spares using fuses [46]. With FPGAs, application-independent methods are pessimistic since many faulty components and worst-case paths would not actually be activated by an application. Moreover, screening for marginalities will not work if *every* manufactured chip has marginalities. The ITRS roadmap warns that the gray area between good and bad will expand, with effects such as "non-catastrophic gate oxide breakdown or highly resistive vias" [52].

A more customized approach is to test chips based on application-specific requirements. The prime example is Xilinx's EasyPath service [115]. A customer submits up to two configurations needed by their application. The configurations are fed into a Xilinx internal tool which generates the appropriate chip tests. For instance, an application typically uses only 1%–10% of a chip's configuration cells, and thus those are the only ones tested. Chip yield greatly improves because many chips with don't-care defects can still be shipped. Since the manufacturing costs can be amortized over more good dies, chip costs are reduced by up to 75%. With the Virtex-6 and entire "7 Series" of FPGAs, Xi-

linx guarantees a cost reduction of at least 35% [115][116]. Application-specific screening does not enable the full benefits of customizing an application to an individual die. In fact, adapting in the field is problematic since faults and marginalities may be lurking in the untested resources.

The next level of manufacturing-time customization is to actually create a map of the variations on each chip and to generate a customized implementation of the application. A proposal for this is presented in [24], using variation-aware place and route and FPGAs. However, the costs are high to use expensive testers for fine-grained characterization, to perform resource-intensive computer-aided design cycles, and to handle large numbers of unique configurations.

A further limitation of all of these manufacturing-time approaches, as with pre-manufacturing approaches, is that they do not address parameter shifts that occur in the field.

1.3.3 Lifetime Adaptation

The research most closely aligned with this dissertation involves lifetime adaptation in digital systems. Historically, a driver for such approaches was the threat of hard faults appearing in the field and the need for fault tolerance. There have been proposals for handling FPGA defects via self-test and re-routing using a lightweight routing tool that runs in the system itself [62][30][105][83]. These ideas provide a starting point but leave unaddressed important issues such as correlated variations, the role of computational activity (e.g., affecting thermal hotspots or data-dependent wearout), and how to make use of functioning but marginal elements.

One proposed strategy for adapting to unique reconfigurable substrates is to generate a large number of implementations at random and to allow each system to identify the

most suitable alternative through trial and error [48][67]. While sufficient for avoiding a small number of defects, this approach does not work well for large systems subject to variations on multiple scales. A randomly generated configuration is unlikely to provide a close fit.

A little preliminary research has addressed the combination of self-characterization of variations, and subsequent optimization of configurations. Some of these studies have focused specifically on the problem of delay variations, in FPGAs [54] or in a non-CMOS reconfigurable technology [38]. Several works have focused just on self-characterization of variations, for instance involving circuit delay [112]. Another set of studies has considered variation-aware optimization techniques using signal polarity inversion [129], technology mapping [92], re-placement [24], and logical-physical mapping in a memory array [79]. A relevant theoretical study of variation-aware optimization is given in [91]. Note that these optimization techniques all depend on accurate characterization data being available.

Some early progress has been made regarding adaptation in integrated circuits [103] and microprocessors [8]. A method of measuring leakage variations on a per-core basis has been proposed [127]. Many of the methods for many-core processors, such as the use of analog sensors and instruction set-based error detectors, are not generally applicable to reconfigurable platforms such as FPGAs, but the goals of introspection and resiliency are consistent with the theme considered here.

Whereas the focus of this dissertation is physical adaptation, related research has been carried out in functional adaptation and adaptive computing. Some of the problems addressed include dynamic switching between previously-generated FPGA configura-

tions implementing different algorithms, finding effective adaptive control policies, and the design of application-specific instruction sets. Some works in functional adaptation share with PAC a need for in-system implementation of computations [107] or on-board decision making. Examples of cognitive architectures for adaptive computing systems include a Bayesian network-based architecture [33], a genetic algorithm-based classifier system [10], and a reinforcement learning-based controller [86]. Autonomic computing is another related area, involving the study of self-management and other so-called self-* properties in computing systems [29]. Much of the work is conducted in the logical realm [88][68].

Some related work involves coarse-grained feedback control of physical parameters. Examples of control schemes for computer systems include chip-wide dynamic voltage/frequency scaling [51][10] and thermal/power management in data centers [49]. These architectures tend to be dynamic but not adaptive. Methods of voltage/frequency scaling for many-core processors are being developed, where each voltage/frequency domain contains multiple cores [28].

A recent related research topic is cyber-physical systems, where computer and communication technologies are used to control an external physical system. Since PAC is focused on physical phenomena in the computer system itself, it is largely complementary to that work. The autonomous adaptation aspect of PAC suggests connections to the broader fields of engineered adaptive systems and intelligent systems. For instance, a study of self-modeling robotic systems deals with the analogous challenge of adapting to a substrate prone to wearout/damage [15].

In summary, PAC research builds upon and has ties to several other research areas, but at the same time poses some distinct and important research questions of its own.

1.4 Proposed PAC Framework

In this sub-section, our approach and general architecture for physically-adaptive computing is proposed. Detailed methods and results follow in the subsequent chapters. Portions of this sub-section were published in [134].

1.4.1 PAC Definition

We define a *physically-adaptive computing (PAC)* system as one in which system configurations are optimized to suit the physical landscape, leading to higher fitness. The general approach proposed here is based on periodic, fine-grained introspection and self-optimization controlled by an adaptation agent.

1.4.2 Fitness

In the context of digital electronic systems, *fitness* is a measure of how well a system is meeting its application-specific objectives. It is sometimes more convenient to use the concept of minimizing cost rather than maximizing fitness. We use the terms fitness and cost interchangeably, with the understanding that lower cost corresponds to higher fitness.

Cost functions are used to define the cost for a given solution among the immense space of possible solutions. The cost can be a function of parameters associated with the solution. For example, the FPGA vendor Xilinx uses the following linear weighted cost function to evaluate different implementations of an application [122]:

$$Cost(Config) = w_1 \times (wire\ length) + w_2 \times (timing) + w_3 \times (power) \tag{1.2}$$

Cost functions are commonly used in computer-aided design but traditionally do not account for the unique characteristics of the substrate. We propose a new approach in which cost is instead defined for a (*Config*, *Sub*) pair:

$$Cost(Config, Sub) = f(Par_1, Par_2, Par_3 \ldots) \tag{1.3}$$

1.4.3 Introspection

In the context of PAC, introspection is the process of a system learning about its own parameter values and the phenomena affecting its fitness. This generally involves obtaining data, using the data to estimate the related parameters, and sometimes building internal models. One method of obtaining data is internal sensing. This is appropriate for regional variations since the parameter value at the sensor location is correlated to the surrounding region. A second method is self-test of component parameters; this addresses local variations that occur at individual components.

In the domains of intelligent control and more specifically autonomic computing, sensor feedback data tends to be always available. This is considered automatic sensing. Introspection in PAC goes further in that there are more extensive requirements for *active sensing*. An adaptation agent may need to decide which of many parameters to characterize and then take actions to obtain data, for instance by initiating an appropriate self-test procedure.

System configurations, as encoded in bitstreams, act as models of the computing system that can be used to reason about logical structure, physical placement, etc. An agent can also build a model of the physical substrate, by collecting sensor data and estimating the physical parameters. In the case of spatially-correlated variations, the agent can acquire data samples and build a model of a parameter profile across the entire substrate.

For instance, a model could be built that has a quadratic dependence on the *x* and *y* positions on the die: $Par(x,y) = ax^2 + by^2 + cx + dy + e$. Building and maintaining detailed models can be demanding [44]. It has been claimed that the best model of the world is the world itself. In this sense, the agent can sometimes collect local physical data and immediately perform local self-optimization, without the need for building and maintaining a full variation map.

1.4.4 Self-Optimization

Self-optimization in the context of PAC is the process of a system finding new configurations with higher fitness, given a physical substrate and environment (Figure 1.10). This includes generating and validating the configurations. Generally, the generation of alternative configurations cannot be done at random but rather needs to be informed by the fine-grained introspection mentioned above. The opportunities for optimization in reconfigurable systems are many, including local logic swapping, adaptive body bias, selection of new netlist-level designs, LUT input re-ordering, inverted encodings, re-synthesis, re-mapping, re-packing, re-placement, and re-routing.

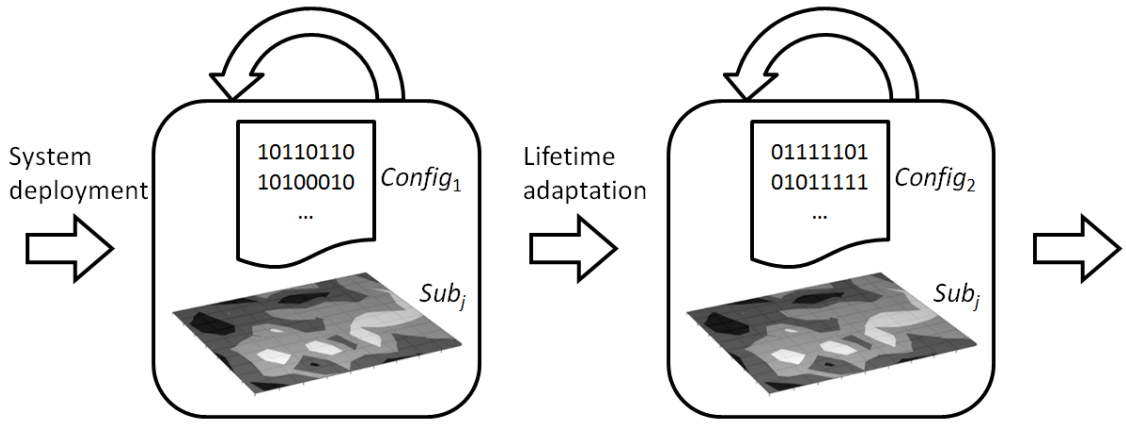

Figure 1.10: High-level view of the PAC concept

1.4.5 Time Scale

The intent with PAC is for self-adaptation to occur periodically as needed. Upon first being deployed, a system can direct a broad set of physical adaptations to account for manufacturing variations. For example, optimizing the circuit placement or body bias may enable a slight reduction in supply voltage, thereby decreasing dynamic and static power consumption. Following this initial period, a system can continue to adapt periodically to match the time scale of significant shifts in parameters. The adaptation loop can require tens of milliseconds for reconfiguration alone, so certain high-frequency dynamic behavior cannot be addressed by PAC in real-time. Dynamic voltage scaling can occur within one hundred microseconds [51]. Similarly, unintended dips in the supply voltage can occur on the order of microseconds. However, many important phenomena can have time scales that are much longer. Temperatures and workloads can sometimes change significantly over seconds and minutes. Solar events can increase the radiation flux by four orders of magnitude for periods of hours or days [35]. Aging effects may occur over a period of weeks. Moreover, even high-frequency phenomena can sometimes be addressed indirectly. For instance, while an adaptation loop is typically not quick enough to respond to a specific voltage transient, it can be used to mitigate systematic problems detected, such as recurring voltage transients in a specific location.

1.4.6 Fine Granularity and FPGAs

We define physical phenomena as fine grained if they have a length scale or time scale much smaller than that of an entire computation. This includes most intra-die variations such as various “hot spots” and random local variations. Correspondingly, methods of introspection and self-optimization should be considered fine grained if they have a resolution similar to the phenomena of interest. Examples include the ability to sense

spatial variations at scales much finer than the correlation distance, the ability to perform self-test on small blocks such as individual latches, and the ability to re-arrange logic functions or re-route around individual wires or switches.

Approaches with only chip-level spatial granularity are coarse by comparison. Examples include chip-wide voltage or frequency scaling, and chip-wide adaptive body bias. A middle ground is granularity at the processor core level. As many-core processors become common there is growing use of sensing and control methods affecting one or a small number of cores. It has even been argued that the processor "core has become the new transistor" [78]. While this may be true from a system architect's point of view, it is not sufficient for PAC. Many of the phenomena motivating PAC are much finer than a processor core which can contain millions of transistors. Imagine the vias connecting wires on adjacent layers of a chip; highly-resistive vias are expected to become a common problem according to ITRS projections [52]. If the only compensation method is to turn off an entire processor core, system efficiency will drop precipitously.

For the above reasons we believe the case is strengthening for finely reconfigurable platforms such as FPGAs. These enable sensing on the scale of small clusters, and reconfiguration at the level of LUT, flip-flops, wires and switches. It may even become possible to infer the properties of individual transistors inside latches, as will be seen in Chapter 4.

There is an important additional rationale for FPGA-based implementations. They tend to have a moderate density of power and heat, since the logic is spread across islands surrounded by interconnect, and many resources are left unused. Previously, this low density and higher resulting cost per chip were considered significant disadvantages in

many domains. Application-specific integrated circuits (ASICs) and microprocessor units (MPUs) were much denser, providing higher performance and lower cost per chip at high volumes. However, the situation is changing dramatically as power and heat densities severely constrain the scaling trends. Dense platforms are generating more heat than can be easily removed from a system, forcing much of a chip to be left idle. Some studies of many-core processors have indicated that the majority of the cores must be idle, a problem sometimes called "dark silicon." In a recent keynote address, ARM's Chief Technology Officer Mike Muller projected that by 2020 and the arrival of 11 nm technology, the power budget may only allow 9% of a chip's transistors to be active at any one time [70]. Thus the density advantage of ASICs and MPUs over FPGAs is less compelling. In some domains, the choice will be between a dense, fixed architecture in which most of the logic is idle, and a sparse, reconfigurable architecture capable of much finer-grained PAC. For an increasing number of applications the finely-reconfigurable platforms are likely to win out.

1.4.7 Assisted Adaptation

We define *self-adaptation* as a process of system adaptation that is controlled and executed by a system itself. We define *assisted adaptation* as a special type of self-adaptation in which some of the resources for the adaptation process are provided by a remote server or peer. A portion of the introspection and/or optimization tasks are offloaded.

With PAC, the time scale of parametric shifts can be very long relative to the time-scale of computations. For instance, substrate parameters may change over a period of many days due to wearout. Moreover, some portions of the physical adaptation process can require more resources than are available in the system. This characteristic of need-

ing infrequent, heavyweight resources suggests a role for assisted adaptation, in which a PAC system off-loads portions of the adaptive process to a remote server. For instance, standard place and route tools generally require server-class memory and computing resources and cannot feasibly run on a highly-constrained embedded system. Likewise, estimation of certain logical parameters, such as error propagation probabilities using statistical fault injection, can require high-performance computing.

Offloading execution to a remote machine has been realized in more abstract contexts such as "cyber-foraging," in which a highly-constrained system searches over a network for usable resources [39]. Now there are several indications that such an approach holds promise in the PAC context. Embedded systems with network connections are proliferating, and there have been significant advances in "cloud computing." As an example, frameworks are being developed for evaluating the energy efficiency of local vs. remote execution [60]. Second, preliminary studies such as one by Hyder *et al*. [48] illustrate remote execution for a type of physical adaptation, using multiple systems that are peers instead of using a server. Third, missions such as the Cibola Flight Experiment have demonstrated that many of the pieces are already in place [19]. In that mission, physical data regarding faults and temperature swings on FPGAs is regularly transmitted to the ground. After re-design and refinement on the ground, new FPGA configurations are securely uploaded to the spacecraft. Up to 20 uncompressed configurations can be maintained on-board. Though not fully automated and not as fine grained, the process is analogous to assisted adaptation. Lastly, the first cloud for compiling FPGA applications came into existence in 2010 courtesy of National Instruments [58]; see Figure 1.11. The LabVIEW FPGA 2010 beta tool allows compilation jobs to be farmed out to a cloud.

The jobs are performed in a fault-tolerant fashion by machines with high amounts of RAM and located at data centers operated by Amazon, Microsoft, or others. Advanced security measures are employed just as with other high-security applications. Thus much of the infrastructure needed for assisted PAC is arriving.

Figure 1.11: FPGA Compile Cloud Service proposed by National Instruments [58]

1.4.8 Proposed Architecture

We now outline an architecture for a PAC agent. The main objective of the agent is to generate physically-adapted configurations that optimize fitness. It has at its disposal a set of interfaces, a library of routines, a bitstream containing the original configuration, and introspection data. It uses these to build and reason about internal models, and to take appropriate actions.

The actions available to the agent can be categorized as follows. 1) Introspection: the agent must seek out data to help in estimating parameter values. Examples include performing a self-test of the reconfigurable substrate, collecting sensor data, and requesting logical data about an application from a remote server. 2) Optimization: the agent must decide whether to re-implement some or all of an application, and by which method. Examples include performing incremental re-placement or re-routing, inverting signal

polarities [129] to compensate for data-dependent wearout, and requesting a full variation-aware re-implementation cycle from a remote server. 3) Selection of configurations: the agent may need to load a self-test configuration, load a newly implemented configuration so it can be validated on the substrate, or load a validated configuration having the highest fitness.

A generic PAC system is sketched in Figure 1.12. The adaptation agent process executes periodically as needed, either on a separate processor chip, on a hard core that is part of the same substrate (e.g., a PowerPC or ARM core [123] in Xilinx FPGAs), or even on a soft core implemented in the reconfigurable logic.

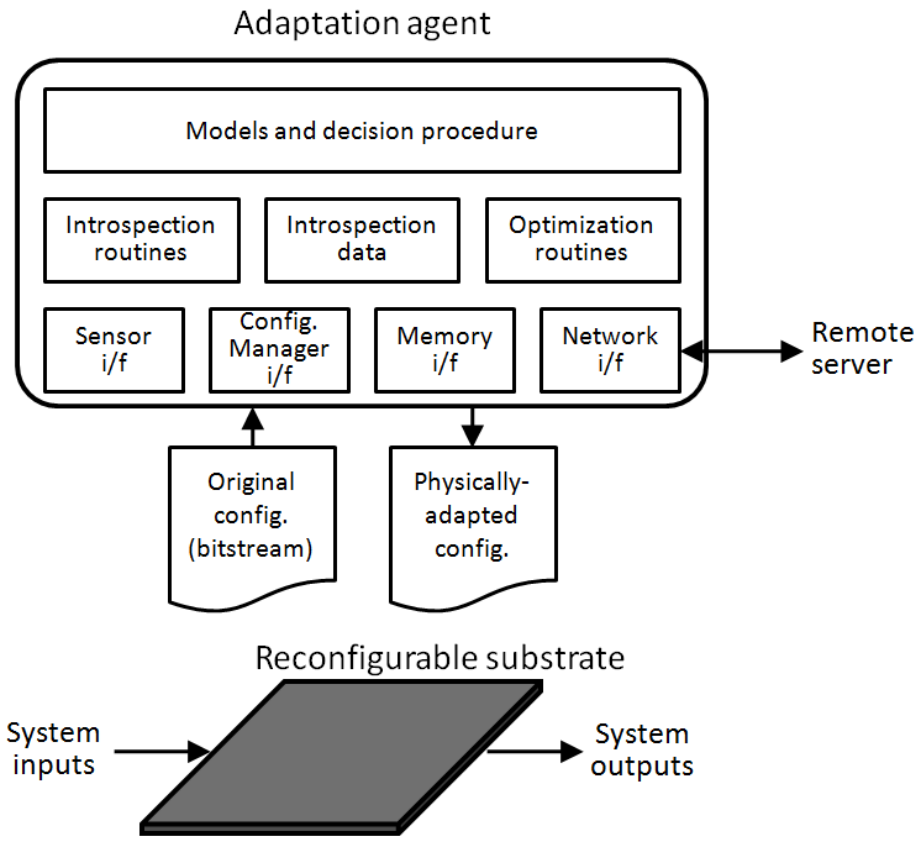

Figure 1.12: Sketch of a generic PAC system

The decision procedure for the adaptation agent can take a variety of forms. When the adaptation process is straightforward, a rule-based decision procedure may be sufficient. An example of a simple rule is, "If the time since the last self-test is at least 24 hours, run self-test now." This will be the approach used in the case study presented in Chapters 5 and 6. A question for future work is the extent to which the decision procedure itself can be made adaptive. A variety of cognitive algorithms have been proposed for use in adaptive computing systems [33][10][86].

A simplified flow of the adaptation process is shown in Figure 1.13. In contrast with some previous flows of this type [105], the process need not be a linear progression of characterization-optimization-reconfiguration. The agent may need to repeat certain types of actions, schedule actions to be performed remotely, and compete for resources with other agents.

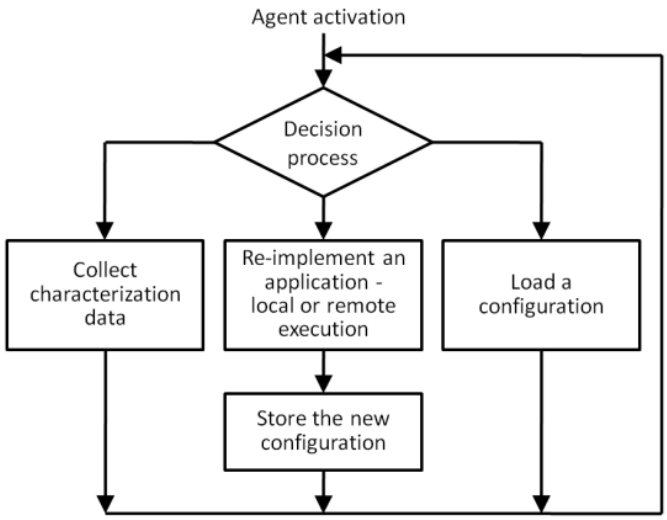

Figure 1.13: High-level view of the adaptive loop

Here is a brief example of a PAC scenario. After a specified time interval, the operating system activates the adaptation agent. The agent decides to collect sensor data

while the rest of the system operates normally. After processing, the data indicates that one region of the substrate is operating much more slowly than average. The agent must query the sensors a second time in case the slowdown was caused by a transient phenomenon such as a voltage "droop." When the problem persists, the agent requests that a self-test of the substrate be performed at the next available opportunity. The self-test data confirms an inherent slow region in the substrate. Since the problem is regional and not highly localized, a global optimization algorithm requiring server resources is in order. The agent sends the relevant data to a remote server and requests that a new configuration be generated. The agent process terminates and the rest of the system continues operating. Within 24 hours the adapted configuration is securely uploaded to the system. The agent is re-activated and requests that the new configuration be loaded and validated. Upon success, the new configuration is put into service and system health is evaluated. The original substrate-independent configuration is maintained as a backup. The agent process then terminates until the next interval.

1.5 Dissertation Outline

The main contributions of this dissertation are a conceptual framework for physically-adaptive computing, new methods of introspection, previously unpublished evidence of physical variations, and new methods of self-optimization.

The middle four chapters of this dissertation are organized along two dimensions as shown in Table 1.2. Chapter 2 covers the problem of estimating regional variations in system parameters. We introduce a flexible, highly-compact sensor that can be implemented with reconfigurable logic and embedded in an application. An array of such sensors can be used by a system to gain useful information about delays, voltage transients,

and even leakage variations. We then cover self-optimization for regional variations in Chapter 3. We demonstrate PAC at the level of function netlists, and conduct case studies of three applications. In the case studies, we suggest using a computational vulnerability parameter in the cost functions, and we introduce a comprehensive soft error metric for this purpose. Results show that by identifying and selecting the design best fitted to a region, fitness can often be improved by 30%–40%.

Table 1.2: Organization of middle four chapters

Topic	**Introspection**	**Self-Optimization**
Regional variations	Ch. 2	Ch. 3
Local variations	Ch. 4	Ch. 5

Chapter 4 covers introspection for local physical variations. We conduct a unique study of local variations in transient fault reliability and establish limits of conventional characterization. We then propose on-chip noise emulation as a much more feasible method of self-characterization. As a proof-of-concept, we demonstrate an FPGA-based system capable of injecting noise into latches and uncovering previously hidden variations. Chapter 5 is a study of self-optimization for local variations, using the introspection results from Chapter 4. For instance we evaluate the use of re-packing and local replacement on a set of benchmark circuits to compensate for random variations, and show how reliability can be improved via low-cost self-adaptation as well as assisted adapta-

tion using a remote server. Lastly in Chapter 6, we summarize our contributions, identify directions for future research related to PAC, and provide some final thoughts.

CHAPTER 2

Introspection for Regional Variations

Regional variations in physical parameters are leading to excessive power consumption, thermal hotspots, and reduced performance and reliability. Methods of low-cost, *in-system* characterization are needed. This type of introspection is one of the key aspects of PAC.

While some FPGA platforms include a single analog temperature and voltage sensor, they do not provide a means of measuring a variety of quantities at arbitrary locations. Thus one challenge is to use the standard digital logic in a reconfigurable fabric to measure as many key physical parameters as possible, and as unobtrusively as possible.

The key research questions include the following: Which types of physical parameters can be measured in reconfigurable systems? How can they be measured, and with what overhead? This chapter addresses these questions. The main contributions of this work, which was originally published in [132], are the following:

- The design of a flexible, compact, and easy-to-use sensor implemented in reconfigurable logic
- Improved procedures for measuring variations in delay, leakage power, dynamic power, and temperature

- A case study of a Virtex-5-based experimental system instrumented with over 100 sensors, including results from thermally-controlled experiments

2.1 Background

We briefly discuss some of the work related to on-line sensing of physical parameters. Reconfigurable platforms sometimes include an analog voltage sensor and temperature sensor connected to an A/D converter, such as with the Xilinx System Monitor [125]. However, the sensor coverage is limited to a single fixed location. An early paper by Quénot *et al*. proposes the use of distributed ring oscillators to measure the voltage and temperature profile across an entire chip [81]. Works such as [57] describe associated measurement procedures. A method of measuring leakage on a per-core basis in multi-core processors was put forth in [127]. Regarding FPGAs in particular, there have been several attempts at on-line thermal sensing using ring oscillators, including [22][27][50][64]; unfortunately, these methods entail relatively high overhead, and are becoming less accurate since the temperature dependence of advanced reconfigurable logic ($\leq$ 65 nm) increasingly is swamped by voltage noise. Additional research focuses on finding an appropriate arrangement and density of sensors across the reconfigurable fabric [69]. In this chapter, we build upon the approaches mentioned above.

There has been little work published regarding on-line measurement of FPGA parameters other than temperature. An example of off-line characterization of FPGA component delays is the work by Sedcole and Cheung [90].

A brief consideration of some relevant application domains helps to motivate fine-grained introspection. Many embedded systems are required to exhibit low power, high

reliability, and high autonomy. Examples include the FPGA-based systems deployed in spacecraft, UAVs, and similar systems [31][106]. Certain physical effects can be handled with coarse-grained control techniques (e.g., chip-wide schemes for voltage/frequency scaling, body bias, or power gating). However, extreme variation calls for a finer-grained and more adaptive approach. Data on regional variations may enable improved efficiency and lifetimes in these embedded systems.

Another relevant application domain is high-performance reconfigurable computing. As with embedded systems, there is a need for fine-grained sensing of power, temperature, and wearout. One difference is that high-performance systems may be more amenable to mitigation methods such as swapping out a failing FPGA, thermal-aware task scheduling [61], or on-line design optimization. Note that high-performance FPGA coprocessors are increasingly created via high-level synthesis, meaning low-level physical effects such as hotspots may go untreated. In fact, there are proposals for performing lightweight circuit synthesis at run-time [107]. The trend towards high-level design increases the need for handling low-level effects at run-time.

2.2 Instrumentation

To enable effective on-line adaptation for physical effects, an FPGA-based system needs to be instrumented with the ability to measure physical parameters. The nature of spatial variations across an FPGA chip necessitates that measurements be relatively fine grained, while system constraints dictate that the solution be low cost and efficient. A key question is how instrumentation can be designed to achieve these conflicting goals.

One common type of on-chip digital sensor is based on a ring oscillator circuit that feeds a frequency counter. The ring oscillator acts as a test circuit because its frequency is sensitive to parameters such as voltage. The oscillator is enabled for some reference period and the number of pulses is counted. The measured frequency is then used to estimate the physical quantity of interest. The utility of such sensors for FPGA systems has been limited by their high sensitivity to supply voltage fluctuations, and by the associated hardware overhead.

We now consider a much more efficient method of implementing the requisite frequency counters, as well as a compact design of a ring oscillator. Subsequently, we describe the proposed sensor and its methods of operation and access.

2.2.1 Efficient Counting

One approach to frequency counting is to implement a single centralized counter shared by multiple sensors [64]. Aside from routing congestion, this approach only allows one sensor to be enabled at a time, which prevents a snapshot of a spatial profile to be captured. (A parameter like dynamic power can change dramatically in the several milliseconds needed for serialized measurements.) Furthermore, certain parameters require a system to be paused during measurement, so serialization would cause a linear decrease in performance. For these reasons, we need a compact counter that can be instantiated in each sensor and operated in parallel.

The goals for the counter design include not just compactness but also ease of use by the software that reads and decodes the sensor data. Traditionally these two goals have been mutually opposing, forcing a trade-off. A standard *binary counter* uses a binary representation for the count as the name implies; this format can be readily processed in

software without special decoding. Unfortunately, a binary counter consumes a relatively large amount of hardware resources; a counter with period M requires $\lceil \log_2 M \rceil$ LUTs and flip-flops. An alternative design is a linear feedback shift register counter (*LFSR counter*), which can be implemented very compactly using Xilinx's shift register LUTs (SRLs). For instance, a counter with period $2^{15}-1$ can be built from an SRL, a flip-flop, and a LUT configured as an XOR [124]; this requires just two LUTs, compared to 15 for a binary counter. However, the count value is scrambled by the LFSR and must be recovered by solving the discrete logarithm problem [25]. The discrete log is normally difficult to solve, which is why it has uses in the field of cryptography. LFSRs have been used as event counters in cases where slow off-line decoding is acceptable, such as in silicon debug [25]. The required overhead can be reduced somewhat through the choice of the LFSR's characteristic polynomial, but remains non-trivial. A series of exponentiations must be performed (via polynomial multiplication), followed by lookups to residue tables that must be held in memory, followed by an application of the Chinese remainder theorem:

$$count = \sum_{i=1}^{k} r_i \left(\frac{M}{m_i}\right) v_i \bmod M, \tag{2.1}$$

where k is the number of moduli, r_i is a residue, M is the counting period, m_i is a modulus, and v_i is a weight found with the extended Euclidean algorithm.

We propose a counter design that is both compact and relatively easy to decode—a *residue number system (RNS) ring counter*. An example is shown in Figure 2.1. This style of counter was proposed long ago for generating timing signals [32], but to our knowledge its potential as an extremely efficient event counter for FPGAs has been unrecognized. The design is composed of multiple shift registers of varying lengths, each

feeding back to itself. Each shift register acts as a modulo-m_i "ring counter" and can be implemented with a m_i-bit SRL plus an optional flip-flop. The shift registers are initialized with a pattern such as a one-hot codeword. When the count is to be incremented, the patterns shift to the right; the position of each "hot" bit represents the count value modulo m_i, in other words, a residue r_i. A modulo-3 ring simply cycles through the states of 100, 010, and 001. By implementing rings whose moduli are all pairwise relatively prime, the counter can reach a period of $M = \prod m_i$. In the small example shown in Figure 2.1, the moduli are five, seven, and nine, and thus the counter period is $5 \times 7 \times 9 = 315$. The hot bits started to the left, and the counter was advanced ten times. For instance, the hot bit in the bottom counter has wrapped around twice and returned to the leftmost position, since 10 mod 5 = 0. Since the residue values are readily available simply by reading out the individual shift registers, the RNS count value can be easily recovered just by applying the Chinese remainder theorem (2.1). In this example, the residues are one, three, and zero, and the resulting decoded count is ten.

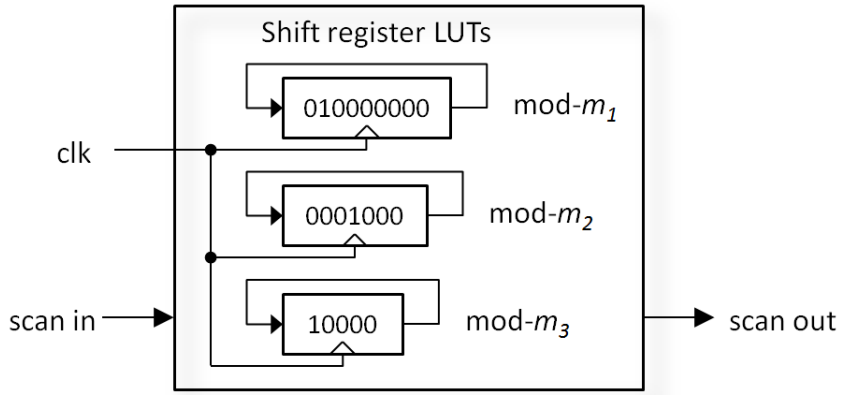

Figure 2.1: Compact counter design using the residue number system (RNS). All moduli are pairwise relatively prime. The period is $M = m_1 \times m_2 \times m_3$.

LFSR Counter Decoding Algorithm
y ← *polynomial contents of LFSR; // read counter*
for *i* = 1 **to** *k* **do**
  for *j* = 1 **to** 2 $\lceil \log_2(M/m_i) \rceil$ **do** *// exponentiations*
    $y_i \leftarrow y_i \times y$; *// polynomial multiplication*
  end
  $r_i \leftarrow TABLE(y_i)$; *// look up residue from table in memory*
end
for *i* = 1 **to** *k* **do** *//* Chinese remainder theorem
  count = *count* + r_i (M/m_i) v_i mod *M;*
end
return *count;*

(a)

RNS Ring Counter Decoding Algorithm
r ← *counter; // read k residues from counter*
for *i* = 1 **to** *k* **do** *//* Chinese remainder theorem
  count = *count* + r_i (M/m_i) v_i mod *M;*
end
return *count;*

(b)

Figure 2.2: Sketch of algorithms to recover the count for (a) an LFSR counter [25] and (b) the proposed RNS ring counter

With the proposed counter, most of the processing overhead of an LFSR counter is avoided, and no memory is required for tables. Pseudocode for the two counter styles is shown in Figure 2.2. Consider an example in which a counter of period 2^{13} is required. A comparison of the three design styles is shown in Table 2.1. An RNS ring counter can be implemented with just two LUTs, using moduli of 33, 17, and 16. With the Virtex-5/6, one LUT acts as a 32-bit SRL and one acts as two 16-bit SRLs. This two-LUT design is far more compact than a binary counter, and just as small as an LFSR counter. Moreover, the RNS ring counter requires no tables in memory, while the LFSR counter requires multiple tables totaling on the order of hundreds of bytes. Lastly, the RNS ring

counter is almost as trivial to decode as a binary counter, comparing favorably to the dozens of polynomial multiplications needed for an LFSR counter.

Table 2.1: Comparison of counter designs assuming a maximum period of 2^{13}

Counter type	**Hardware overhead**		**Ease of decoding**
	Logic (LUTs)	**Memory (Bytes)**	
Binary	13	0	No decoding
LFSR using SRLs	2	Hundreds [25]	Dozens of polynomial multiplications; ≥2 table lookups; ≥4 integer ops
RNS ring using SRLs (proposed)	2	0	6 integer operations

2.2.2 Ring Oscillator Design

The main goals for our ring oscillator are compactness, and sensitivity to relevant physical parameters. The basic design is shown in Figure 2.3. As with standard ring oscillators, it includes an odd number of inversions, and an on/off control switch. Two aspects of the design are non-standard. First, a latch is instantiated along with each LUT. The latch is held in the open state and acts as additional delay, increasing the fraction of ring oscillator delay that is due to transistors. Since the effect of temperature is normally stronger on transistors than wires, this increases the overall sensitivity to temperature. This improved sensitivity was validated by a hardware experiment (detailed below in Experimental Results) comparing ring oscillators both with and without the latch. Other researchers have used latches in a ring oscillator [56], but the impact of latches on temperature sensitivity appears to have been unrecognized.

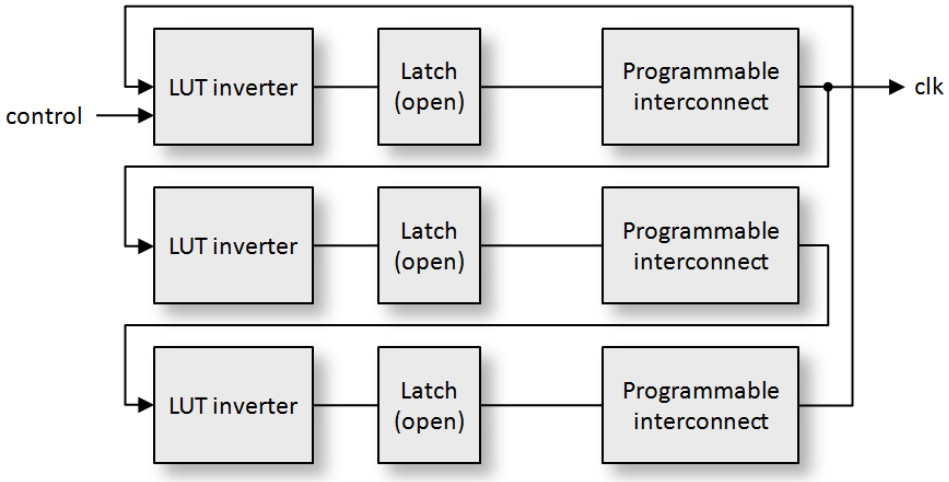

Figure 2.3: Proposed ring oscillator design

The second benefit of the proposed ring oscillator is its compactness, using only three stages. Ring oscillators for thermal sensing have used various numbers of stages, including 4 [14], 48 [50], and 75 [22]. One rationale given for a large number of stages is to cancel out local, uncorrelated variations, in the hopes that all ring oscillators will behave similarly. However, calibration would still be required due to regional variations. A second reason is to minimize "self-heating," since a slow and sparse design has lower heat density than a fast and dense one. We conducted a hardware experiment comparing a compact 3-stage ring oscillator to a 21-stage, sparsely-placed design and found no evidence of self-heating problems. We also tested the ring oscillator using enable periods ranging from nanoseconds to milliseconds and found highly consistent frequencies.

The main challenge in using reconfigurable logic-based ring oscillators to measure delay or temperature is their high sensitivity to the supply voltage. We propose leveraging this sensitivity to measure various types of voltage drop, and to subsequently infer

useful physical parameters. The issue of voltage sensitivity will be discussed in Section 2.3.

2.2.3 Sensor Design and Deployment

The proposed sensor design includes a ring oscillator, a frequency counter, and logic for control and access. A high-level diagram is shown in Figure 2.4. The sensor is enabled by a reference timer that is either on-chip or off-chip. A single timer can be used to enable multiple sensors simultaneously. The sensor resolution is dictated by the ring oscillator frequency and the length of the timer pulse. For instance, a frequency of 250 MHz and a timer pulse of 40 μs allow a resolution of one part in 10,000. After a measurement, the sensor data is read out via a scan output. An array of sensors can be connected via one or more scan chains. Routing overhead includes the scan chain and two global signals: the timer signal and the scan chain enable. Embedded software controls the scan sequence and reads the sensor data. Note that the sensor data can be readily accessed without using the FPGA's built-in capability for reading out its configuration (readback), and similarly sensors can be initialized without involving the FPGA reconfiguration process.

A distinguishing feature of the proposed sensor is its compactness. The entire design fits into a single Virtex-5 configurable logic block (CLB) containing just eight LUTs. This includes logic for scan and for synchronizing the timer signal to the ring oscillator clock. As can be seen in Table 2.2, this is much smaller than previous designs that have been proposed for thermal sensing, even after normalizing to a single architecture.

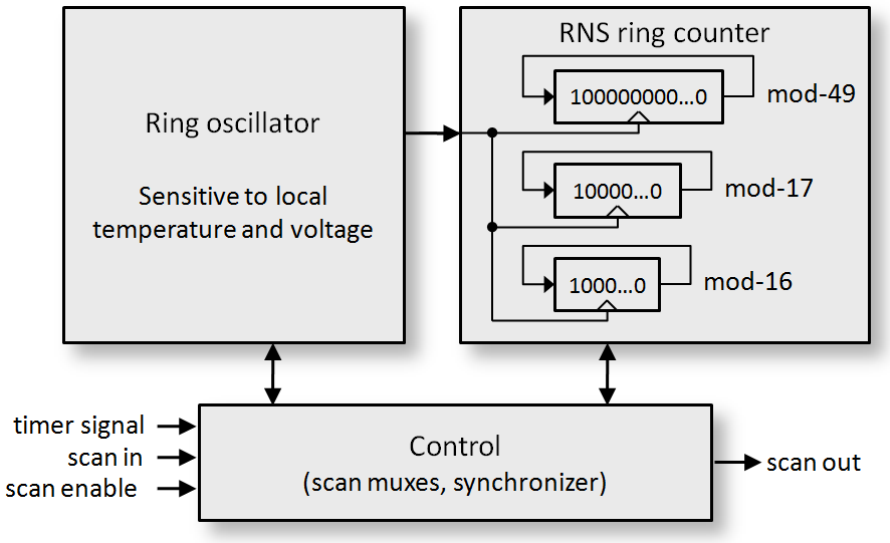

Figure 2.4: High-level diagram of proposed sensor

Table 2.2: Comparison of reconfigurable-logic-based sensors

Sensor design	**LUT count (target platform)**	**LUT count normalized to Virtex-5**
Chen *et al*. 2007 [22]	140 (Altera ACEX)	≥ 70
Jones *et al*. 2007 [50]	100 (Virtex-4)	≥ 50
López-Buedo *et al*. 2004 [65]	34 (Virtex-1)	31
Design proposed here	8 (Virtex-5)	8

The sensor design can be defined with a schematic or an HDL netlist. Sensor locations can be easily specified by using physical constraint statements such as Xilinx's RLOC_ORIGIN. Directed routing statements are used to ensure consistent ring oscillator instances. Software drivers contain the code for operating the sensors and for accessing, decoding, and processing the data. By integrating the sensing infrastructure into a traditional system design flow, as shown in Figure 2.5, an instrumented application can be generated.

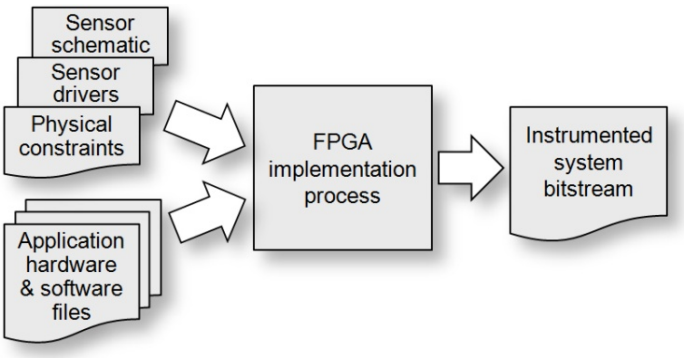

Figure 2.5: Overview of the proposed design flow for FPGA system instrumentation

An interesting question is, what is an appropriate spatial arrangement of sensors? Related works involving FPGA-based thermal sensors have either suggested a regular rectangular grid [65], or an irregular arrangement focused on expected application-specific hotspots [69]. Here we seek to perform systematic measurements of a variety of physical parameters, and thus employ a regular grid; specifically, we recommend a hexagonal tessellation due to its efficient area coverage [113]. An example of a hexagonal sensor array can be seen in Figure 2.6, where each sensor (shown as a light rectangle) has six nearest neighbors that are nearly equidistant.

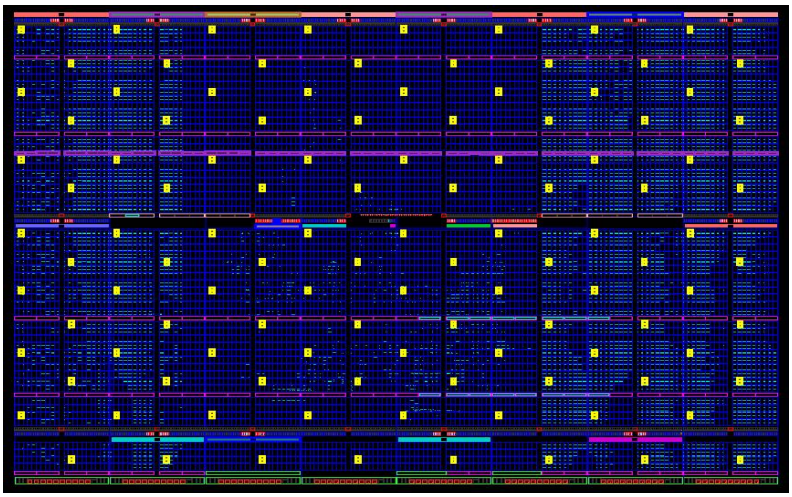

Figure 2.6: Floorplan of an FPGA-based system instrumented with an array of compact multi-use sensors (light-colored rectangles) implemented in reconfigurable logic

A second question is, what is an appropriate density of sensors? In the past, this has been largely dictated by hardware overhead. Most work on FPGAs involves between one and 32 sensors per chip. One work for ICs suggests sensing at 96 locations [57]. With our proposed compact design there is the possibility of deploying over 100 sensors while incurring only 1% overhead. In principle, over 12,000 sensors could be instantiated in a high capacity Virtex-5 while using only half of the available logic. The actual density should match the granularity of the physical phenomena being sensed. One relevant measure is the distance at which areas of the chip are correlated (correlation distance). Sylvester *et al*. demonstrate that higher sensor densities enable increased accuracy, for instance in predicting chip lifetimes [103].

2.3 Measurement Procedures

In this section, we address three general questions: which physical parameters can be measured in a reconfigurable fabric, how is such data useful, and what are the procedures for measurement? After defining some necessary concepts, we introduce enhanced procedures for measuring variations in four key parameters: delay, leakage, dynamic power, and temperature.

Ring oscillators have been widely used for sensing variations in delay. They have also been proposed for sensing temperature. The parameters of delay, temperature and voltage are all closely related. For instance, at typical supply voltages, a ring oscillator will slow down with higher temperature, but speed up with higher voltage. If two of the three related quantities can be measured accurately, then the third can be inferred. This is typically how researchers have proposed using ring oscillators in reconfigurable fabrics.

For instance, the ring oscillator delay is measured at a known voltage, and then a model is employed to estimate the temperature.

The traditional approach to ring oscillator-based sensing, for thermal sensing in particular, suffers from several limitations. One is that actual supply voltages are often lower than intended due to a phenomenon called *static voltage drop* (also known as *static IR drop* [3]). Supply voltages can furthermore undergo rapid, high-frequency swings after dramatic changes in switching activity. We refer to these as *voltage transients* (also known as $L\,{}^{di}/_{dt}$ events or *voltage droops* [57]). Lastly, for platforms such as the Virtex-5, measurements of the supply voltage can only be taken at a supply pin (we call this voltage VCC_p), and with limited resolution.

2.3.1 Delay

Every chip has fast regions and slow regions, due to factors such as imperfect lithography and variations in transistor critical dimensions [90]. By mapping and adapting to delay variations, the system frequency can potentially be improved.

Regional delay variations can be mapped via controlled measurements of ring oscillator frequencies across a chip. A traditional procedure is to configure an FPGA fabric with a special test circuit containing ring oscillators. The chip is left in an idle state for an extended period (seconds to minutes) to eliminate voltage or temperature gradients caused by switching activity. Then the ring oscillators are sampled, either one at a time or (when feasible) simultaneously. The procedure can be repeated for a range of temperatures and supply voltages if necessary. We propose using a traditional procedure but with a key difference: the sensors are embedded in the application logic and the sensing is performed in the field. This reduces the bottleneck that occurs with expensive testers

at manufacturing time. Furthermore, with an instrumented application there may be no need for storing a separate test bitstream and for performing reconfigurations. The delay characterization can be performed occasionally during a system's lifetime to track gradual, regional shifts caused by wearout.

2.3.2 Leakage

Leakage is a phenomenon in which current flows through a transistor at the wrong time (i.e., when turned off) or wrong place (gate oxide), as with a leaky water faucet or pipe. The amount of variation is much larger than it is for delay; the ratio of leakage from worst-case to best-case transistors on a single FPGA die has already passed 2.0 [111]. Two main types are sub-threshold leakage, which is very sensitive to temperature, and gate oxide leakage. In fact, with upcoming high-K metal gate transistors used on 28 nm FPGAs, gate oxide leakage will be largely mitigated [11], meaning the relative importance of sub-threshold leakage (and thus temperature) will increase. Hot transistors leak more current and generate more heat, driving a positive feedback loop. It would be helpful to know a chip's leakage profile so the problems of static power, thermal hotspots, and temperature-dependent reliability could be more effectively mitigated.

To our knowledge there has been no low-cost method of sensing a leakage profile in FPGAs. The authors of [98] lament, "Programming leakage sensors on the FPGA may not be feasible to employ due to their analog nature." Given a chip package and laboratory equipment, one may be able to carefully measure the leakage currents at various power supply pins and map the data to regions on the die inside. We propose a relatively straightforward method of *in-system* characterization, using our digital sensors. First, we explain the physical mechanisms at work, and then describe the measurement procedure.

Historically, ring oscillator frequencies have been modeled as linear with temperature, assuming a constant supply voltage. However, leakage current increases exponentially with temperature, and on modern chips such as the Virtex-5, this current is high enough to cause noticeable drops in the supply voltage and thus ring oscillator frequencies. Temperature-dependent voltage drop can be observed at the supply pins simply by reading the built-in analog voltage sensor at different ambient temperatures. For instance, we have found that this global drop increases by roughly 1 mV from 75°C to 85°C on the Virtex-5 we tested. We also separately found evidence of *regional* on-chip voltage drops caused by regional currents interacting with the on-chip power grid. At higher temperatures, all ring oscillators will exhibit some slowdown due to the global effects (namely a decrease in carrier mobility, and global voltage drop due to both leakage and increased resistance in the power grid). In addition, there will be a *variable* slowdown primarily caused by leakage-induced regional voltage drop. This slowdown provides evidence for the amount of leakage in the vicinity of the oscillator. While the absolute amounts of leakage are difficult to estimate accurately, this method provides an estimate of the relative leakage profile.

The measurement procedure involves performing the ring oscillator delay characterization from Section 2.3.1 at two different die temperatures. (Systems must either leverage natural swings in ambient temperature, control the die temperature via a fan, or if necessary, use a 'heater' configuration.) With these two measurements the effects of increased temperature on each region of the chip can be compared, and the leakage profile inferred.

As an example, imagine that ring oscillators spread across a die are sampled at a stable, nearly-uniform temperatures of both 0°C (where leakage is smallest) and 85°C. The difference in oscillator frequencies turns out to be 2.0% on average, largely due to global IR drop. However, one spatial region exhibits larger differences of 2.5%, and another close to 3%. These spatial anomalies can provide indications of high-leakage regions.

2.3.3 Dynamic Power

Dynamic power can have an uneven spatial distribution due to uneven application activity. This leads to thermal hotspots, reliability issues, and extra power consumption. For example, the power supply may have to be set to a high voltage to offset the worst-case regional voltage drop. Dynamic power profiles can usually be estimated at pre-manufacturing time, or in some cases with performance counters used at run-time. Nevertheless, certain systems may have an unknown profile, such as those with autonomously-generated configurations [107]. Sensing the profile in these cases may allow improved configurations to be found. We now describe a simple method of inferring a dynamic power profile. The basic approach is to sample ring oscillator frequencies as the system is running, briefly pause the system until no switching occurs, and then immediately sample the frequencies again. A change in frequency indicates the size of regional voltage drop due to switching current. Temperature and leakage effects remain nearly constant since the time between readings is much less than the thermal time constant which can be tens of milliseconds. Very similar approaches have been used for sensing voltage variations [27][57], but here we exploit the fact that voltage drops can be used to infer dynamic power. Furthermore, we propose the following enhancement to previous me-

thods. Voltage transients can introduce large errors in ring oscillator data. We add a simple check for such events; instead of taking a single reading while the application is running we propose taking multiple consecutive readings. Each reading is spaced out by the length of the problematic transients (on the order of 1ms). A quick check of data consistency can determine whether a voltage transient event occurred. If not, the procedure can move forward and the system can be paused; otherwise new readings are required. This method is most practical for applications with a relatively steady-state power profile or activity phases much longer than milliseconds. The proposed procedure is as follows:

1. Sample ring oscillator frequencies twice while application is running
2. Check consistency; if a voltage transient is detected repeat 1
3. Pause the application
4. Wait for voltage transients (caused by pausing activity) to dissipate
5. Sample ring oscillator frequencies again
6. Resume the application
7. When convenient, compare the frequency shifts of ring oscillators

2.3.4 Temperature

Thermal hotspots lead to early wearout, lower operating frequencies, higher static power, and extra cooling costs. On-line sensing of a thermal profile can account for physical realities (e.g., variations in leakage or packaging) and allow for enhanced mitigation schemes. Unfortunately, thermal profiles are very difficult to measure with standard reconfigurable logic. The main problem is that circuit delays with modern CMOS technologies are no longer very sensitive to temperature [110]. With older technologies such as the Xilinx XC3000/4000 in the mid-1990s, ring oscillator frequencies would shift

by 20% over the full temperature range [14]; with the Virtex-5 we find the shift to be only 2.5%.

The dependence on voltage, however, is quite strong. On-chip voltages cannot be easily controlled or measured, adding noise to temperature measurements. Previous works tend to assume that by pausing system activity, the supply voltage very quickly becomes spatially uniform and thus temperature can be cleanly estimated. This is no longer valid in general, since leakage variations (greatly amplified by temperature variations) cause non-uniform drops in the supply voltage. A second problem is that the supply voltage at the pins (VCC_p) may not be measurable with fine resolution. For instance, the Xilinx System Monitor [125] reports VCC_p with only 3 mV resolution, which could introduce errors of 10°C–20°C into temperature estimates. A third problem is that voltage drop causes non-linearity in the delay, temperature, and VCC_p relationships, so the conventional linear models are becoming less accurate. Temperature sensing could be improved if additional built-in analog sensors were available on FPGAs, or if leakage current could be selectively disabled via power gating. Absent those, there is a need for new models and methods.

We propose the following procedure for estimating a thermal profile. First, ring oscillator measurements are made across the range of voltages and temperatures. A model of temperature as a function of delay and voltage is built, for instance via surface fitting in MATLAB. This characterization and modeling can be performed off-line and need only be done once for a given ring oscillator design and target platform type. An example is detailed in the next section. When a system is deployed into the field, variation-aware calibration is performed to find the coefficients of the temperature model for each

sensor. The data from delay and leakage characterization can be used for this calibration. At run-time, the chip region being sensed must be paused and left idle for long enough to allow any voltage transients to dissipate. The ring oscillator frequencies are then measured, and the supply voltage at the pins is sampled (an average of multiple readings can improve the limited resolution). This data is then plugged into the empirical model and the temperature around the sensor is estimated.

2.4 Experimental Results

We now describe our experimental setup and then present the results of sensing experiments involving delay, leakage, dynamic power, and temperature. We designed an experimental FPGA-based system and instrumented it with 112 of the proposed sensors. A block diagram of the design is shown in Figure 2.7. The design was instantiated on each of two Xilinx Virtex-5 FPGAs residing on XUPV5-LX110T boards. The reconfigurable fabric is composed of 160 × 54 CLBs, corresponding to 160 × 108 Virtex-5 slices (aka clusters). The sensors are arranged on a hexagonal grid of size 16 × 7 to fit the dimensions of the die. Sensors are nearly equidistant from each of their six neighbors. The layout can be seen in Figure 2.6, rotated right by 90 degrees. The sensor frequency counter was designed with moduli of 49, 17 and 16 for a counting period of 13,328. Key sensor specifications are shown in Table 2.3. Note that the readout time for the entire sensor array is about 100 μs, which is much faster than thermal time constants.

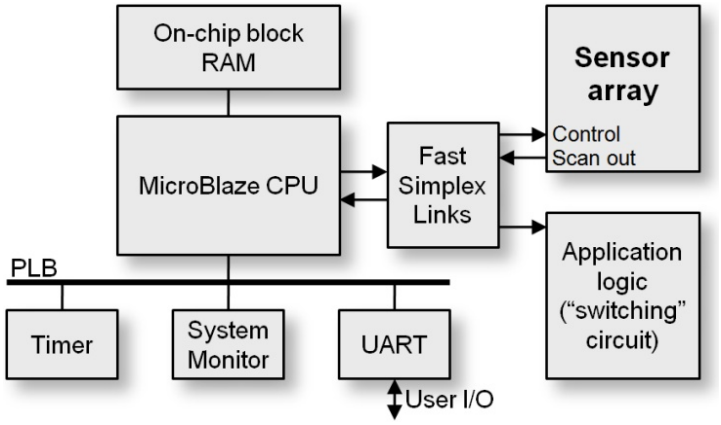

Figure 2.7: Block diagram of the experimental system implemented on a Virtex-5 FPGA

Table 2.3: Sensor specifications

Item	Specification
Hardware resources	8 LUTs per sensor (896 total)
Data size	82 bits (= 49 + 17 + 16)
Measurement period	40 μs
Resolution	1 part in 10,000 @ 40 μs
Sensor array readout time	~100 μs

The system contains a MicroBlaze 7.10.a CPU, and three peripherals connected to the processor local bus (PLB): an XPS Timer for enabling the sensors, a core for interfacing to the System Monitor, and a UART16550 serial interface for user input and output. All of these operate at 100 MHz. In addition, application logic was included for experimental purposes. The application logic acts as a switching/heater circuit with regions that can be independently enabled. It consists of 28,200 LUTs and flip-flops that can toggle at high speed. Both the array of sensors and the application logic are connected to the MicroBlaze via a shared set of fast simplex links of minimum size. The design was implemented with the Xilinx ISE and EDK 10.1 toolset. The experimental control software was written as a standalone application in C for the MicroBlaze, and fits into 32 KB of

block RAM. Thermally-controlled experiments were performed using a TestEquity 105A thermal chamber, as pictured in Figure 2.8.

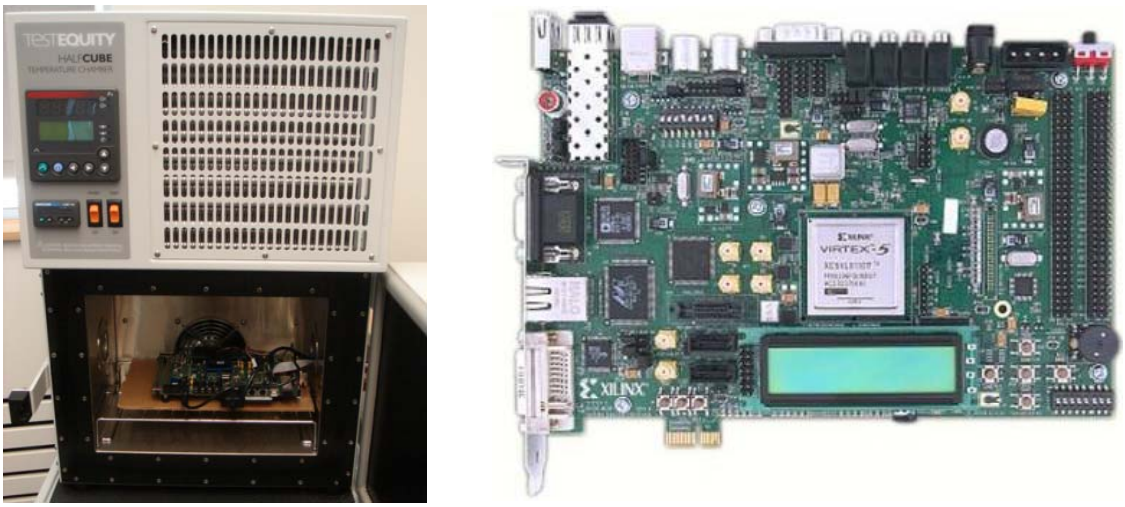

Figure 2.8: Experimental setup in the thermal chamber (left); close-up of an XUPV5 circuit board (right)

We first tested the proposed ring oscillator design, and compared its temperature sensitivity to a conventional design. Oscillator frequencies were measured at a range of temperatures from 0°C to 85°C, while the supply voltage at the pins was maintained at a constant 1.009 V. Results confirm that the proposed design has a stronger temperature dependence than the conventional design, as seen in Figure 2.9. The relative improvement in the slope is 17%.

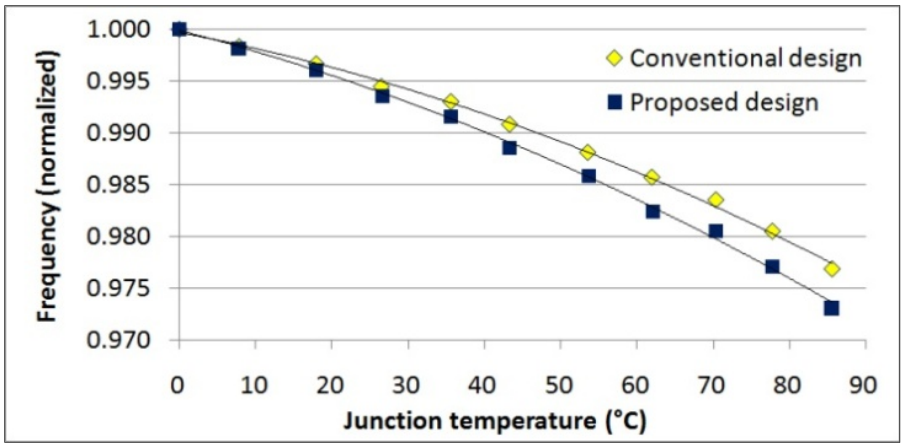

Figure 2.9: Frequency dependence on temperature for proposed ring oscillator design and conventional design

2.4.1 Delay

The first experiment using the proposed on-line measurement method is a characterization of delay variations. This type of measurement is required for calibrating the ring oscillators. The ambient temperature was set to 25°C and the system was left in the idle state until it reached thermal equilibrium. The junction temperature (T_j) was 35°C as reported by the System Monitor. Ten consecutive readings of the oscillators were taken. We found that the readings at individual sensors are highly consistent; the standard deviation for each set of ten readings is approximately 0.02% of the mean. In other words, the random measurement error for an oscillator with mean frequency 250 MHz is just 0.05 MHz.

The frequency profiles of the two FPGA chips are shown in Figure 2.10. The x and y coordinates represent Virtex-5 slice locations. Dots indicate the average frequencies measured at the locations of the sensors. The frequencies at locations between sensors are estimated using linear interpolation. Several observations can be made. Chip 2 is noticeably faster than chip 1, due to inter-die variation and board-level differences

(slightly different supply voltages and heat sinks). Here we are more concerned with *intra-die regional* variation. With both chips, the right side is the slowest region. The fastest region is near the top of the grid for chip 1 but near the bottom for chip 2. The distribution of measured frequencies is shown in Figure 2.11. The amount of spatial variation can be expressed via a coefficient of variation, which is the standard deviation σ across all sensor frequencies divided by the mean μ. Both chips exhibit a coefficient of variation of approximately 2.3%, which corresponds to a 3σ variation of approximately 7%.

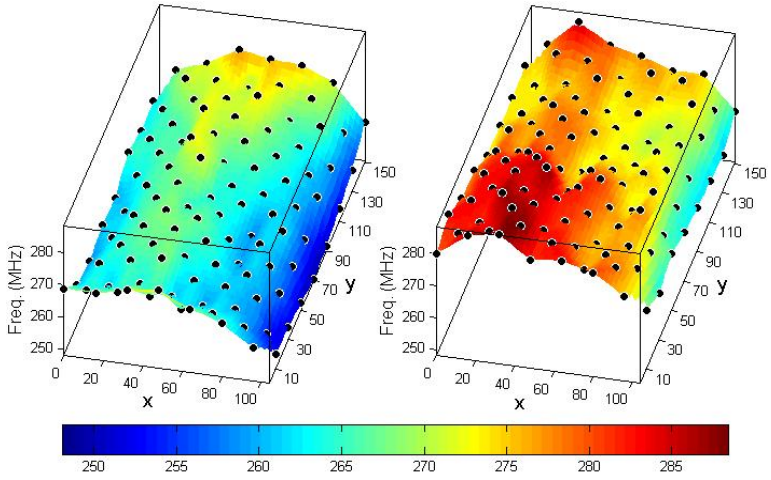

Figure 2.10: Frequency profile for chip 1 (left) and chip 2 (right) in the idle state at T_j = 35°C

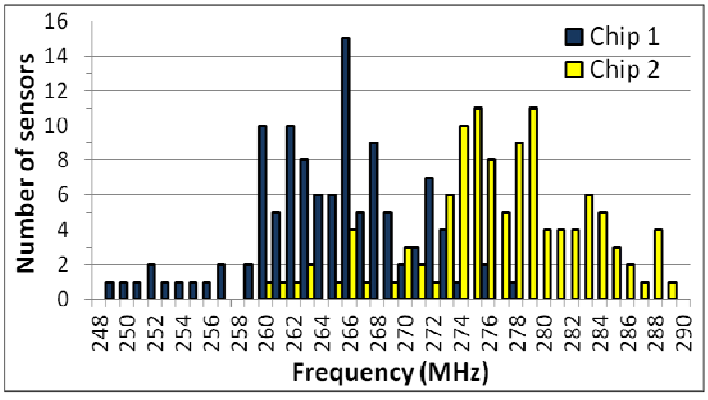

Figure 2.11: Histogram of ring oscillator frequencies for chip 1 (dark) and chip 2 (light) in the idle state at T_j = 35°C

2.4.2 Leakage

The second experiment is a characterization of a current leakage profile. We measured all ring oscillator frequencies while the system was idle, but this time at two different temperatures. The ambient temperature was controlled by the thermal chamber. The measured leakage variations are visualized in Figure 2.12, which shows the relative frequency shifts that occur when the steady-state junction temperature is changed from 35°C to 85°C. Larger shifts are evidence of higher leakage. Certain ring oscillators on chip 1 slow down by 1.6%, while others slow down by 2.7%. For chip 2 the range is 1.3% to 2.4%. Ring oscillator frequency and leakage both depend on threshold voltage, so one might expect the two parameters to be highly correlated. In other words, fast regions might tend to be leaky regions. Analysis of the intra-die experimental data shows a weak correlation. The correlation can be measured using linear regression and the R^2 statistic. A value of 1 indicates perfect correlation. We found the R^2 value to be 0.19 for both chip 1 and chip 2.

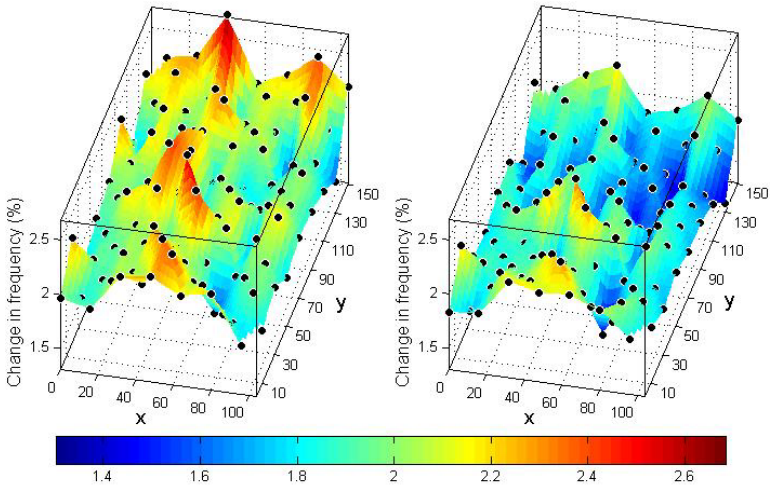

Figure 2.12: Map of leakage current variations for chip 1 (left) and chip 2 (right)

2.4.3 Dynamic Power

We next test the ability to sense a dynamic power profile. A demonstration application was set up to generate high switching activity near the top and bottom of the reconfigurable fabric. The middle portion was left idle so that clean measurements could be taken and validated against the System Monitor, which sits near the center of the die. Specifically, 4,500 slices toggled in the upper third of the die with y coordinates ≥110, and 2,550 slices toggled in the bottom portion of the die. The proposed procedure was applied, isolating the effect of switching activity. Measurements indicate that ring oscillator shifts are indeed largest in the two regions with switching activity, as shown in Figure 2.13. The image is rotated to illustrate the changes along the y axis. This type of measurement provides important information about the spatial extent of switching effects. For instance, while the application was set up with a dramatic discontinuity in activity at y = 110, the measured frequency shifts are surprisingly continuous, showing a relatively even slope that begins around y = 100 and extends all the way to the top of the die.

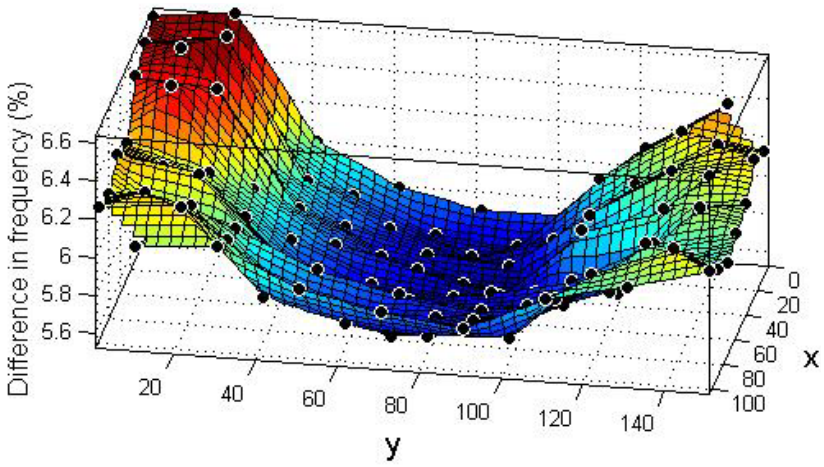

Figure 2.13: Map of frequency changes due to switching activity

2.4.4 Temperature

In this final experiment we consider temperature variations. We first build an empirical model of frequency, temperature and voltage. The sensor array is placed such that one sensor is at $(x,y) = (48,81)$ immediately adjacent to the System Monitor, allowing temperature estimates for that location to be validated against the built-in analog sensor. We want to know what type of model fits the data and with what accuracy. We measured frequencies over a range of temperatures from 0°C to 85°C, and over a 40 mV range of voltages. The supply voltage VCC_p was indirectly modified by enabling various amount of application activity (affecting the static voltage drop), from zero activity all the way to 100% application activity in 10% steps (equivalent to about 600 slices of application logic). In total, 34 data points were collected, with ten readings at each point. Various models were fitted to the data using MATLAB, in order to find an effective temperature estimation function $T(x,y) = f\,(Freq(x,y), VCC_p)$, where $T(x,y)$ is the temperature at die location (x,y), and f is a function of frequency $Freq(x,y)$ and voltage VCC_p. We found that a traditional model with linear dependence on frequency and voltage provides a root mean square error of 5.7°C. In contrast, we found that a 2nd order polynomial model provides a better fit and lower error. The form of the model is as follows, with coefficients c_i:

$$T(x,y) = c_1 Freq(x,y)^2 + c_2 VCC_p^2 + c_3 Freq(x,y) VCC_p + c_4 Freq(x,y) + c_5 VCC_p + c_6 \,. \tag{2.2}$$

This model reduces the error to 3.5°C. Plots of the surface fit can be seen in Figure 2.14; the surface is curved and twisted rather than planar. The error with this model is compa-

rable to the accuracy of analog sensors such as the one built in to the System Monitor, which is rated to ±4°C.

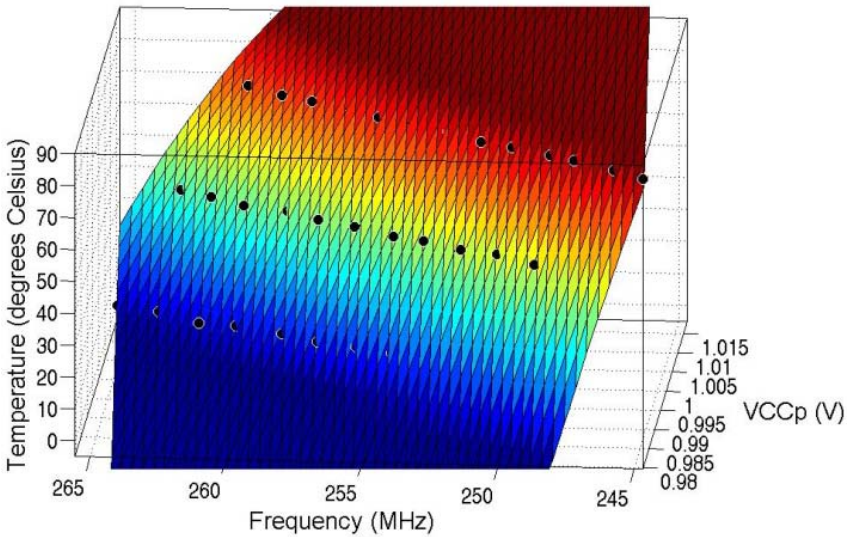

Figure 2.14: Measured relationship between temperature, sensor frequency, and supply voltage for the sensor at (48,81)

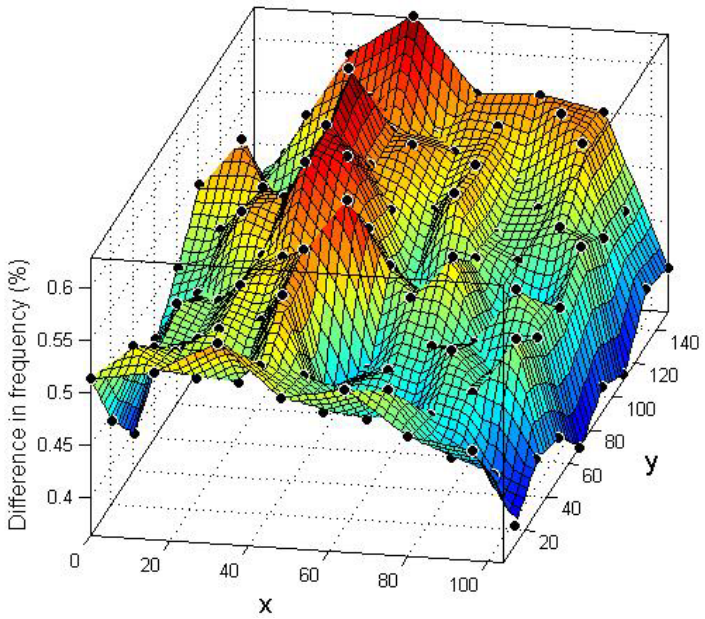

Figure 2.15: Map of frequency changes due to temperature effects

In addition to the validated measurements at location (48,81), we sensed the thermal effects across the entire fabric for the application mentioned above. Ring oscillator frequencies were measured immediately after the application was paused, and compared to the steady-state idle case. The resulting shifts in frequencies are illustrated in Figure 2.15. This profile includes not only the direct effect of temperature, but also the indirect effect caused by temperature-dependent leakage.

2.5 Summary

We have introduced an approach to on-line sensing of regional variations that includes a compact multi-use sensor, methods of instrumenting an application, and enhanced procedures for measuring physical parameters. This type of low-cost introspection enables a PAC system to gain important knowledge about its physical circumstances. Novel use of a residue number system counter yields sensors that are surprisingly compact, while an enhanced ring oscillator enables a 17% improvement in temperature sensitivity. The proposed sensor fits into just eight LUTs on a Virtex-5, and is nearly one-quarter of the size of the smallest previously published design. We have described procedures for measuring the profiles of four parameters of key interest to system designers: delay, dynamic power, leakage power, and temperature. The work suggests possibilities for future research involving additional physical parameters, measurement procedures, and sensor array arrangements. As demonstrated with our experimental system, the proposed approach is suitable for low-cost sensing of a variety of parameters, and is accurate enough to estimate spatial profiles. The regional data learned via introspection can be used to inform the PAC optimization methods discussed in the next chapter.

CHAPTER 3

Self-Optimization for Regional Variations

In this chapter, we describe methods of self-optimization for regional variations, using data gleaned by introspection. First we briefly discuss regional adaptive body bias and re-placement of circuits. Then we consider self-optimization at a higher-level of abstraction; given regional variations, alternative netlist-level designs are evaluated on-line and the fittest one selected. We argue for the importance of including soft error vulnerability in the netlist evaluation, and propose a new figure-of-merit for logical vulnerability. Then we demonstrate quantitative examples of PAC in three applications, with a cost function that includes area, throughput, and the proposed vulnerability parameter.

3.1 Adaptive Body Bias

The characteristics of a group of transistors can be altered by changing the bias voltage applied to the body of the devices. This is called adaptive body bias (ABB). Reverse biasing is when a negative bias voltage is applied, raising the effective threshold voltage. This generally improves leakage but slows down switching. Conversely, forward biasing is when a positive bias voltage is applied, lowering the threshold voltage and leading to a leakier device but faster switching.

ABB can be used in a chip-wide fashion at manufacturing time to account for die-to-die variations. With FPGA chips such as the Altera Stratix III and IV, it is now possible to perform regional ABB, with regions containing just two clusters (i.e., logic array blocks or LABs) and a total of just 20 8-input LUTs. The current state-of-the-art is for the ABB settings to be determined at pre-silicon design time based on static timing analysis [63]. Regions containing critical or near-critical path logic are set to the high-performance mode, and all other regions are set to the low-leakage mode. This approach is oblivious to substrate variations and assumes that using pre-manufacturing timing estimates is good enough.

Nabaa *et al.* propose an alternative ABB approach involving in-system characterization of delays in the actual substrate [75]. The accuracy of their delay measurement method is unclear, and there is no method of leakage measurement. Their simulation model oddly includes local, uncorrelated variations and does not address regional variations. Furthermore, the approach requires a special characterizer circuit that would have to be built into the FPGA as a hard macro; current FPGAs do not have such a circuit.

The methods of introspection presented in Chapter 3 can be used to improve upon the two previously mentioned methods and allow optimization of ABB settings in the field, as follows. For a given chip type, a self-test configuration is created that includes an array of sensors. The sensors are packed as densely as needed to match the correlation distances for delay and leakage. In the extreme case, a sensor can be placed at every single region (pair of LABs), which on the highest-capacity Stratix IV platform means roughly 10,000 regions. Once a system has been deployed, the self-test configuration is loaded and measurements are taken of the delay and leakage profiles. Critical regions

can be identified—these are regions that had been assigned the high-performance ABB setting at pre-manufacturing time, and also have been mapped to the slowest portions of the reconfigurable substrate. These regions establish a maximum achievable frequency f_{max}. Next, regions with extra timing margin (slack) are identified—these are regions that had been assigned the high-performance setting but have been mapped to substrate portions that are significantly faster than worst case. If the slack is greater than a desired threshold, the ABB setting is switched to low-power mode, reducing static power.

Conversely, regions that were initially placed in the low-power mode may have much less slack than expected if the delay is abnormally high. These could be switched to the high-performance mode to lessen the probability of soft errors due to timing violations.

This qualitative example suggests an opening for further work in this area. The approach is readily applicable to platforms such as the Stratix III and IV. A fully self-adaptive scheme could be enabled by vendor support for direct modifications of the ABB settings in the bitstream. In any case, PAC systems can leverage assisted adaptation by communicating the desired settings to a server running the Altera CAD tools. The server would generate the adapted bitstream and transmit it back to the PAC system.

3.2 Re-Placement

Once regional variations have been characterized as discussed in Chapter 2, a system can in principle be optimized via variation-aware placement. Cheng *et al.* proposed an algorithm for "chipwise" placement called vaPL, within the VPR framework [24]. This algorithm depends on accurate characterization data being available; the authors did not

address how this would be possible, admitting that "synthesis of test circuits to generate the variation map is ongoing research." The methods of Chapter 2 help to fill the gap. Cheng *et al.* simulated the proposed algorithm assuming total variation (die-to-die, regional, and local) of $3\sigma = 10\%$ and a correlation distance of 2 mm, and estimated an average improvement in circuit speed of 12%.

Another variation-aware extension to the VPR placement algorithm was proposed in [98]. The algorithm depends on accurate delay and leakage information being available, though once again no solution was provided. The authors felt that "leakage sensors on FPGAs may not be feasible." Assuming that methods of sensing became available in the future, their simulations showed the possibility of a 10% improvement in frequency and a simultaneous 14% improvement in leakage. Fortunately, the methods described in Chapter 2 now provide a low-cost method of estimating delay profiles and even getting a rough estimate of leakage profiles.

With both of the above placement algorithms, the assistance of a server would generally be needed. Note that the above frequency gains of 10%–12% may increase the price or utility of a system by much more than 10%–12% for applications that are performance-constrained.

3.3 Netlist-Level PAC

While Sections 3.1–3.2 considered optimizations that are made to a physical implementation of a function, we now consider applying PAC at a higher level of abstraction. Namely, adaptation for physical objectives can occur at the level of the function netlists *FN*. Netlists act as models of computational "designs" which may realize the same high-

level function, e.g., the same algorithm. Informed by introspection into regional variations, a system can consider alternative designs for use inside the region of interest, and dynamically select the one with highest fitness. Figure 3.1 is a simple illustration of the scheme. Alternative designs are maintained in a library, either on a server or in a PAC system itself. As an example, recall that the Cibola Flight Experiment can maintain 20 uncompressed bitstreams onboard [19]. A PAC system senses regional variations in substrate parameters and in dynamic parameters such as transient fault rates and regional supply voltages. The system reads the alternative designs' static parameters such as area and throughput, and combines them with the introspection data to perform on-line cost evaluation. The fittest design can be selected and scheduled for integration into the system configuration. The actual integration can occur via dynamic partial configuration or after generation of a new system configuration. The intent is not to reconfigure at the time scales of individual computations, which can be on the order of milliseconds, but rather on an occasional basis to match shifts in substrate parameters (over days, weeks, etc.) and fault environments (e.g., over minutes).

A scheme with some similarities to ours has been proposed in [47]. In that work, a mobile device transmits a user's computational needs to a server that selects among alternative versions of FPGA co-processors. The selected versions are then made available for download to the mobile device. The problem considered here differs in that it requires regional physical introspection and optimization, and more of the work is performed by the adaptive system itself rather than by a server.

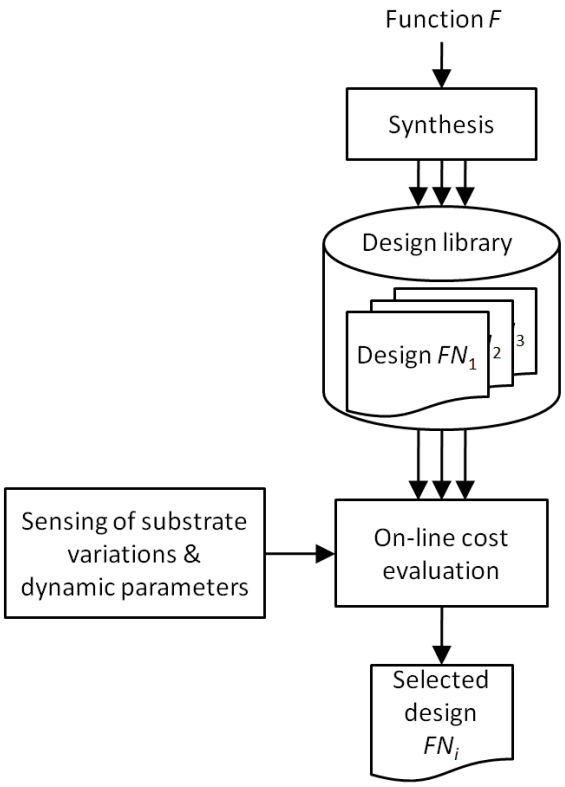

Figure 3.1: Dynamic selection of fittest design

A function F can be associated with a set of function netlists $\{FN_1, FN_2\ldots\}$ representing different computational designs of F. Each netlist FN_i has an associated configuration $Config_i$ which is a physical implementation targeting a specific platform type.

A system can be tessellated into a grid of physical regions with unique properties due to variations. In Chapter 2, we tessellated a chip into a hexagonal grid, with a sensor at the center of each hexagonal region as shown in Figure 2.6. Many circuits of interest, such as those implementing co-processors, are much smaller than the correlation distance and fit inside a region with nearly homogeneous properties. In other words, regional variations from the system perspective can appear as global variations from the subsystem perspective. The region where the circuit is placed acts as the computational substrate for

that circuit, denoted $Sub(x,y)$, where (x,y) is the region coordinate on a 2-dimensional chip.

PAC at the netlist level, just as with PAC at lower levels of abstraction, requires a cost function. As defined in Section 1.4.2, the fitness or cost of pair ($Config_i$, $Sub(x,y)$) is an application-specific function of system parameters. In the next section, we argue for the importance of considering soft error vulnerability of alternative netlists in the cost function. We propose a soft error metric to capture the logical vulnerability of computations, and show how it can be used as a parameter, along with introspection data regarding regional fault rates. In Section 3.5, we provide experimental results of soft error simulations and three case studies demonstrating netlist-level PAC with regional variations.

3.4 Computational Vulnerability

Just as transient faults are an important physical phenomenon, the tendency for soft errors to propagate and corrupt computational results is an important logical phenomenon. A full consideration of the soft error threat requires that logical vulnerability be measured and included as a logical parameter in system cost functions. In this study, we show that existing soft error metrics are a poor fit for this purpose. We propose a new metric and validate it with vulnerability experiments on three applications in Section 3.5. This study was originally published in [130].

Background. The integrity of computed results in leading-edge electronic systems is increasingly threatened, especially in scientific computations with large extent in space and/or time [18], computations performed under marginal operating conditions to save

power [128], and embedded computations subjected to harsh radiation [40]. In each of these domains full protection against soft errors is often infeasible, and thus the computations performed have some amount of intrinsic vulnerability.

One challenge is to quantify the relationship between low-level transient faults and the resulting system-level soft errors. A popular metric for characterizing this relationship is the fraction of faults that lead to errors. This fraction goes by many names depending on the level of analysis and on institutional preference, including architectural vulnerability factor [74], error cross section, residency, and logic derating [55]. We use the term *vulnerability fraction* (VF) as a catch-all for these nearly equivalent terms. A conventional approach is to multiply an SEU rate by a VF value to arrive at another widely-used metric, the system-level soft error rate (SER). SER represents the frequency of errors of a particular type, and is often expressed in units of failures-in-time (where one "FIT" equals one fault per 10^9 hours), or alternatively, as mean-time-between-failure (MTBF) or mean-time-to-failure (MTTF). The difference between MTBF and MTTF involves the repair time, but in the case of soft errors, the terms are nearly equivalent.

Architectural vulnerability factor (AVF) signifies the fraction of SEUs in a CPU structure that lead to an architectural error [74]. The prime methods of estimating AVF are architecturally correct execution (ACE) analysis and statistical fault injection (SFI), the merits of which are debated in [109][12]. Regardless of its accuracy, ACE analysis is applicable only to instruction set architectures (ISAs) and not, for instance, to computations performed via hardware-implemented algorithms in FPGAs or ASICs.

For our purposes, vulnerability fraction metrics are insufficient in that they do not measure the amount of vulnerability per computation. Thus they cannot be readily used

as a figure-of-merit for computations of different lengths or spatial exposure. For instance, two designs may have identical vulnerability fractions but have vastly different hardware area and thus vastly different vulnerability per result.

The system-level SER metric combines VF with an absolute rate of SEUs to estimate the absolute rate of computational errors. SER is a natural fit for certain soft error types, such as detectable unrecoverable errors. Such errors affect system availability and are often handled by aborting a computation and performing a recovery operation or system restart. Clearly it can be helpful to estimate the rate at which these occur. SER is far less suitable for undetectable errors that cause corrupted results, however, since it fails to account for the length of a computation. Two different designs may have an identical SER but greatly different vulnerability per result. Another drawback is the dependence on the SEU rate which may not be known, especially at design time.

A different type of modeling is classical reliability analysis. For instance, assuming that errors are exponentially distributed and have a constant rate λ, we can model the soft error reliability of a computation having length t as $R(t) = e^{-\lambda t}$. This type of analysis is common in reliability engineering and was applied to computational reliability in [71]. Unfortunately, when it comes to corrupted computations, the error rate is difficult to estimate and may not be constant. We may know the rate of SEUs per bit, but are less likely to know when and where an SEU would cause a computational error.

A measure of memory vulnerability that accounts for both space and time is mentioned in [97]. A measure called mean-work-to-failure addresses the temporal and spatial extent of vulnerability, though it requires knowledge of the SEU rate [82]. A spatially-

weighted extension of the AVF metric was proposed [21], as was a temporally-weighted extension [13], but neither captures both dimensions.

3.4.1 Fault Model and Soft Error Model

The fault model used here consists of an SEU for which the value of a single state element in the machine (e.g., a latch) is flipped at a particular time. In our experiments, we assume that memories are covered by error detection and therefore we focus on latches/flip-flops. Latches are only vulnerable to SEUs during a certain timing window [93]; we assume in this study that this timing consideration is built in to the SEU rate. We focus on computations incurring no more than one SEU. Lastly, we do not consider in this study either single event transients in combinational logic or single event multi-bit upsets.

The soft errors of interest here are those that are undetectable by the system and cause incorrect computational results. Because such errors are disguised from any error detection method and yet have harmful effects, we describe them as *insidious soft errors* (ISEs). An ISE is initiated by an SEU and eventually causes an incorrect result to be returned, i.e., a computational failure. In the literature, the aforementioned ISE scenario is sometimes referred to as a silent data corruption (SDC) error. Silent data corruption can occur in a variety of domains (memory, storage, processing) due to a variety of fault mechanisms (software bugs, permanent hardware faults, transient faults, etc.). ISE refers specifically to soft errors in computations.

3.4.2 Proposed Metric

A metric is needed that captures the extent of logical vulnerability in the computations performed by a certain design. In other words, given the existence of physical

faults such as SEUs, what is the logical vulnerability to soft error propagation and corruption of computational results? We treat a computation performed using a configuration $Config_i$ as having N associated spatial resources, each containing one or more state elements. Resource n has a spatial weight $w_s(n)$ which represents its relative rate of SEUs. For resources consisting of the same type of state elements (e.g., the same flip-flop design), and under the single fault assumption, the weight is simply the number of state elements. For now we assume that the physical elements in a given region exhibit uniform fault susceptibility; in later chapters we account for local variations. The computation also is assumed to have K temporal intervals, with each interval k having weight $w_t(k)$ indicating its duration. In the simplest case, all weights are uniform, for instance when the intervals represent clock periods. For each resource n at each interval k, some fraction of possible SEUs at (n,k) would in fact initiate an ISE and lead to a corrupted result; this fraction is denoted the vulnerability fraction $VF(n,k)$. The *vulnerability over space and time* (VST) is the sum of weighted vulnerabilities in the computation:

$$VST(Config_i) = \sum_{n=1}^{N} \sum_{k=1}^{K} VF(n,k) \times w_s(n) \times w_t(k). \tag{3.1}$$

By aggregating vulnerability over an entire computation, we obtain a single VST value which represents the total logical vulnerability intrinsic to the computations performed using configuration $Config_i$, independent of the SEU rate. The VST values differ across designs that use different spatial or temporal resources or that have a different logical structure. For instance, designs with extensive error correction will tend to have low VST.

VST is expressed in units of "bit·s" which captures both the spatial and temporal dimensions. The use of "bit" here refers to information held by the state elements which in turn have some spatial exposure to SEUs. As an example, assume a computation has 10^6 state elements with weight 1 (i.e., one bit of state), and those elements are fully vulnerable for 10^6 periods of weight 10^{-9} s. Summing up the vulnerabilities gives a VST of 10^6 bits $\times$ 10^6 $\times$ 10^{-9} s = 10^3 bit·s.

The usefulness of VST will be illustrated in simulation experiments described in the next section. Subsequently, the metric will be used as a parameter in cost functions in three case studies of netlist-level PAC.

3.5 Experimental Results

A problem formulation is given for netlist-level PAC with regional variation. Three applications are introduced, and results are presented for VST experiments and for netlist-level PAC.

3.5.1 Problem Formulation

In defining the cost function, we consider three parameters: area, throughput, and soft error vulnerability. For an area estimate, we use the number of flip-flops used by a design. Throughput is simply the number of operations per second. For soft error vulnerability, we use the probability of a soft error affecting a computational result. This probability is a function of the transient fault rate and the VST value. While VST can often be estimated through pre-manufacturing simulations as in the previous section, the fault rate will often need to be determined through in-system characterization. This is because the fault rate can depend on regional substrate variations (e.g., in threshold voltages), op-

erating conditions such as the supply voltage (which can vary regionally), and a dynamic noise environment that can involve changes in particle flux or in the mix of particle energies. As one example, spacecraft encountering solar ejection events [35] or passing through the South Atlantic Anomaly [108] can temporarily experience upset rates 2–4 orders of magnitude higher than normal, as depicted in Figure 3.2.

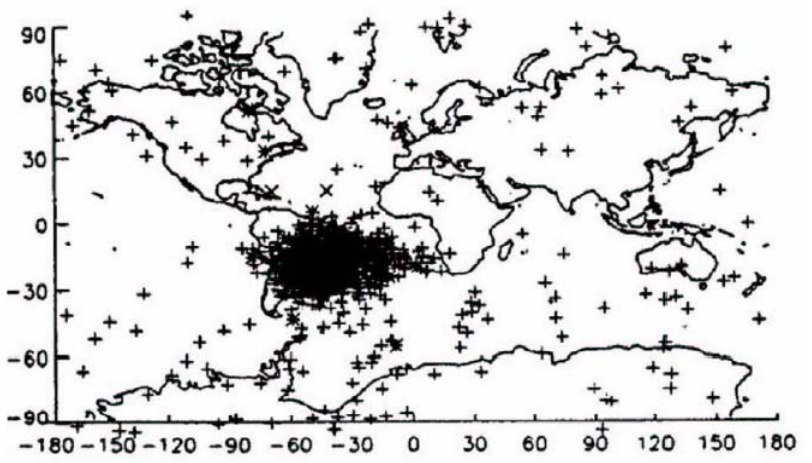

Figure 3.2: Large variation in radiation-induced SEUs occurring in the NASA SAMPEX mission in low Earth orbit, due to the South Atlantic Anomaly [108]

The average fault rate associated with region (x,y) at time t will be denoted $\lambda(x,y,t)$. This accounts for the time-varying noise environment, as well as the inherent fault susceptibility of the region at (x,y). This fault rate can be determined via introspection using methods such as those in Chapter 4. Once the fault rate is learned, it can be combined with VST to determine the probability of a computational error. When certain conditions hold, the probability is just the product of the two physical and logical factors. The conditions are the following: a uniform fault rate within the region of interest and throughout the computation, no more than a single fault per computation, and a fault rate that is independent of the data (0 or 1). If the fault rate is data-dependent, the equation can be ex-

tended by separating out the 0 and 1 cases. Provided the above conditions hold, the probability of a computational error is estimated as:

$$P_{err} = \lambda(x, y, t) \times VST. \tag{3.2}$$

As an example, assume a computation has an SEU rate of $\lambda = 2.8\times10^{-15}$/bit·s (equal to .01 FIT/bit) and $VST(Config_i) = 10^{13}$ bit·s. The probability of a result error would be:

$$P_{err} = 2.8 \times 10^{-15}/\text{bit} \cdot \text{s} \times 10^{13}\ \text{bit} \cdot \text{s} = 0.03.$$

Now that all three necessary system parameters have been specified, we can define the cost function. For these case studies, we use a simple linear weighted sum of the area, throughput, and soft error parameters:

$$Cost(Config_i, Sub(x, y)) = w_{area} \times area + w_{thr} \times thr + w_{cv} \times \lambda(x, y, t) \times VST. \tag{3.3}$$

3.5.2 Applications

PAC at the netlist level will be demonstrated on three sample applications. In each case, we generated alternative netlists (expressed in Verilog) using the Xilinx CORE Generator 9.2i tool.

CORDIC Algorithm. The COordinate Rotation DIgital Computer (CORDIC) is a long-standing algorithm for solving trigonometric, hyperbolic, and square root equations [120]. We compare two different CORDIC designs – a "word serial" design and a streaming design. Each is based on the Xilinx CORDIC v3.0 core [120], targeting the Virtex-II Pro FPGA platform. The word serial design is capable of handling only a single computation at a time, with a latency of 20 cycles and repeat rate of 19 cycles (consecu-

tive computations overlap by one cycle). The streaming design also has a latency of 20 cycles, but is capable of handling 20 simultaneous computations, with one completing each cycle. Both designs use maximum pipelining and have 16-bit inputs and output. The designs are configured to perform vector rotation computations.

Floating-point Addition. Floating-point is a number representation supporting wide ranges, and floating-point arithmetic is used in many important computations such as scientific applications. We generated three different designs of a double-precision floating-point adder, with varying amounts of register pipelining. The adders are all based on the Xilinx Floating-Point Operator v3.0 [119] and target the Virtex-5 platform. We generated adders with the minimum amount of pipelining allowed (2 stages of registers), an intermediate amount (7), and the maximum amount (12).

Fast Fourier Transform. The Fast Fourier Transform (FFT) is an efficient method of computing the Discrete Fourier Transform, and is very popular for digital signal processing [121]. We generated three different hardware designs of the FFT, based on the Xilinx FFT v5.0 core [121]. Each design is capable of computing a complex, one-dimensional fixed-point FFT with a programmable size of up to 64K points. The designs are labeled according to the architecture used: Radix-2 Lite, Radix-2, and Radix-4. Each computes the FFT by making a number of passes through the data. A pass can be thought of as a computational "phase." The number of phases is defined by the architecture to be $\log_r N$, where r is the radix and N is the number of points in the signal.

3.5.3 Computational Vulnerability

We conducted vulnerability experiments that demonstrate how the proposed VST metric can be used for evaluating different netlist designs of a given function. We mod-

eled SEU faults in flip-flops as mentioned earlier. We did not model SEUs in memory cells such as on-chip RAM or FPGA configuration cells; these tend to be detectable and thus not a contributor to ISE.

The gate-level Verilog netlists of each design were instrumented with fault injection circuitry such that an SEU could be injected at any flip-flop and clock cycle. Care was taken to ensure that SEUs could occur even in flip-flops with a de-asserted clock enable, just as they can in actual hardware.

We performed fault simulation to find a complete set of vulnerabilities associated with a computation, and used those to calculate VST. In the FFT case, we performed exhaustive fault simulation on 32-point computations and found the VST associated with each phase, allowing us to project the VST for larger-scale computations. For comparison purposes, we also determined the conventional vulnerability fraction (VF) associated with each design.

CORDIC computations. We performed exhaustive fault simulation across ten computations with random input values, and determined the average VST associated with the two designs. The results shown in Figure 3.3 indicate that computations performed by the streaming design have a lower VST and thus lower vulnerability. Experimental data is given in Table 3.1.

Table 3.1: VST of CORDIC computations

Design	SEU faults injected	Insidious soft errors (ISEs)	Ave. VST per computation (bit·s)
Word serial	107000	15120	1.14×10^{-4}
Streaming	268200	11931	7.02×10^{-5}

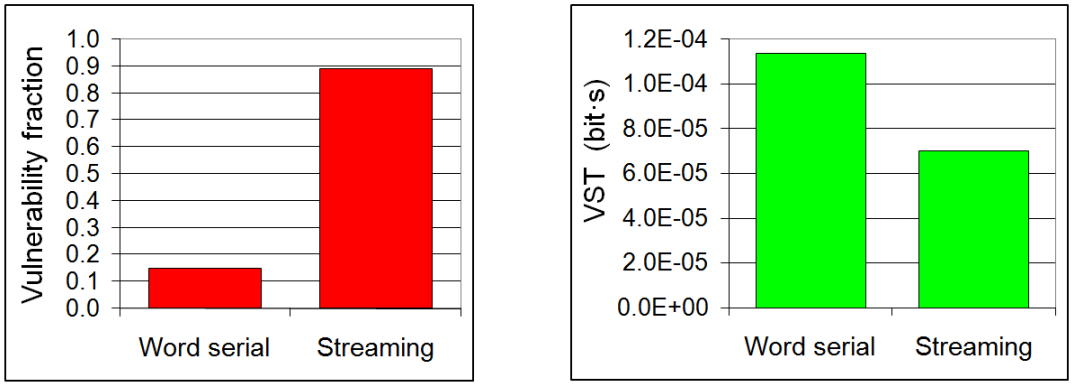

Figure 3.3: Traditional vulnerability fraction of CORDIC designs (left); vulnerability for CORDIC computations using the VST metric (right)

Note that the standard VF metric is not suitable for this type of comparison. The streaming architecture has a high VF, which can simply be an indication that the design is highly efficient, with almost all computational resources being highly utilized. The throughput is 19 times higher than the word serial case, and the clock period is shorter (shown below in Table 3.3). Thus the computations performed by the streaming design are significantly less vulnerable. The word serial design can perform only a single computation at a time, causing many of the state elements (such as input and output flip-flops) to remain under-utilized and leading to a low VF. Moreover, the reliance on a single shift-add circuit requires data to be fed back, which can force a longer clock period and actually increase the vulnerability per computation.

Floating-Point Addition. For each design we performed exhaustive fault simulation to determine the associated VST. The input patterns consisted of ten pairs of random floating-point numbers. The VST values for a given design showed very little dependence on the input values, differing by less than 1% in all cases. We determined the average VST of a floating-point addition on each design. This average VST can be thought of as the intrinsic vulnerability of the associated computation.

As can be seen in Figure 3.4, computations performed on the minimum-pipelined design have the least vulnerability (lowest VST), and those on the maximum-pipelined design have the most vulnerability. Experimental data is shown in Table 3.2.

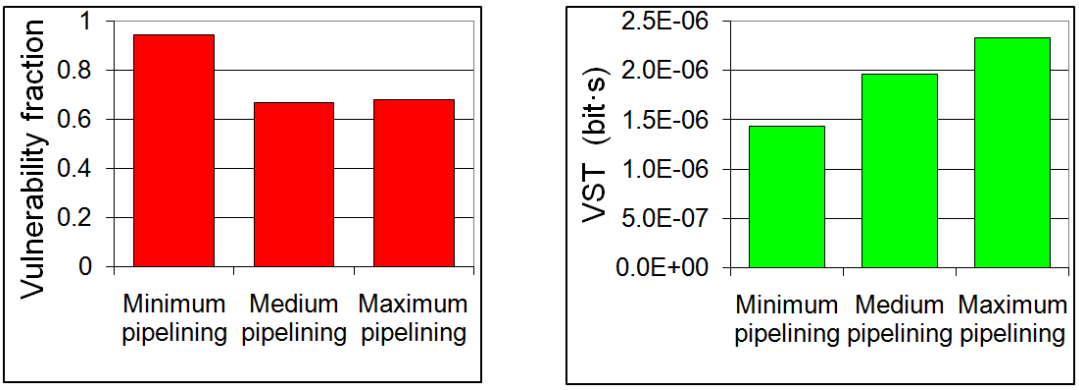

Figure 3.4: Traditional vulnerability fraction of floating-point adders (left); vulnerability for floating-point additions using the VST metric (right)

Table 3.2: VST of floating-point additions

Level of pipelining	SEU faults injected	Insidious soft errors (ISEs)	Ave. VST per computation (bit·s)
Minimum	2940	1388	1.43×10^{-6}
Medium	51240	4878	1.96×10^{-6}
Maximum	130200	7387	2.33×10^{-6}

It is important to note that VF gives a very different and potentially misleading picture of vulnerability, as can be seen in Figure 3.4. The minimum-pipelined design has the highest VF, but its computations are in fact the least vulnerable. Moreover, whereas the CORDIC experiment favors the largest, highest performing design, this study favors the smallest, lowest performing design. This illustrates that there is not always a simple relationship between the vulnerability of the computation and the size or performance of a design.

FFT. We used exhaustive fault simulation to determine vulnerability of a 32-point Radix-2 FFT. A portion of the VST map for this computation is shown in Figure 3.5; shaded cells indicate vulnerable bit-times. We also calculated VST values for each clock period to determine the dynamic nature of vulnerability, as shown in Figure 3.6. The VST for the entire computation is simply the area under the curve. Note that there are $\log_2 32 = 5$ compute phases. The dramatic swings in vulnerability were a surprising result. They arise from the flowing of state information back and forth between computational logic and on-chip memory during each phase. The data is protected from insidious soft errors when residing in memory, and potentially vulnerable when venturing into the exposed logic. The dynamics highlight the fact that, at least at fine time scales, the rate of insidious soft errors need not be anywhere near constant. Thus the routine assumption of a constant error rate needs to be carefully considered [114].

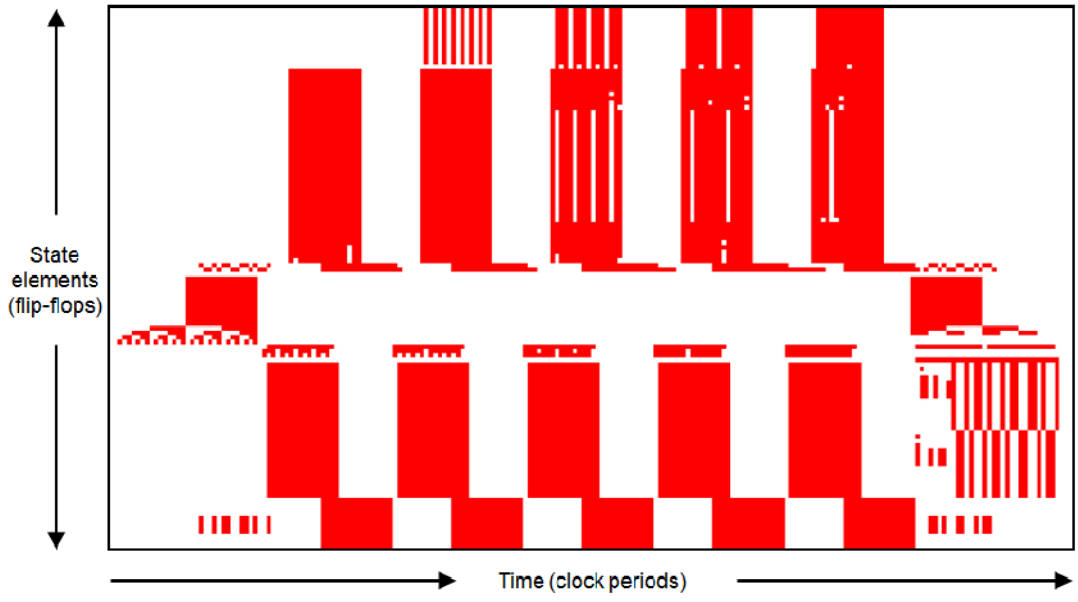

Figure 3.5: Portion of the vulnerability map for 32-point Radix-2 FFT

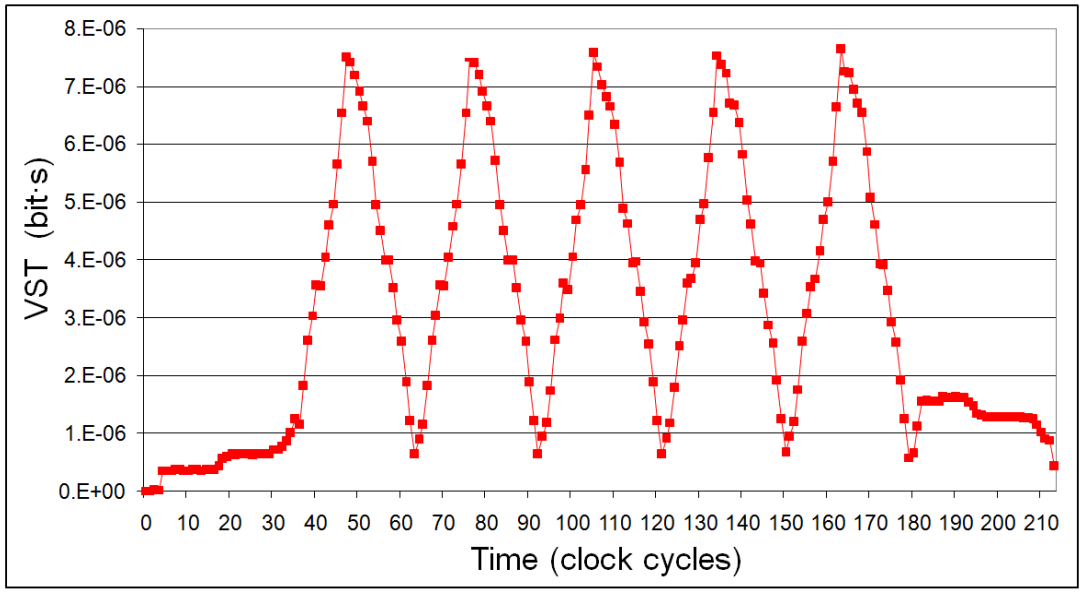

Figure 3.6: Dynamic behavior of VST for 32-point Radix-2 FFT

A general parameterized formula of vulnerability for these N-point FFT computations is given by:

$$VST = (VST \text{ per phase}) \times (\#\text{ of phases}) = A\, N \log_r N. \tag{3.4}$$

The coefficient A is design-specific and is based upon data gleaned from fault simulations of small-sized FFTs. For instance, it depends on the peak VST values, as well as the length of each compute phase, which in turn depends on the number and depth of the design's "butterfly engines." Note that this equation accounts for the total vulnerability of the compute phases; we did not study the vulnerability of the load or unload phase (e.g., the first or last 32 cycles in Figure 3.6), which tends to be similar for each design.

The VST values for large-sized FFT computations can be projected, as seen on a log-log scale in Figure 3.7. Computations performed on the Radix-2 Lite design are the most vulnerable, while those on Radix-4 are the least vulnerable. On a linear scale, the pronounced differences in VSTs are readily visible, as depicted in Figure 3.8. As with the

other experiments, measurements of VF do not capture the actual vulnerability relationships. The Radix-2 Lite design has the lowest VF, but the highest absolute vulnerability per computation.

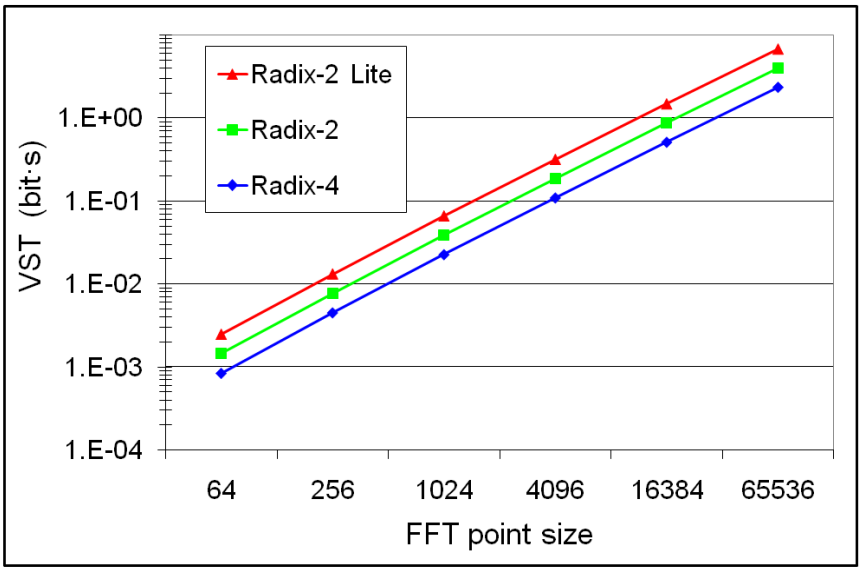

Figure 3.7: Projected VST for various FFT sizes

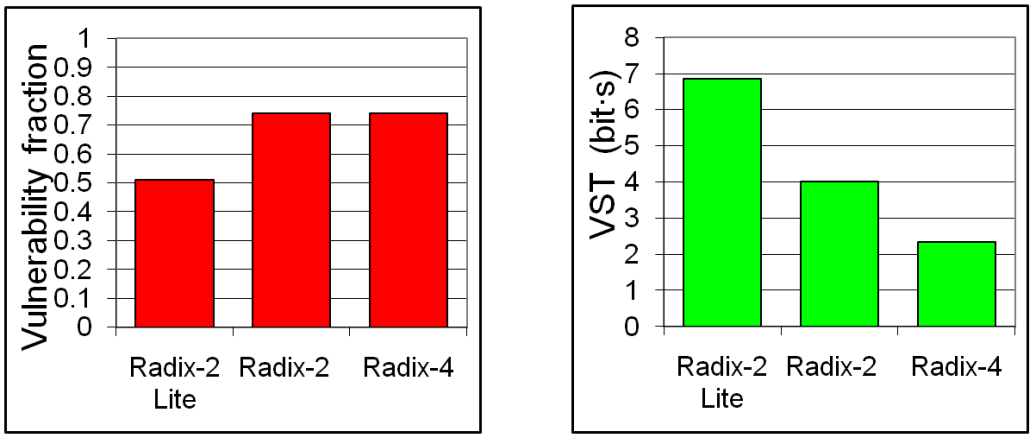

Figure 3.8: Traditional vulnerability fraction for 64K-point FFTs (left); projected vulnerability using the VST metric (right)

The experimental results indicate that the relationship between the vulnerability of a computation (as measured by VST) and other design characteristics is not a simple one.

In two experiments, higher throughput was associated with lower VST, but with floating-point additions the opposite was true; the addition of pipeline stages reduced the clock period but increased the number of state elements, leading to a net increase in VST. Nor was VST well-correlated with the number of state elements in the design. These findings stress the importance of performing characterization, and the need for treating reliability as an independent design consideration along with power consumption, area, and performance.

Another result of this study is a clear demonstration that the traditional vulnerability fraction metrics and the associated SER metric do not capture the absolute vulnerability per computation, and in fact can present a misleading picture. While highly-reliable designs have a low VF as one would expect, we have seen that inefficient designs *also* have a low VF. A prevalence of don't-care state bits does not improve the vulnerability of a computation but increases the circuit area and causes a deceptive decrease in the VF. There is a second and more subtle phenomenon at work as well. Inefficiencies in a design can lead to a longer clock period or additional cycles per computation, either of which forces vulnerable state information to be exposed for longer periods. These effects cause an increase in the absolute vulnerability that is not captured by VF metrics but is in fact captured by VST.

When evaluating architectures with different spatial and temporal extents, the VST metric can be used as a figure-of-merit. Results from the fault simulation experiments illustrate some of the inadequacies of the existing approaches and the efficacy of VST.

3.5.4 Netlist-Level PAC

Using the problem formulation, applications, and VST metric from Sections 3.5.1–3.5.3, we illustrate netlist-level PAC. We assign values to the weights in the cost function and show how cost varies with the fault rate. An example of a plausible fault rate used in research is 2.8×10^{-15}/bit·s, which corresponds to 0.01 FIT. The fault rate range considered here is from 10^{-16}/bit·s to slightly above 10^{-11}/bit·s.

CORDIC computations. Table 3.3 displays the static parameters for the two CORDIC designs: area, minimum clock period (which affects throughput), throughput, and VST.

Table 3.3: CORDIC design characteristics

Design	Area estimate (no. of flip-flops)	Minimum clock period (ns)	Throughput (ops/s)	Ave. VST per computation (bit·s)
Word serial	535	7.52	7M	1.14×10^{-4}
Streaming	1341	5.88	170M	7.02×10^{-5}

The cost functions for the configurations representing the two designs, $Config_{word}$ and $Config_{streaming}$, can be expressed as:

$$Cost\big(Config_{word}, Sub(x,y)\big) = w_{area} \times 535 + w_{thr} \times 7 \times 10^{6} + w_{cv} \times \lambda(x,y,t) \times 1.14 \times 10^{-4}, \quad (3.5)$$

$$Cost\left(Config_{streaming}, Sub(x,y)\right) = w_{area} \times 1341 + w_{thr} \times 1.7 \times 10^{8} + w_{cv} \times \lambda(x,y,t) \times 7.02 \times 10^{-5}. \quad (3.6)$$

The streaming design has a better VST, and thus for higher fault rates, its relative cost compared to the word serial design will improve. We illustrate this with an example.

Assume $w_{area} = 1$, $w_{throughput} = 10^7$, and $w_{cv} = 10^{21}$. Equations 3.4 and 3.5 can be solved to find the fault rate at which they are equal:

$$\lambda_{threshold} = 1.84 \times 10^{-14}/\text{bit} \cdot \text{s}.$$

It turns out that for any fault rate below this threshold, the word serial design has the lowest cost. Above the threshold, the streaming design has the lowest cost. The relationship for an entire range of fault rates is illustrated in Figure 3.9. Note that cost is shown on a log scale. At the lowest fault rates, word serial is a full 59% better, and at the highest, streaming is 38% better.

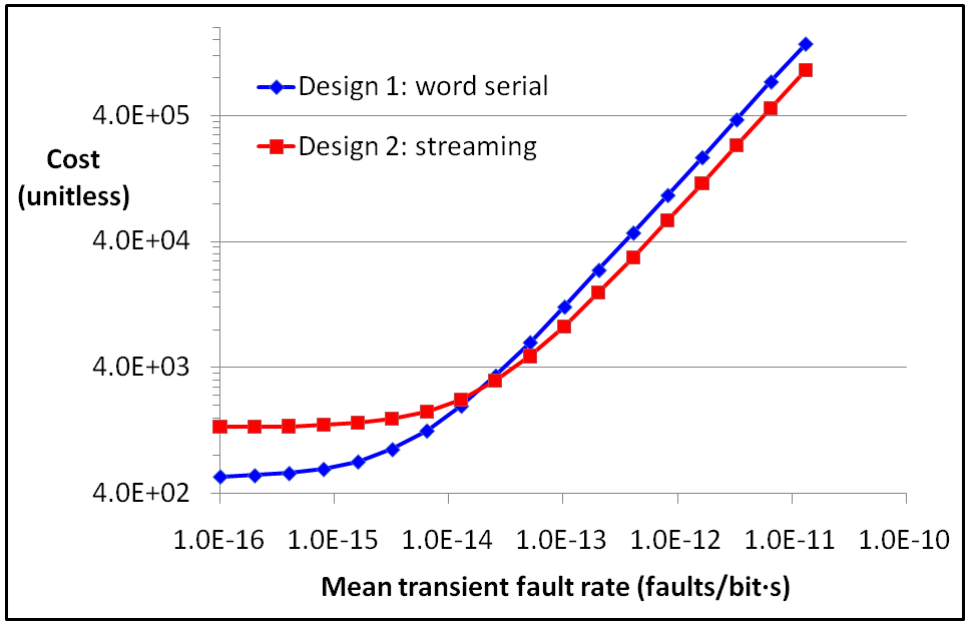

Figure 3.9: Cost of CORDIC designs vs. fault rate

Floating-Point Addition. Table 3.4 lists the static parameters of each design. Throughput is given in terms of floating-point operations per second (FLOPS).

Table 3.4: Floating-point adder characteristics

Level of pipelining	No. of pipeline stages	Area estimate (no. of flip-flops)	Minimum clock period (ns)	Throughput (FLOPS)	Ave. VST per computation (bit·s)
Minimum	2	147	10.31	97M	1.43×10^{-6}
Medium	7	732	4.02	249M	1.96×10^{-6}
Maximum	12	1085	3.16	316M	2.33×10^{-6}

As another example of relative costs changing with the dynamic fault rate, we assume $w_{area} = 1$, $w_{throughput} = 10^{12}$, and $w_{cv} = 10^{22}$. Results are shown in Figure 3.10 on a log scale. In solving the three cost functions, we find two thresholds. Between fault rates of 1.35×10^{-13} and 1.08×10^{-12}/bit·s, the medium pipelined design is the fittest by a small margin. Below 1.35×10^{-13}/bit·s, the maximum pipelined design is the fittest by up to 10%. Above 1.08×10^{-12}/bit·s, the minimum pipelined design is the fittest by up to 24%.

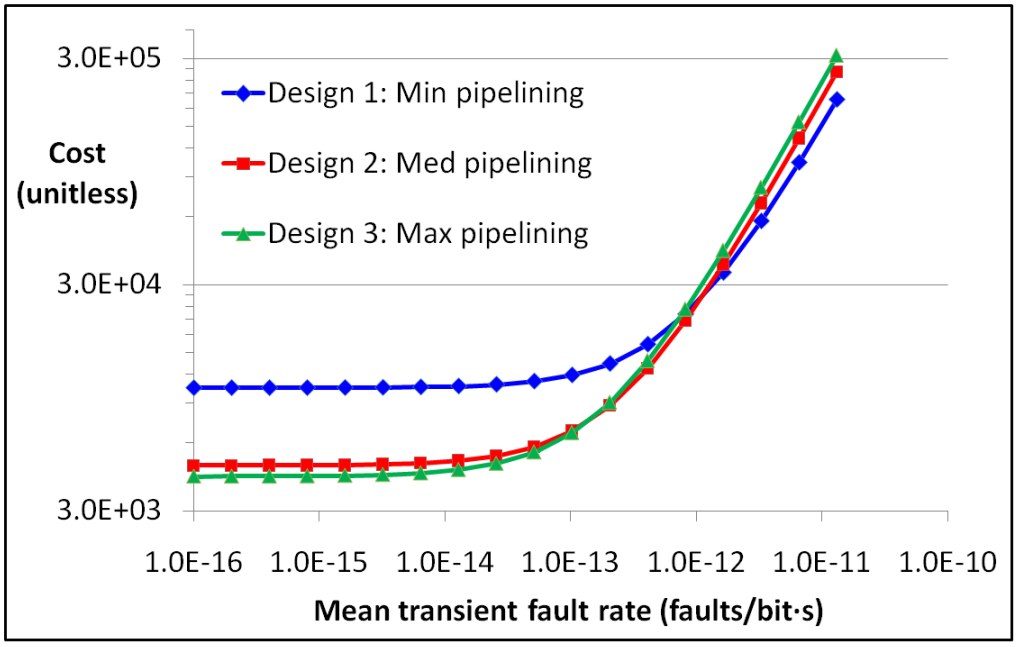

Figure 3.10: Cost of floating-point adder designs vs. fault rate

FFT. The static parameters of each design are listed in Table 3.5.

Table 3.5: FFT design characteristics

Design	Area estimate (no. of flip-flops)	Minimum clock period (ns)	No. of engines	Length of each compute phase (clock cycles)	Projected VST per 64K-point computation (bit·s)
Radix-2 Lite	1459	8.77	1	$(\log_2 N)+12$	6.86
Radix-2	1566	6.58	2	$\lceil 0.5 \log_2 N \rceil+13$	4.00
Radix-4	3715	6.45	4	$\lceil 0.25 \log_4 N \rceil+16$	2.33

As before, we select weights in order to provide an instructive example: $w_{area} = 1$, $w_{throughput} = 1$, and $w_{cv} = 10^{15}$. For a fault rate below 3.33×10^{-14}/bit·s, the maximum Radix-2 Lite design is the fittest. Above 1.07×10^{-12}/bit·s, the Radix-4 design is the fittest. Between those two thresholds, the Radix-2 design is the fittest. In the limit of high fault rates, the Radix-4 design has nearly half the cost of the next best alternative.

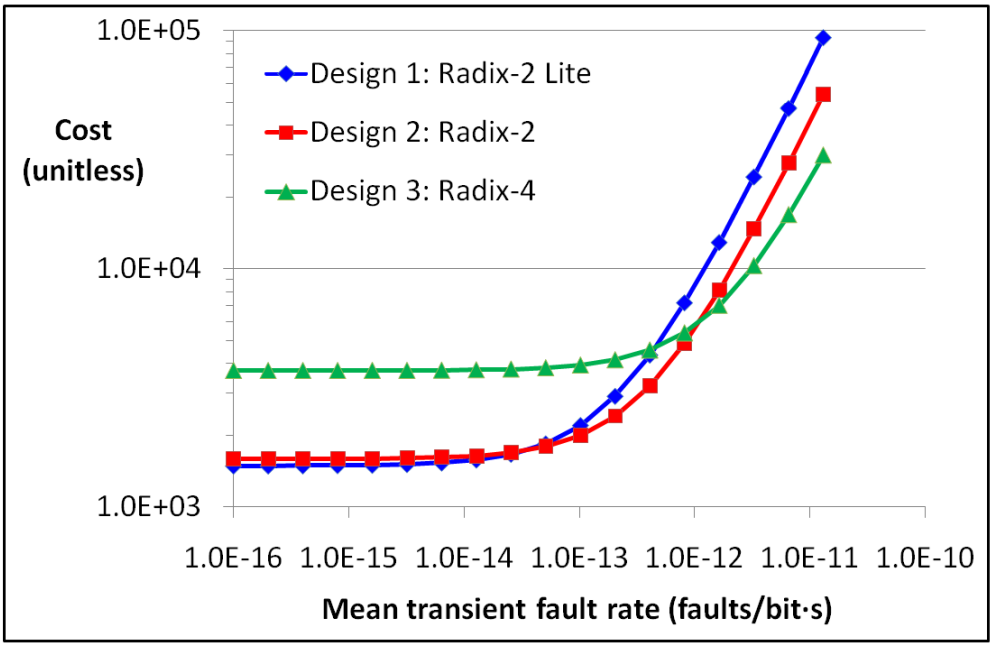

Figure 3.11: Cost of FFT designs vs. fault rate

In all three applications, significant reductions in cost are possible by dynamically identifying the fittest design for the given regional variations and fault environment. The table below shows the reductions possible by switching from the 2nd fittest to the fittest design, both for the low and high fault rate environments. Furthermore, it shows the cost reduction enabled by the fittest design relative to the average cost over all designs.

Table 3.6: Reduction in cost after selecting fittest design

Application	Cost reduction relative to 2nd fittest design ($\lambda = 10^{-16}$/bit·s)	Cost reduction relative to 2nd fittest design ($\lambda = 1.31\times10^{-11}$/ bit·s)	Cost reduction relative to average cost ($\lambda = 10^{-16}$/bit·s)	Cost reduction relative to average cost ($\lambda = 1.31\times10^{-11}$/ bit·s)
CORDIC	59%	38%	42%	24%
Floating-point adder	10%	24%	34%	23%
FFT	6%	45%	34%	49%

3.6 Summary

Reconfigurable systems have great potential not only to learn about their regional variations but to compensate automatically for them. Adaptive body bias, re-placement, and selection of alternative netlists are just three promising approaches. With an appropriate netlist-level soft error metric and its inclusion as a parameter in cost functions, systems can better trade off reliability and efficiency.

Having discussed PAC for regional variations in Chapters 2 and 3, we next move on to local, random variations in Chapters 4 and 5. Local variations are different in character and require novel methods of introspection and self-optimization.

CHAPTER 4

Introspection for Local Variations

In this chapter, we propose methods of introspection for local, random variations. In contrast to the regional variations covered in Chapters 2 and 3, local variations are so finely grained and uncorrelated that they can be very difficult to observe or keep track of. A case study is provided of variations in transient fault upsetability in latches. We establish limits on using a system's actual noise environment for self-characterization, and then propose in-system noise emulation as a much more feasible method. As a proof-of-concept, we demonstrate an FPGA-based system capable of injecting noise and uncovering previously hidden variations. Learning about local variations enables special PAC optimizations as discussed in Chapter 5. The two major portions of this chapter were published in [131] and [133].

4.1 Background

Delay, power, and leakage variations are widely recognized to be reducing system efficiency. The amount of 3σ variation in these parameters is reaching 60%, 72%, and 255%, respectively [52]. However, another threat is emerging and is much less understood. As seen in Chapter 1, technologies at the nanoscale are increasingly prone to local variation, and at the same time increasingly prone to noise. We see a strong need for un-

derstanding and mitigating the threat posed by this combination—variations and noise. While the threat is not well understood, it is clear that spreads in V_t and transistor gate length are impacting the minimum amounts of charge (Q_{crit}) that can upset logical states [9][34]. A recent simulation of flip-flop designs found large variations in Q_{crit} for individual instances of a given design, although it did not quantify the contribution made by local variations. Using a hardware-calibrated model of 65 nm CMOS transistors, the study found that even for the best-case flip-flop design, the total 3σ variation in Q_{crit} was as high as 20% for flip-flops in the 0 state, and 44% in the 1 state [73]. Most alarming of all, transient fault rates tend to have an exponential dependence on Q_{crit}; one model indicates that a 70% difference in Q_{crit} can correspond to a 15 times difference in fault rates [77].

Isolating and observing behavior at this extreme level of granularity is very difficult. Sensors such as those proposed in Chapter 2, whose measurements are correlated to the parameters of the surrounding region, are of little help with local, uncorrelated variations. New methods of local characterization are needed.

Related Work. Several works have considered local variations in the context of delay, including methods of characterizing FPGA delay variations [90][112] and for performing variation-aware re-placement [92]. These approaches are not applicable to local variations in upsetability. A method was recently proposed for characterizing extreme variations in resistances and threshold voltages for a future reconfigurable nanotechnology with 5 nm gate lengths [38]. Much related work has focused on hard defects, which might be thought of as a pathological type of local variation. Approaches to self-healing

via mapping and re-routing around defects have been suggested [62][105], but are not sufficient for identifying or compensating for variations in latches.

Little research has been published regarding local variations in transient fault upsetability. A simulation-based study of total variation in 65 nm flip-flop reliability appeared recently as mentioned above [73]. Some attempts have been made at on-line characterization of single event transients [80][87] and single event upsets [19], but not local variations. Some methods have been proposed for mapping out the marginal cells in an SRAM array and for preferentially matching an application to the appropriate cells/regions [79], but these are not applicable to logic. Testing with an external radiation source is sometimes used to estimate upsetability for a particular chip type. The noise sources used include alpha particle emitters, and beams of protons [55], heavy ions [4], neutrons, and light (i.e., a laser). Typically the mix of particles and energies is limited, making it difficult to achieve fidelity to actual noise environments. The noise locations generally cannot be finely controlled, so characterization of fine-grained variations is impossible. Empirical data on local variations in upsetability is thus severely lacking. In any case, radiation testing is nowhere near feasible for characterizing every component on every chip to be deployed.

4.2 Local Variations in Latch Reliability

In this section, we examine in more detail the phenomena of transient faults, latch upsets, and local variations. Transient faults in digital systems are one-time events that upset the state of a node and potentially lead to system-level soft errors. Faults can be triggered by particle radiation, coupling, thermal noise [85], and other mechanisms. Due

to local variations, each component can have a different inherent susceptibility to transient faults. This susceptibility property goes by many names such as "error cross-section" in the specific context of radiation-induced faults. We use the general term *upsetability*, and define it as the probability of a component incurring an upset in a specific noise environment. In many cases, upsetability can be thought of as a rate of upsets. Often we normalize upsetability values to a mean; for instance, a component with twice the mean number of upsets will have a value of 2.0. Note the difference between upsetability of hardware components and the more abstract vulnerability of logical soft error propagation; the latter was covered in Chapter 3.

This case study focuses on faults that occur in a specific type of component, a latch. Latches store temporary bits of information to enable computing of the desired function. Transient faults affecting latches change the state of the stored bit. Upsets occurring in a memory array or a static FPGA configuration bit can often be efficiently detected and corrected without any system-level failures. Upsets in latches, however, are more difficult to handle. This is because latches change state rapidly and at locations all across a die, so they cannot be protected with a low-overhead error correction scheme such as ECC. It is possible to design custom latches that are less susceptible to upsets [126], but doing so adds expense and prevents the use of standard field-programmable logic. Protection of latches thus often requires a high-overhead scheme such as triple modular redundancy (TMR). There is a need for low-cost mitigation methods that can be used instead of or in conjunction with TMR.

An example of a CMOS latch is shown in Figure 4.1. Latches are typically used in a master-slave pair that forms a flip-flop. Upsets can occur when a latch is holding data

(closed). There are four main types of upsets corresponding to the master latch holding a 0 or 1, or the slave latch holding a 0 or 1. We refer to these as *m*0, *m*1, *s*0 and *s*1, respectively. The master and slave latches generally have opposite clock phases so only one is holding data at a time. Thus a flip-flop is susceptible to only one of the four upset types at any given time.

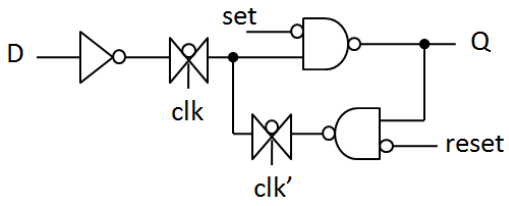

Figure 4.1: Example of a CMOS latch [96]. It can be used to form a master-slave flip-flop with asynchronous set and reset.

While some have predicted that upsets in combinational logic will become as prominent as upsets in latches, recent evidence from a 32 nm chip shows that latch upsets continue to dominate [37]. Furthermore, an ITRS working group has studied and simulated the failure rates for upcoming technologies, and has found that parametric failures of latches (e.g., latches that are too slow to be operational) are increasing much faster than failures of SRAM cells or combinational logic gates. In fact, they project that variation-induced failures of latches may increase by an astonishing 24 orders of magnitude between the 45 nm and 22 nm technology nodes, from 10^{-30} to 10^{-6} failures/latch (Figure DESN8 in [52]). On top of that, a new latch upset mechanism is looming on the technology horizon. As supply voltages edge closer to threshold voltages and CMOS scales down towards fundamental limits, latches will be upsetable by intrinsic thermal noise. One study suggests that by 2022 thermal upsets in latches will be very frequent at supply

voltages of 0.25 V; with threshold voltage variations, thermal upsets may occur even at 0.45 V [85].

4.3 System-Level Soft Error Model

We define a system as having a set of logical state bits *SB* and a set of physical flip-flops *FF*. We define an associated matrix **x** that specifies which logical state bits are placed at which flip-flop locations. The variable x_{ij} is 1 if state bit *i* is matched with flip-flop *j* and is 0 otherwise. Each state bit must be placed at exactly one location. Each state bit *i* also has a signal probability SP_i representing the probability that *i* is in the 1 state given the workload. Each flip-flop $j \in FF$ is prone to transient faults and has four upsetability parameters $m0_j$, $m1_j$, $s0_j$ and $s1_j$, corresponding to the rates of the four upset types.

The fault rate of a bit/flip-flop pair *ij* is essentially the fraction of time spent in a state multiplied by the fault rate associated with that state, summed over all states. We assume here that the clock duty cycle is 50% such that the master and slave latches are exposed for equal periods. The fault rate associated with a pair *ij* is then:

$$\lambda_{ij} = (m1_j + s1_j)\, SP_i + (m0_j + s0_j)\,(1 - SP_i). \tag{4.1}$$

Assuming that the error propagation probability VF is independent of the data state, the system-level soft error rate associated with pair *ij* can be estimated as:

$$SER_{ij} = \lambda_{ij} \times VF_i. \tag{4.2}$$

Assuming further that upsets occur at a single latch and that only a single soft error exists at any one time, the total system-level soft error rate can be estimated as:

$$SER = \sum_{ij} SER_{ij}. \tag{4.3}$$

4.4 In Situ Characterization

Variation-aware adaptation requires a feasible method of characterizing the upsetabilities. Radiation beam testing is out of the question for this task; it is not practical for individual latches to be characterized in this way, much less every latch on every chip that is to be deployed. Here we consider direct characterization via detection and counting of latch upsets *in situ*, meaning upsets occurring in the field due to a system's actual noise source. A potential advantage of this approach is high accuracy, due to the realistic noise environment. Furthermore, it may capture lifetime shifts in component parameters. However, this approach is only feasible if latch SEUs can be made to occur often enough to generate statistically significant data. Several methods of SEU acceleration will be considered next. We investigate using accelerated latch SEU counts as maximum likelihood estimators of the actual upsetabilities. For example, a latch incurring twice as many SEUs as another will be estimated to have twice the upsetability. The number of SEUs that must be observed to make this approach sufficiently accurate will be determined in Chapter 5.

Latches under test can be placed in a configuration that maximizes the rate of detectable SEUs. The latch clock can be turned off in order to maximize the window of vulnerability (WoV), which is the fraction of time that a latch is prone to an SEU that propagates beyond the current clock cycle. In normal operation this window is often open only

10% of the time due to timing masking [93]; here we increase the window to 100%. This un-clocked configuration also minimizes clock power.

An example of a simple in situ test procedure is shown in Figure 4.2. (a) Latches under test are first initialized to the desired state, and then left idle for an extended period to act as SEU detectors. (b) Any SEU that occurs leaves a latch in an error state indefinitely. (c) The latch states are periodically read out and monitored by a controller, and evidence of SEU events is recorded in individual SEU counts maintained in SEU-protected memory. The latches are then re-initialized and the process is repeated.

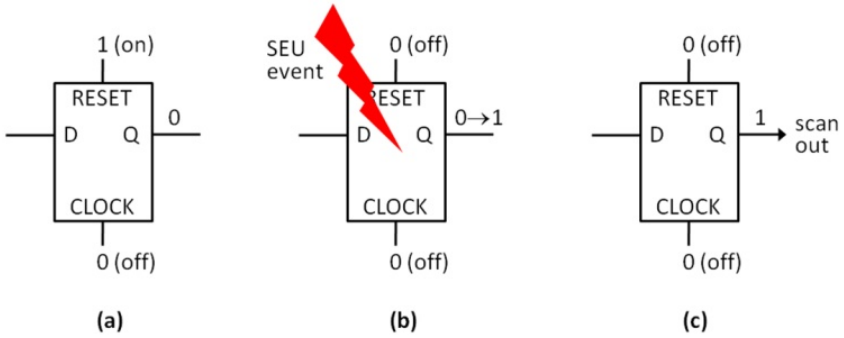

Figure 4.2: Example of on-line SEU characterization of level-sensitive latches

Note that this procedure can typically be implemented on a reconfigurable platform without requiring any special circuitry. For instance, with the Virtex FPGA family, latches can be initialized with either an asynchronous set/reset or by re-loading the bit-stream. The latch states can be read out in the background during system operation via the Virtex capture and readback feature.

One in situ test scenario is to characterize all unused latches opportunistically during system operation. For instance, a design may require 70% utilization of latches, allowing the remaining 30% to be characterized without impacting system availability. If neces-

sary, user logic could be swapped from location to location such that all latches would eventually undergo testing; similar schemes have been devised to test for permanent faults [36][101].

An alternative scenario is to perform SEU testing of all latches simultaneously during system down time. This nonconcurrent approach is sometimes the simplest and quickest way to characterize the entire platform. A particular advantage is the greater freedom to set the operating conditions. By lowering the operating voltage, the Q_{crit} values can be decreased and thus the SEU rates can be significantly accelerated. (The supply voltage must not be so low as to significantly shift the relative upsetabilities.) In [42], lowering the supply voltage to 0.4 V is shown to increase the latch SEU rates by one to two orders of magnitude.

Testing may be further accelerated by leveraging periods of high radiation. As mentioned in Chapter 3, spacecraft encountering solar ejection events or passing through the South Atlantic Anomaly can experience upset rates up to 10,000 times higher than normal [35]. Such periods may be unsafe for normal system operation, but they could provide a useful source of SEUs for accelerated testing.

4.5 Selective In Situ Characterization

Even with the above methods of acceleration, there is the complication of characterizing latches in all four permutations of data and clock states. The master and slave latches cannot necessarily be tested simultaneously; some reconfigurable platforms do not support both clocks being turned off at the same time, and thus only one type of latch is susceptible to SEUs at any time. This implies that at least four separate test campaigns

(two for master latches and two for slave latches) would be needed in order to fully characterize all latches. Most problematically, the SEU rate in some of these four configurations may be much lower than in others, forcing much longer test times.

Here we consider whether an in situ approach can be made more feasible via *selective* characterization. We calculated the relative SEU contributions of the four states using typical values for the various parameters. First we note that latch SEU rates are often heavily dependent on the data state of the latch. The latches in the Actel ProASIC FPGA are ten times more susceptible when holding a 1 bit than when holding a 0 [5]. Note also that latches are prone to upsets only for particles above a certain energy (measured in linear energy transfer), and that this energy threshold may depend on the data state. In the case of the Xilinx Virtex-4QV, the threshold in the 0 state is 1.5 MeV-cm^2/mg, but the threshold in the 1 state is only 0.5 MeV-cm^2/mg [4]. Radiation flux tends to be dramatically higher at lower energies, so even slight differences in threshold can lead to large differences in fault rates. In any case, we used the flip-flop study performed in [42] as a guide and assumed a 0–1 bias of 4.0.

Next, we modeled the difference in raw SEU rates between master and slave latches. Often master latches have higher raw SEU rates, for instance due to having lower drive strengths and lower Q_{crit}. Four of the five flip-flop designs studied in [42] have a significantly higher rate for the master latch; the ratios are in the range of three to ten, and we assume here a master-slave bias of four. Lastly, we chose typical values for the window of vulnerability. The WoV tends to be significantly higher for master latches; slave latches are susceptible to SEUs only during the second phase of a clock cycle when it

may be too late for the SEU to be captured by downstream logic. We set the master WoV to 0.25 and the slave WoV to 0.04, based on data from [93].

Combining the above factors, we determined the relative contributions of the four SEU types to the total number of SEUs that propagate to the next clock cycle. We found that with these parameter values, a full 77% of all propagated SEUs originate from a master latch in the more susceptible data state *A*. (State *A* can be a 0 or a 1 depending on the technology and latch design.) Some 19% come from a master latch in the less susceptible data state *B*. Only 3% of all propagated SEUs come from a slave latch in state *A*, and a mere 1% from a slave latch in state *B*. These estimates are summarized in Table 4.1. The rate of propagating SEUs shown in the fourth column is simply the raw SEU rate multiplied by the mean WoV.

Table 4.1: Example of typical SEU contributions

Scenario	Normalized SEU rate	Mean WoV	Rate of propagated SEUs	Fraction of propagated SEUs
Master latch in state *A*	1.0	0.25	0.25	**77%**
Master latch in state *B*	0.25	0.25	0.0625	**19%**
Slave latch in state *A*	0.25	0.04	0.01	**3%**
Slave latch in state *B*	0.0625	0.04	0.0025	**1%**

We find it surprising that, under some typical parameter settings, the vast majority of system-level soft errors can be traced back to a single latch type (master latches) and a single data state. In fact, due to differences in Q_{crit} and windows of vulnerability, 96% of all system-level errors in our study originate from master latches. For applications with a large imbalance of this kind, the SER can be lowered by optimizing the master latch case. In some systems, this can be accomplished simply by lowering the clock duty cycle.

Given their overwhelming importance, it may be possible to use a simplified approach to in situ characterization with a focus on master latches in state *A*. This reduced focus would greatly improve the test time; instead of four tests of varying lengths (lower SEU rates require longer test times), only a single test campaign would be performed and at the highest fault rate. This would also reduce the amount of characterization data that must be maintained, and enable a simplified self-optimization strategy that will be evaluated in the next chapter.

4.6 Self-Test

Given the limitations of in situ characterization established in the previous section, can a system feasibly characterize its own local variations in component reliability? Are such variations significant? In this section, we introduce a method of self-test for local variations in reliability using on-chip noise injection. We furthermore present the results of hardware experiments showing a large spread in latch reliability on two 65 nm FPGAs.

4.6.1 Fault Models

As discussed previously, digital circuits are prone to transient faults caused by radiation and other noise sources such as thermal noise. In particular, latches are high-performance storage cells prone to SEUs and SETs. As an empirical case study, we consider the latches in currently available (65 nm) FPGA chips. These "user" latches are employed directly in application logic, as distinct from the configuration cells used to hold the static configuration data. Prior work suggests that in a 65 nm process, SET pulses have widths of roughly 400–900 ps [9]. While FPGA vendors have tried to de-emphasize the importance of SETs and SEUs in latches, the problem is real and growing.

For instance, as a countermeasure, Xilinx is hoping to produce a special 65 nm chip series in 2011 called the Virtex-5QV. This platform would include special dual-node latches less prone to SEUs, and "filters" on all latch inputs to remove SET pulses of up to 800 ps in width. The drawbacks of this platform include extremely high cost (likely in the tens of thousands of dollars), and a lag of one manufacturing generation behind the leading edge of reconfigurable platforms. For these reasons, the vast majority of applications of reconfigurable technology will still be threatened by SETs and SEUs.

4.6.2 Method

We now address the question of how to characterize local variations in reliability at low cost. The basic approach we propose is in-system self-test using an on-chip "noise emulator" to test upsetability with respect to various sources of noise. Synthetic noise is generated on-chip and injected into a group of latches in the form of a very short pulse on one of the common signal inputs. Each latch will either be upset or not upset by the applied noise pulse. Through a series of trials, the inherent upsetability of each latch with respect to the noise of interest can be characterized. Systems need only perform the self-test procedure occasionally, or in some cases just once, to match the time scale of significant shifts in upsetability. For instance, upsetability values may shift over a period of weeks due to aging effects.

We now describe a method of noise emulation, and a method of calibrating the noise levels. We inject noise via the latches' asynchronous set and reset lines. The faults being modeled with this type of noise are twofold—SET pulses on the set and reset lines themselves, and (indirectly) SEUs. Detailed transistor-level and electrical models of the latch design are unavailable (they are usually vendor-proprietary), thus we can only indirectly

estimate upsetability with respect to SEUs. By injecting noise we aim to learn something about the stability and static noise margin of the latch, which in turn can provide some information about upsetability to radiation-induced SEUs that hit a latch directly, and thermal noise. If necessary, noise can be injected via the clock or clock enable line to gain additional information.

The proposed noise emulator must be able to generate very short pulses with fine resolution. It can be implemented using conventional reconfigurable platforms in at least three ways. One approach would be to use two phase-locked loops and the Vernier principle; this provides a resolution of 35 ps on a Xilinx Virtex-5 FPGA [23] but requires a long synchronization time before each pulse. A second method would be to use a special-purpose delay line such as an IODELAY block on the Virtex-5; this provides a resolution of 78 ps [95]. We propose a pulse generator based on a carry-chain delay line. It essentially acts as a digital-to-time converter, which takes an input value and creates a pulse of a certain width. The design is simple and compact, relying on the carry logic to control the pulse width with resolution better than 50 ps. Such carry logic is available on a variety of FPGA platforms. An example of a carry-chain delay line in a different context can be seen in [6]. A portion of our pulse generator chain is shown in Figure 4.3. Initially all of the level-sensitive latches are in the 0 state. When the trigger signal arrives, all latches open and the latch outputs begin to rise simultaneously. The "carry in" signal propagates through the carry-chain, forcing each latch to 0 after a predictable delay. Each resulting pulse has a slightly different width. One signal is selected at a multiplexer (mux) to act as the noise source that is driven on to either the set or reset signal.

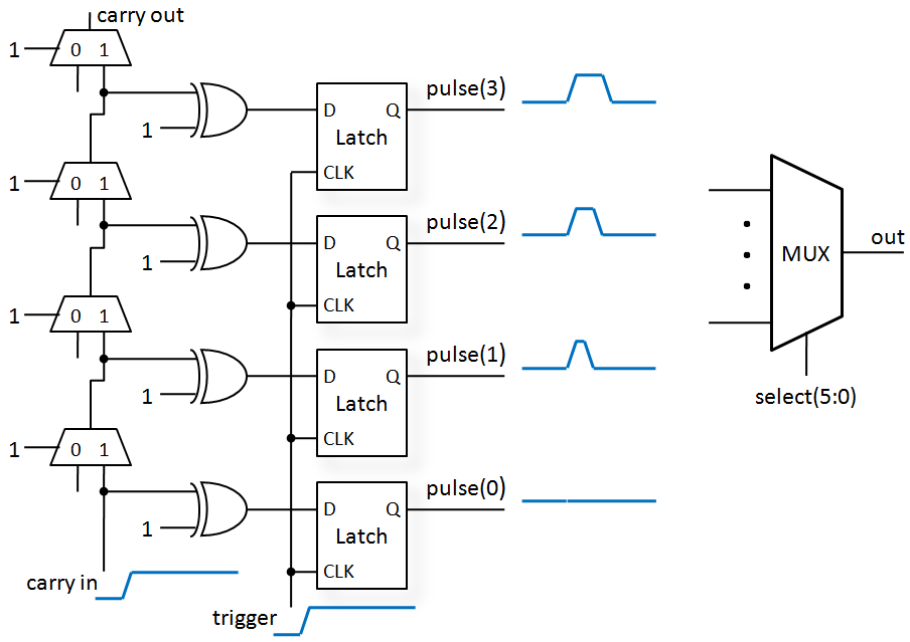

Figure 4.3: Portion of pulse generator circuit

In addition to the pulse generator, there is a need for an effective noise distribution system. With recent FPGAs, the noise can be distributed using built-in "clock" buffers and dedicated interconnect networks, both of which can be either global or regional. Such resources minimize pulse distortion and rise/fall delays. The buffers, distribution networks, and local interconnections however do exhibit variation themselves. Pulses traversing different paths can be subtly narrowed or widened differently, thereby skewing the amount of noise energy delivered to different locations. Thus, a key challenge is how to enable fair comparisons of upsetability at different locations. One important observation is that flip-flops are inherently arranged in clusters (small groups of logic), and the flip-flops within a cluster can be driven with the same control line. A very similar amount of noise can be delivered to the flip-flops within a given cluster. For this reason, we focus on characterizing *intra-cluster variations*. Comparisons across different clus-

ters (i.e., inter-cluster) are more difficult. An experiment that validates our intra-cluster approach will be discussed in the results below.

The self-test process is controlled by a program running on a processor core. The program can calibrate the noise level, set up test conditions, trigger noise, and read out and process the results. The amount of noise can be calibrated for each cluster and for each of the four test cases by stepping through a range of pulse widths and finding the noise threshold at which latch upsets occur. Thus one can compensate for variations in the interconnect as well as for non-linearities in the carry-chain. In fact, variations can be leveraged to improve the noise emulation capability; multiple independent carry-chains can be implemented to provide a greater selection of pulse widths. To reduce simultaneous switching and transients in the voltage supply, only one cluster is calibrated at a time. The clusters not under test are initialized to a state from which they will not switch, e.g., 0 in the case of reset noise. The proposed self-test circuit implementable on reconfigurable platforms is sketched in Figure 4.4.

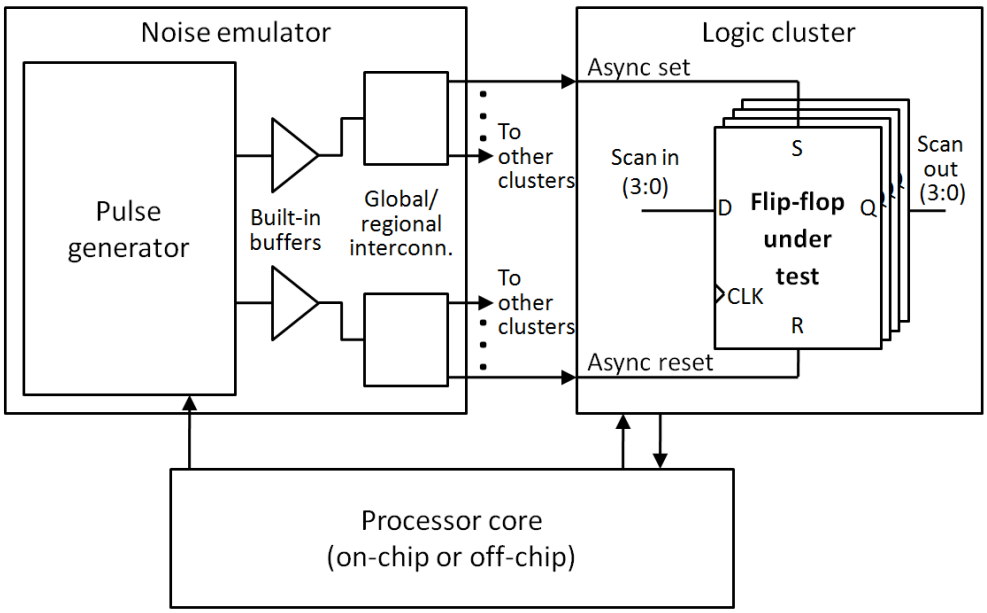

Figure 4.4: Proposed self-test circuit configuration

As a simple example of how upsetability is estimated, assume that self-test is performed for *s*0 upsetability of the four flip-flops at cluster coordinate (0,0). After hundreds of trials, the number of upsets occurring at the four flip-flops happens to be 300, 200, 400 and 100, and the mean upset count is 250. The relative *s*0 upsetability of the four flip-flops is estimated to be 1.2, 0.8, 1.6 and 0.4, respectively.

4.6.3 Experimental Results

We built an experimental system as a proof-of-concept of the proposed methods, and to answer two main questions: Is it feasible for a system to test itself for local, random latch variations? Can such variations be significant even on 65 nm FPGAs?

The system consists of an array of flip-flops under test, an on-chip noise emulator, and a MicroBlaze control processor. It was implemented on each of two XUPV5-LX110T boards containing a Xilinx Virtex-5 FPGA. The main focus of the study is a collection of 1,024 flip-flops arranged in a 32 × 32 array. The Virtex-5 architecture separates flip-flops into clusters of four (a Virtex-5 slice), connected to the same set and reset lines. We configured the flip-flops to include an asynchronous set and reset (Xilinx FDCPE primitives). The collection of 256 clusters is situated within a single clock region in the lower right corner of the chip. The flip-flop data inputs and outputs are daisy-chained together to form a single 32 × 32 shift register that can be accessed by the control processor using fast simplex links.

The noise emulator uses a pulse generator with eight independent carry-chains, each with eight stages. Pulses can be generated with widths as narrow as roughly 500 ps or as wide as 700 ps. The noise is distributed to the 256 clusters using two regional clock buffers and regional clock networks. The master and slave latches are treated here as black

boxes subject to noise; the internal transistor-level and electrical models are proprietary and unavailable to us. The experimental program is a standalone application written in C for the MicroBlaze using EDK 10.1; it fits within 64 KB of on-chip RAM. All experiments have been conducted using a 1 V supply voltage for the internal logic, and at ambient room temperature.

First, self-calibration of the noise levels was performed for each of the clusters and upset types; the total execution time was three minutes. The noise was calibrated to the lowest energy that allowed upsets to be detected in each of the four flip-flops. Clusters were calibrated individually in an effort to reduce any experimental effects (e.g., transient voltage droops) and thus improve accuracy; in large-scale systems, parallel calibration may be required in order to reduce overhead. An 8-bit calibration value was stored for each cluster and upset type, requiring 1 KB of memory. Then self-test was performed by applying noise to a single cluster at a time, reading out the results, and repeating the test 255 times for each cluster. The process was repeated for all four upset types, requiring a total time of five seconds. The entire self-test was conducted on each of the two FPGAs.

Our results show significant, local variations in upsetability. Figure 4.5 illustrates a map of $m1$ upsetability for the 256 clusters of flip-flops. Each cell in the image corresponds to a flip-flop, and each vertical group of four cells corresponds to a cluster. The value in each cell represents the ratio of the number of upsets to the average for the associated cluster. For instance, a ratio of two indicates that a latch incurred twice the average in that cluster. The primary purpose of the data is to characterize variations within each of the 256 clusters of four flip-flops; comparisons across different clusters are more difficult. We calculated the coefficients of variation for latch upsets within a cluster, as

shown in Figure 4.6. The coefficients range from 17% to 77% (3σ values of 51% to 231%). This amount of variation is generally higher than the ITRS projected levels for delay or power mentioned earlier, and in some cases even as high as leakage. This suggests that in some noise environments, faults in the many marginal components will be the dominant contributor to the overall fault rate, and thus there will be a large potential for upsetability-aware optimizations.

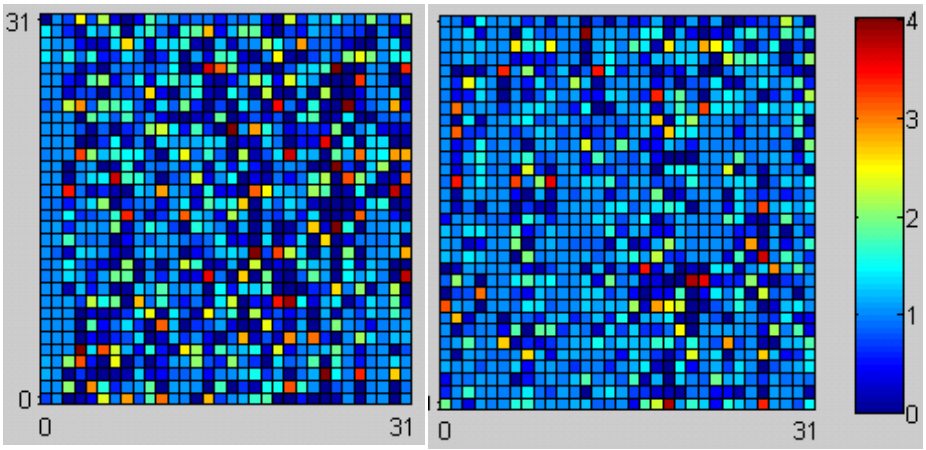

Figure 4.5: Local, intra-cluster variations in upsetability for chip 1 (left) and chip 2 (right). Shown is $m1$ upsetability relative to the local cluster average for 256 clusters of four flip-flops. Clusters span four cells vertically.

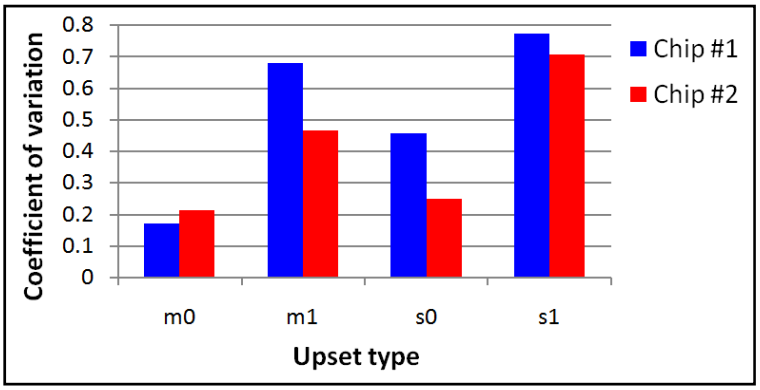

Figure 4.6: Coefficient of variation (σ/μ) for latch upsets within a cluster in the tested noise environment, averaged over 256 clusters

We saw no obvious systematic or regional correlations, providing support that the measured variations are indeed random by latch. We explicitly searched for a systematic bias based on flip-flop location in a cluster. The data indicates that upsets are nearly uniformly distributed across the four locations (A, B, C, D), as one would expect with local, uncorrelated variations. The relative distribution of 500,000 upsets across the four locations was (.252, .248, .253, .247) for chip 1 and (.253, .249, .252, .245) for chip 2.

Although the main focus is on intra-cluster variations for a given test case, some secondary conclusions can be drawn; we see more variation in slave latches than master latches, and far higher variation when a latch is in the 1 state than in the 0 state. The latter finding is consistent with a recent simulation of four types of 65 nm latches [73]. Note also that while the focus here is on flip-flops, the idea of injecting noise and estimating upsetability is also applicable to certain SRAM configuration cells. Many LUTs can be configured as shift register LUTs (SRLs), and noise can be injected through the clock or clock enable line, potentially uncovering variations in the constituent SRAM cells. We found a data bias using this method—a 1 bit in an SRL was much more likely than a 0 bit to flip due to clock noise.

4.7 Summary

Our results demonstrate that local, random variations can indeed be significant and can be uncovered via careful self-test. Whereas radiation testing is inapplicable, and in situ characterization during system operation is typically infeasible, on-chip noise emulation can quickly provide useful upsetability data. We found that the amount of variation in upsetability can be significant even on 65 nm FPGAs, with coefficients of variation of

up to 77% depending on the noise environment. The next questions, addressed in Chapter 5, are how to enable self-optimization in such a situation, and what kind of benefits can be expected on real circuits.

CHAPTER 5

Self-Optimization for Local Variations

This chapter covers self-optimization for local, random variations. Statistical simulations are used to estimate reliability improvements and find an appropriate size for the optimization neighborhood. We then use actual FPGA hardware introspection data from Chapter 4 to evaluate the use of re-packing and local re-placement on a set of benchmark circuits, and show how reliability can be improved via low-cost self-adaptation as well as assisted adaptation using a remote server. The two major portions of this chapter were published in [131] and [133].

5.1 Variation-Aware Re-Placement

Given the latch upsetability estimates made possible by introspection, a system can be reconfigured with a more effective mapping between logical state bits and physical latch instances. Performing a full variation-aware place and route cycle on-line would be quite costly in terms of computational and memory resources, especially if attempted on an embedded system. Instead we investigate the potential for incremental place and route [26].

We propose decomposing the logic array into a collection of virtual neighborhoods, and then optimizing the placement of state bits only within each neighborhood. This re-

duces the computational and memory burden significantly, and avoids most of the difficulty of routing and timing closure. Logic that has special placement constraints or that resides on a critical path can be made ineligible for re-placement. A simple example of such incremental place and route is illustrated in Figure 5.1.

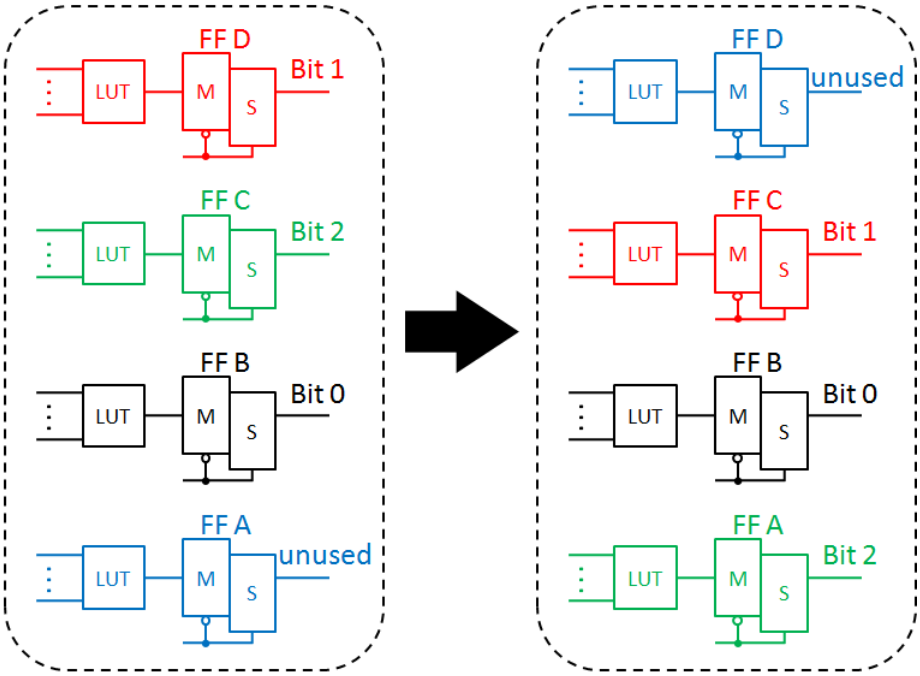

Figure 5.1: A cluster of logic in its original configuration with state bits (1,2,0) placed at master-slave flip-flops D,C,B (left); the same cluster after re-ordering of logic (right)

In order to evaluate different system configurations, estimates will be needed for the factors in (4.1)–(4.3). The upsetabilities can be estimated using one of the two methods presented in Chapter 4: in situ characterization, or self-test based on noise emulation. The logical parameters can generally be estimated at pre-manufacturing time. For instance, the VF factors can be estimated through statistical fault simulations and a careful study of the timing windows of vulnerability [43]. The signal probabilities can typically be found via logic simulation or, if necessary, on-line sampling in the field; with some

FPGAs on-line sampling of logic states can be performed transparently to a computation via non-intrusive scans.

We now investigate the following low-cost heuristic for finding improved placements. Physical flip-flop instances are ranked according to a single variable—the upsetability of the master latch in state *A* (discussed in Section 4.5). Logical state bits in the design are ranked by their weighted vulnerability fraction in state *A* (the probability of being in state *A* multiplied by *VF*). The worst-case unassigned state bit is paired with the best-case unassigned latch instance. Essentially this causes the threats at the physical and logical levels to be negatively correlated, in order to minimize the system-level soft error rate contribution. The process is repeated until all eligible state bits have been placed.

The main questions are whether such an approach can be effective enough in reducing the soft error threat and, if so, what is a proper neighborhood size? We address these questions in the next two sections.

5.2 Statistical Simulation Results

We created a statistical model of a computational system and conducted Monte Carlo simulations to determine the efficacy of the methods in question. The model contains 100,000 master-slave latch pairs, each latch having local variation in each of the four upsetability parameters. In this model only, the upsetabilities are assumed to be normally distributed. Note that in most noise environments the upsetabilities need not be normally distributed even if Q_{crit} is; this is because the energy of noise events is usually skewed toward lower energies. Only local, uncorrelated variation is modeled here; regional, spatially-correlated variation is assumed to be negligible within the small optimization

neighborhoods considered. The *SP* and *VF* variables for state bits are given uniform random distributions from 0.0 to 1.0. Unless otherwise stated, the remaining parameters are set to the values described in Section 4.5, with a neighborhood size of eight flip-flops and a logic utilization of 50%. For each experiment, we simulate latch upsetability characterization followed by variation-aware reconfiguration, and measure the impact on SER.

In the first experiment, we address the question of how much SEU sampling is required to adequately characterize relative latch upsetabilities. One published guideline suggests that 100 SEU events are needed to characterize average upsetability [1]; we are interested to find a similar empirical guideline for relative upsetability of individual latches. Given the dominance of one type of latch upset (master latches with data in the more susceptible state) as shown in Section 4.5, we seek to determine whether it is feasible to characterize and adapt to just one type of upsetability. We simulated successive amounts of SEU data collection followed by reconfiguration based on the upsetability estimates. The standard deviation for each type of upsetability was set to 20% of the mean. Thirty Monte Carlo trials were performed at each data point. The results are illustrated in Figure 5.2. We found that SER could be improved by up to 15%. Surprisingly, we found that most of the possible SER improvement can be realized even after a small number of SEUs are observed per latch. For instance, after an average of eight SEUs, the improvement in SER is over half of what could be achieved with perfect knowledge of the relative upsetabilities. The data is consistent across trials; the root-mean-squared-error is too small to be visible in the figure.

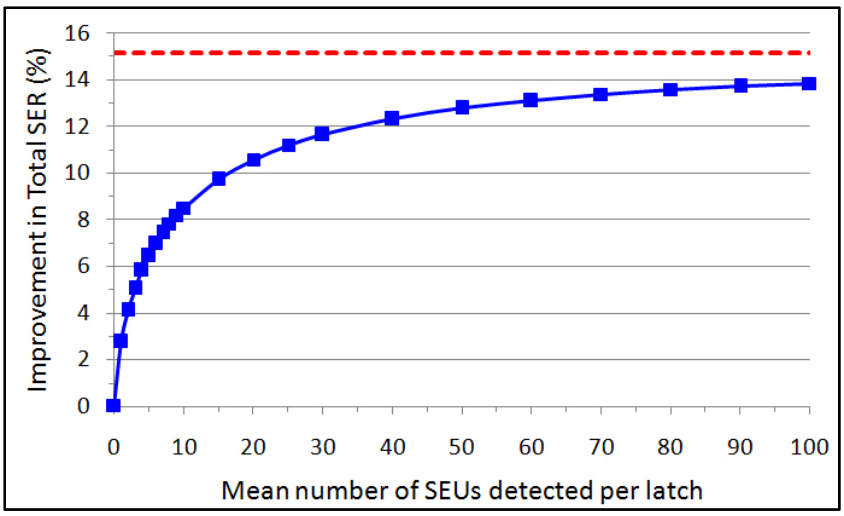

Figure 5.2: Improvement in SER vs. amount of SEU characterization

We measured the potential for SER improvements as a function of the amount of variation and as a function of the optimization neighborhood size. We simulated upsetability characterization (again limited to an average of ten SEUs per master latch), for three different amounts of variation: standard deviations equal to 10%, 20%, and 30% of the mean. These amounts of variation in upsetability are feasible given the spread in Q_{crit} modeled in [9][42][73]. Neighborhood sizes of 2, 4, 8, 16, and 32 flip-flops were simulated, with 30 Monte Carlo trials performed for each size. The results are shown in Figure 5.3. First note that the amount of SER improvement is much higher for higher amounts of variation. Surprisingly, the SER improvement levels off very quickly with neighborhood size, meaning most of the possible improvement can be realized with very small and tractable sizes. For instance, the amount of improvement with neighborhoods of just eight flip-flops is 75% of the amount associated with a maximum-size neighborhood (100,000). This holds true for all three values of SEU variation that were simulated.

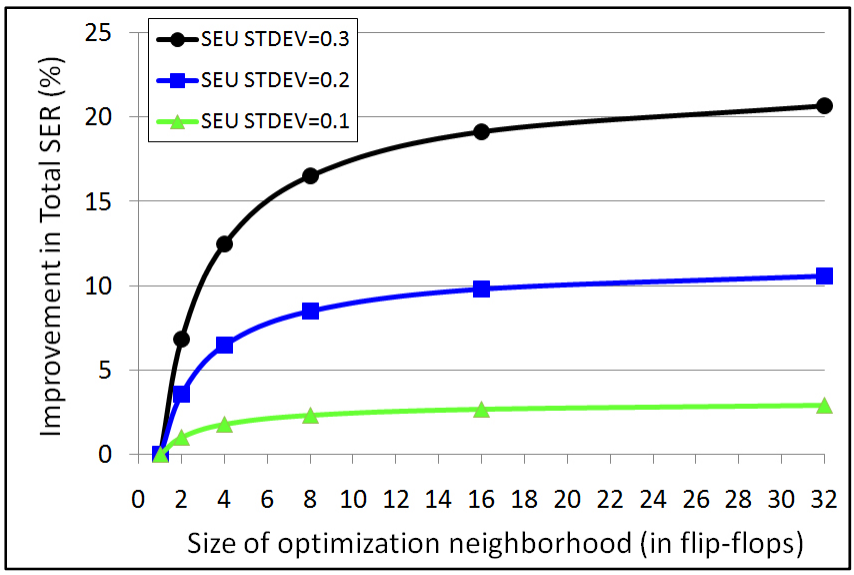

Figure 5.3: Improvement in SER vs. neighborhood size, for three different amounts of SEU variation

Perhaps the most surprising result from this study is the effectiveness of logic reconfiguration within very small neighborhoods. Optimization within each neighborhood of just eight flip-flops can produce significant reductions in the soft error rate. This is encouraging since incremental place and route is far more practical when the neighborhood size is small. In fact, modern reconfigurable platforms tend to naturally employ small neighborhoods in their logic arrays. Xilinx's Virtex-6 FPGAs, as well as all of its upcoming 7th generation FPGAs (Virtex, Kintex, Artix), are organized into clusters (aka slices) containing eight flip-flops [116]. The logic within a cluster uses the same clock and control signals, making it more interchangeable.

We found that the mean number of faults that must be observed in situ per latch to enable an effective re-implementation is on the order of ten. This compares favorably to an existing rule of thumb [1] suggesting that 100 SEU events are generally required for accurate estimation of fault susceptibility. The advantage of the investigated approach is

that highly accurate estimates of upsetability are not required; all that is needed is enough relative data to identify most of the outliers (components that are particularly upsetable or non-upsetable). Whether ten faults per latch can be generated in a timely fashion depends heavily on the radiation environment and the ability to lower Q_{crit}. In most applications, SEUs are far too infrequent even with accelerated testing, and thus relative upsetabilities cannot be readily characterized with this approach. In situ characterization of fault upsetabilities may become more feasible for emerging technologies with inherently high fault rates. Nevertheless, in general there is a strong need for novel and practical methods of characterization such as the proposed on-chip noise emulation from Section 4.6.

5.3 Case Study

In this section, we present an example of physically-adaptive computing in the context of transient faults, using actual hardware introspection data from Chapter 4.

5.3.1 Problem Statement

We now state the problem using the PAC formalism presented in Chapter 1. We start by assigning the function F to be each of a set of benchmarks functions (circuits) from the IWLS2005 OpenCores suite [53]. Each of the 15 selected functions has no more than 1,024 state bits. The functions are synthesized with version 10.1 of the Xilinx synthesis tool to generate the function netlists *FN*.

We use two reconfigurable substrates Sub_1 and Sub_2 which are instances of a modern FPGA with 65 nm features and fine-grained reconfigurable logic (Xilinx Virtex-5 XC5VLX110T). We specifically study in detail a 32 × 32 array of 1,024 flip-flops near

the lower right portion of the chip. We assume that the substrate parameters of interest (flip-flop upsetabilities) are static while the adaptation agent is active.

The logical parameters are the signal probabilities SP_i of the state bits in *FN*; these probabilities indicate how often each state bit is in the 1 state for the representative workload. The substrate parameters that must be characterized by the PAC system are the four types of upsetability: $m0_j$, $m1_j$, $s0_j$, and $s1_j$, for each of the 1,024 flip-flops. Lastly, the operational parameter of interest is the total fault rate.

A configuration *Config* has an associated matrix **x** that specifies which logical state bits are placed at which flip-flop locations. The variable x_{ij} is 1 if state bit *i* is matched with flip-flop *j*, and is 0 otherwise. The fault rate of a bit/flip-flop pair *ij* is essentially the fraction of time spent in a state multiplied by the fault rate associated with that state, summed over the four type of faults. As in Chapter 4, we assume here that the clock duty cycle is 50% and thus the master and slave latches are exposed for equal periods. The cost of the bit/flip-flop pair is defined here to be the associated fault rate:

$$cost_{ij} = (m1_j + s1_j)\, SP_i + (m0_j + s0_j)\,(1 - SP_i). \tag{5.1}$$

We define the cost function for configuration-substrate pair (*Config*, *Sub*) to be the total fault rate. Assuming that faults affect single latches, we sum all of the component fault rates, and treat the MTBF as the inverse of this total fault rate:

$$Cost(Config,\ Sub) = \sum_i \sum_j cost_{ij} x_{ij}. \tag{5.2}$$

The goal for the PAC system is to characterize the necessary parameters and generate new configuration-substrate pairs (*Config*, *Sub*) that minimize cost (maximize fit-

ness). Note that this involves reducing the raw number of transient faults, a strategy called fault avoidance. Descriptions of our experiments and results are given next.

5.3.2 Experimental Results

We created an experimental FPGA-based system to test self-optimization for local variations, and to develop a better understanding of the achievable overhead. The system is capable of performing self-characterization of locally-varying latch upsetabilities on actual hardware. Lightweight tools for re-implementing an application, e.g., tools that can run on the FPGA itself, are not yet available from chip vendors; however, we created a tool capable of evaluating alternative implementations off-line.

5.3.2.1 Self-Characterization

The characterization method involves the autonomous injection of small amounts of charge into a latch via short pulses on the asynchronous set and reset lines, as described in the previous chapter. Self-characterization was performed for 256 clusters of four flip-flops on both substrates. For every flip-flop, the system characterized the four values of transient fault upsetability associated with the given noise environment. As was seen in Chapter 4, significant local variations were found.

5.3.2.2 Experiment 1: Physical Variation Only

In the first experiment involving re-implementation, we aimed to find a lower bound on the effectiveness of both self-adaptation and assisted adaptation. The physical parameters were determined via self-characterization, while the logical parameters (signal probabilities) were assumed to be uniform so there would be no logical variation to take advantage of. All signal probabilities were set to 0.5 for this experiment only.

We implemented the 15 benchmark functions using the conventional Xilinx ISE 10.1 tool flow, which is independent of the unique reconfigurable substrate. The resulting numbers of state bits, LUTs, and clusters are shown in Table 5.1. Circuits were allowed to use as much logic as necessary, with the constraint that all state bits reside at flip-flops in the 32 × 32 array under test.

Table 5.1: Benchmark circuit characteristics

Benchmark circuit	No. of state bits	No. of LUTs	No. of clusters (aka slices)	No. of clusters w/ >0 state bits	Intra-cluster state bit packing
steppermotordrive	39	62	17	13	0.75
pci_spoci_ctrl	55	190	80	31	0.44
des_area	64	420	281	25	0.64
ss_pcm	87	40	31	25	0.87
usb_phy	98	78	41	40	0.61
i2c	114	173	81	46	0.62
sasc	116	74	57	48	0.60
des3_area	128	593	475	32	1.00
simple_spi	131	114	57	44	0.74
systemcdes	190	347	129	59	0.81
spi	229	609	340	171	0.33
tv80	359	1579	571	151	0.59
wb_dma	516	721	371	162	0.80
aes_core	530	1599	654	189	0.70
systemcaes	670	1411	502	224	0.75

The cost of these original, substrate-independent implementations was calculated. This was done by extracting the cluster coordinates and flip-flop sites of all state bits (using Xilinx Floorplanner), and then evaluating (5.2). Next, an adapted implementation with the lowest cost was found. The method used was re-placement of the state bits within each cluster. Since there are four flip-flop locations and up to four state bits that need to be placed, there are up to 4! = 24 alternative placements for each cluster. Our re-implementation tool performed the variation-aware optimization within each of the 256

clusters, needing only 256 × 24 evaluations of (5.2). The new total mean time between latch failures (MTBF) was then calculated by summing over all clusters.

An example of intra-cluster optimization is depicted in Figure 5.1. Three state bits are placed among the four flip-flops in a cluster. As a system learns more about the relevant parameters, a new and better implementation is found. Gains can come not only from avoiding unreliable components (e.g., flip-flop D) but from matching logical and physical variations when possible.

The next task in this experiment was quantifying the effects of assisted adaptation, in which a heavier-weight algorithm can be employed. The Xilinx tools do not readily support global, per flip-flop variation-aware placement, so we created a method serving as an approximation. First, the global packing of logic into clusters [2] was modified to achieve better balance. For instance, if seven state bits were originally packed into three clusters using flip-flop counts of 4, 2, and 1, our tool would re-pack the logic such that the distribution became 3, 2, and 2. This balancing decreases the number of fully-utilized clusters and thus increases the degrees of freedom—there are more opportunities for avoiding the worst flip-flops and leveraging the best ones. The number of clusters containing utilized flip-flops was held constant so as not to increase circuit area. Intra-cluster re-placement was then performed just as in the self-adaptation case but with the possibility of even better placements. The new MTBF was calculated and compared to the original substrate-independent MTBF.

The flows used in the experiments are shown in Figure 5.4. On the left is the traditional, non-adaptive flow. On the right is the additional flow used to emulate the PAC process.

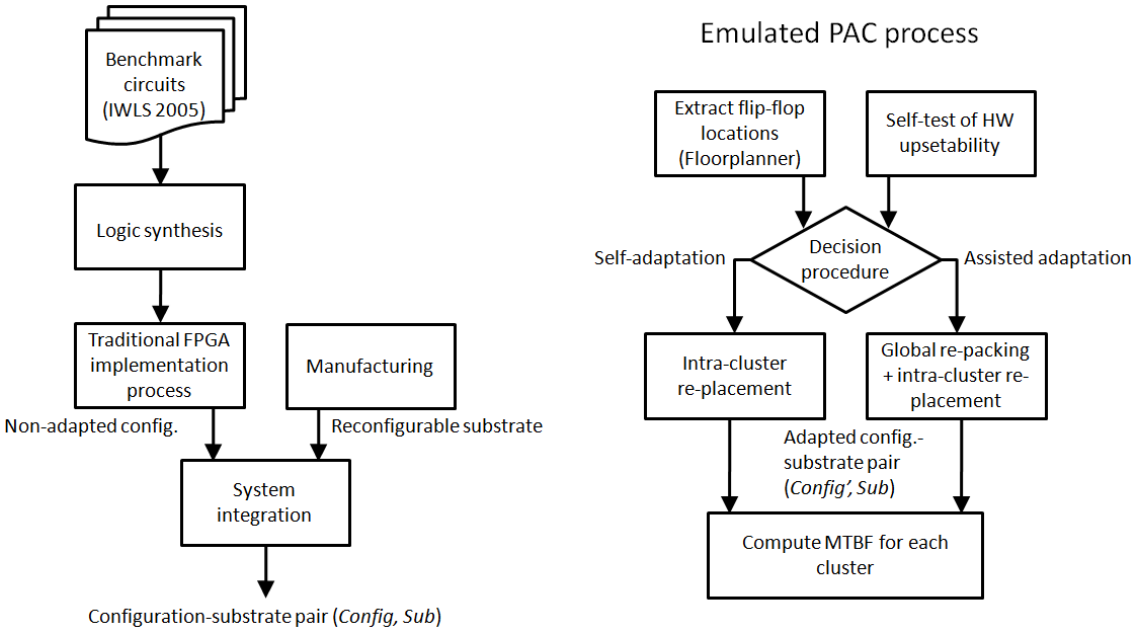

Figure 5.4: Traditional design flow (left) and the additional flow used to emulate the PAC process (right)

The results are shown in Figure 5.5. For self-adaptation, the MTBF improved by an average of 16% using substrate RS_1 and 14% when using RS_2. With assisted adaptation, the improvement reached an average of 35% and 25%.

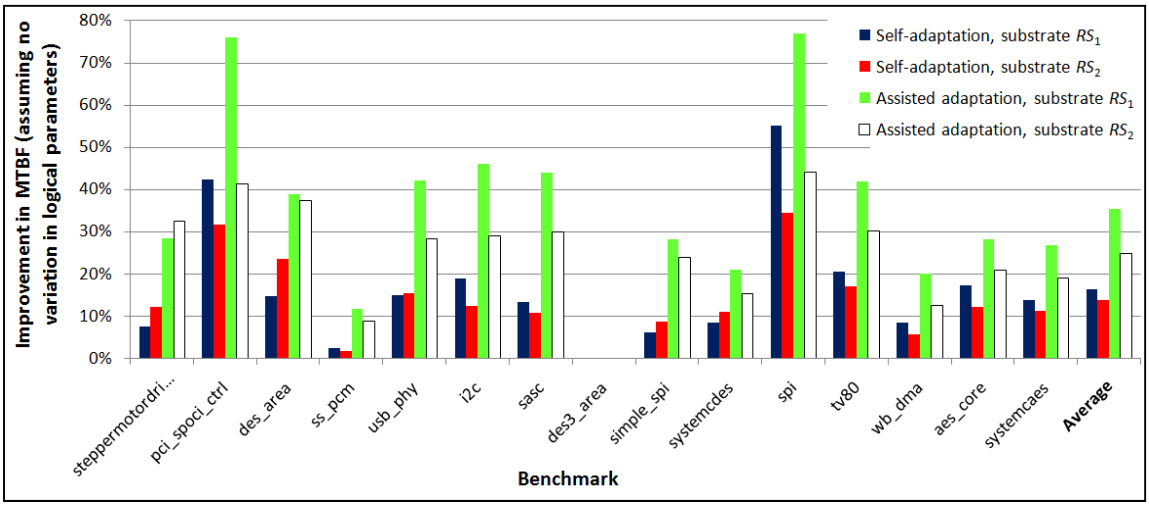

Figure 5.5: Results of experiment 1 showing the improvements in MTBF for self-adaptation and assisted adaptation relative to the non-adaptive case, assuming no variation in signal probabilities

One circuit, des3_area, showed no improvement in this experiment. This is because the circuit is a regular datapath and leads to clusters that are fully packed; without any spare flip-flops and without any logical variation to use as a counteracting force, there is no degree of freedom. Generally, as state bits are packed with less density within clusters, as shown in the right column of Table 5.1, the gains increase. The least dense circuit, *spi*, saw improvements as high as 77%.

5.3.2.3 Experiment 2: Physical and Signal Probability Variation

Having established a lower bound on the levels of improvement, we next conducted an experiment in which there was variation across the logical parameters (signal probabilities). This allowed us to quantify the additional benefits of using logical variation to counteract the physical variation. As with the previous experiment, the experimental PAC system used the actual physical parameter data and either performed self-adaptation (in the form of intra-cluster re-placement), or relied on assisted adaptation, in which case the aforementioned re-packing was performed along with intra-cluster re-placement. For each benchmark and substrate, the signal probabilities were assigned independent random floating-point values between 0 and 1. This is an approximation; in actual applications the signal probabilities can be found through logic simulation or via on-line sampling using a method such as Xilinx's "capture and readback." The process was repeated for ten trials, 15 benchmarks, and two substrates, for a total of 10 × 15 × 2 = 300 re-implementations for self-adaptation and again for assisted adaptation. In each trial, the MTBF after adaptation was compared with the MTBF of the original configuration-substrate pair. The results are shown in Figure 5.6. Across all benchmarks, the MTBF improved by an average of 31% and 27% after self-adaptation when using substrates RS_1

and RS_2, respectively. The gains are much higher than after self-adaptation with uniform signal probabilities (experiment 1), due to the logical variation counteracting the physical variation. For instance, a state bit with a high probability of being in the 1 state can be placed at a flip-flop with a low rate of upsets in the 1 state. The gains are even greater with assisted adaptation, reaching 53% and 36%, respectively.

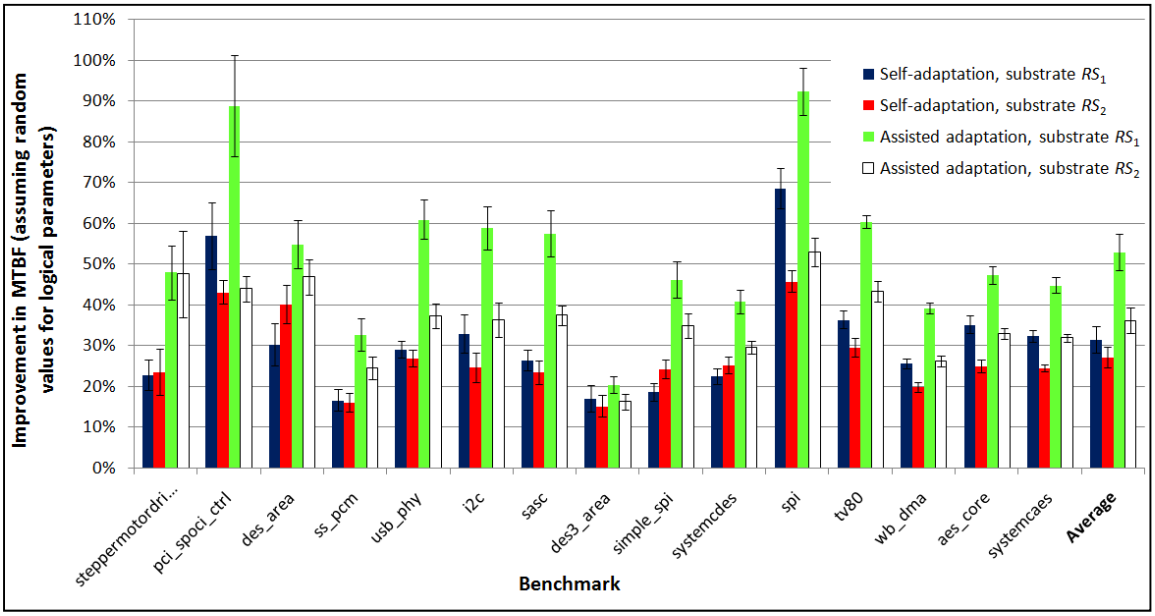

Figure 5.6: Results of experiment 2 showing the improvements in MTBF for self-adaptation and assisted adaptation relative to the non-adaptive case, assuming uniformly random signal probabilities. Error bars show the standard deviation across ten trials.

5.3.3 Limitations

One limitation of the case studies in this chapter is the assumption that all logic in a cluster can be re-ordered; in some cases there can be asymmetries (e.g., the use of carry logic) and restrictions that make re-ordering difficult. Moreover, with our tool we allowed very fine-grained variation-aware changes to an implementation, though this is not yet supported by FPGA vendor tools. There are reasons to believe that in-system re-implementation is becoming more feasible. Two of the leading vendors (Xilinx and Alte-

ra) have announced new and more complete support for partial reconfiguration [117][7]. As physical issues come to the fore, vendor plans for additional fine-grained methods [105] are likely to come to fruition. A final limitation of the case studies is that they involved optimization toward a single objective. In reality, there are typically multiple conflicting objectives (power, reliability, design effort, etc.) that require careful trade-off analysis.

5.4 Discussion

The results of our experiments illustrate some of the potential for physically-focused optimization. One observation is that the value of PAC depends highly on the amount of variation in both logical and physical parameters. For instance, substrate Sub_1 exhibits higher random variation in latch reliability than does Sub_2; the coefficient of variation is roughly 20% higher in relative terms as seen in Chapter 4. As a consequence, the PAC benefits are generally larger when using Sub_1. A second example is the effect of variation in logical parameters; in the presence of variation the benefits of self-adaptation doubled, from 16% (14%) to 31% (29%).

An additional finding is the importance of logic packing. In the absence of logical variation, the presence of spare flip-flops is especially important. The rightmost column of Table 5.1 provides one metric of intra-cluster packing—the number of state bits divided by the number of clusters containing state bits. In experiment 1, the gains are generally inversely related to the level of intra-cluster packing. In experiment 2, additional gains are possible via logical-physical matching.

The potential for optimization is also related to environmental parameters. Latches can have different noise thresholds at which upsets occur, so if the noise environment is dominated by events around the threshold energies, there can be huge variations. Some components may never incur a fault while others may do so frequently. Variation can furthermore be data-dependent; we found that the latches exhibited much higher variation when in the 1 state. These phenomena point to the need for careful characterization and modeling of the many types of system parameters.

CHAPTER 6

Conclusions

This dissertation is intended to contribute to an ambitious long-term vision, in which digital systems achieve significantly higher efficiency and reliability via adaptation. We highlight the main contributions below, then point to and explore some promising directions for future research. Lastly, we engage in some introspection of our own and provide final thoughts on this endeavor.

6.1 Summary of Contributions

The main contributions of this dissertation are the following:

- A rationale and conceptual framework for physically-adaptive computing
- Novel methods of sensing regional variations in reconfigurable systems, including a multi-use FPGA-based sensor one-quarter the size of the previous smallest design
- A novel method of self-test for local variations in upsetability, using on-chip noise emulation
- Previously unpublished silicon data for 65 nm FPGAs, with several surprising findings involving delay, leakage, temperature, and upsetability

- A new soft error figure-of-merit that can be used to evaluate the fitness of alternative designs
- Self-optimization methods for regional variations, such as in-system selection among alternative designs
- Self-optimization methods and experiments for local variations, including a case study quantifying the possibilities for both self-adaptation and assisted adaptation

The general concepts of introspection, variation, and fine-grained optimization have been synthesized into a common framework we call PAC. Typically, these concepts are studied piecemeal and in disparate research areas. In comparing and contrasting natural and artificial adaptive systems, we have identified a unique opportunity for digital systems regarding physical adaptation. We have outlined the key elements of a unified PAC framework and have provided arguments for the importance of fine granularity and remote assistance.

We have established a novel approach to instrumenting an application with an array of flexible, low-cost sensors. Along with improved measurement procedures, the proposed sensor arrays allow estimation of many regional, spatially-correlated parameters. The proposed FPGA-based sensor, which includes a novel residue number system ring counter, is one-quarter the size of the previous smallest sensor.

We have demonstrated a novel self-test strategy of on-chip noise emulation, and presented a design for an efficient, 50-ps-resolution circuit for FPGAs. Noise emulation can uncover previously hidden local variations not observable with radiation testing or in situ monitoring. This approach to introspection can be implemented with separate bitstreams and requires no permanent hardware overhead—something that cannot be achieved with

ASICs or MPUs. The overhead of using an extra bitstream is small; a system such as the Cibola Flight Experiment can store 20 bitstreams, and in many cases, self-test bitstreams can be maintained at a remote server or peer and retrieved occasionally as needed.

Data on fine-grained physical variations tends to be scarce in the literature. We have contributed some unique empirical data. First, we have determined the relationship between ring oscillator frequencies and temperature on a 65 nm FPGA; we found that the curve has flattened by a factor of eight compared to the Xilinx XC3000/4000 chips from the mid-1990s. The data can be used to calibrate and validate analytical models of temperature dependence, which indeed predict that the curve flattens out as supply voltages decrease [110]. Second, we have quantified spatial variations in delay, and established some limits on sensing temperature. Additionally, we have provided novel data regarding local variations in upsetability.

Soft error metrics such as architectural vulnerability factor are widely used. However, we have demonstrated that certain situations require a more comprehensive metric. We introduced the notion of vulnerability over space and time (VST) which accounts for the differing amounts of spatial and temporal resources used by different designs. This metric can be used as a logical parameter in a cost function for evaluating alternative designs.

We have identified many new opportunities for self-optimization. For regional variations, we studied in-system evaluation and selection among alternative designs. Case studies of CORDIC, floating-point addition, and FFT applications showed potential improvements in fitness of roughly 30%–40%.

For local variations, we quantified the benefits of both re-packing of logic across clusters, and re-placement of logic within clusters. The results show that significant gains (30% increase in MTBF) are possible via self-adaptation, and even larger gains (40%–50%) via assisted adaptation.

6.2 Directions for Future Work

The work discussed here suggests a variety of promising research avenues involving support for physical adaptation, variations of the PAC paradigm, and adaptive computing closer to physical limits.

6.2.1 PAC-Friendly Tools, APIs, Hardware

Further work is needed regarding infrastructure that is conducive to PAC. Lightweight tools are needed for performing more of the adaptation process in-system rather than by a server. Ideally, reconfigurable platforms would have open bitstream formats and tools, but commercial interests may hinder that goal. Instead, platform vendors could at a minimum provide APIs that allow more powerful adaptation in the field. Along the lines of Xilinx's Set_CLB_Bits function which provides control of logic block configurations [118], vendors could provide a means to swap logic cell functions and routing. Research into reconfigurable hardware platforms is needed regarding the appropriate number and resolution of built-in sensors, the addition of power gating (which would help to isolate regions during self-test), and symmetric local interconnect that makes swapping easier. As system configurations become more fluid in the field, there will be a need for new approaches to bitstream security.

Design productivity is an increasingly important goal given the immense scale and complexity of systems today. Thus the inclusion of PAC principles in a system should not greatly impact the work of system or application designers. Further research is needed regarding integrating PAC automatically. For example, an infrastructure such as the one proposed in Chapter 3 (sensors, interconnection, physical constraints, and drivers) could be turned into a parameterizable intellectual property core and added to the IP library provided by FPGA vendors. This would allow designers to start with a ready-made PAC infrastructure. Similarly, self-test circuits and optimization routines for local variations could be included transparently in the vendor bitstreams, similar to what Xilinx has already stated as a goal [105].

6.2.2 Variations of Physically-Adaptive Computing

Multiple spatial scales. PAC can potentially be applied at multiple spatial scales—local, regional, system-wide, even inter-system. An analogy exists to biology, where adaptations can occur at a single DNA base pair, a gene, and even in the number of chromosomes. Interactions between adaptation at multiple scales need to be considered. For instance, local introspection data may need to be combined with regional data. When evaluating potential regional optimizations, an agent may have to consider the loss of previously performed local optimizations. A better understanding is needed of the conditions for which local adaptation alone is the best strategy.

Multiple agents. While our experiments used a single adaptation agent, most of the proposed methods are non-global, meaning they can potentially be used for decentralized, multi-agent PAC. As an example, each region of a many-core system could have an associated agent performing introspection and self-optimization. A decentralized approach

is likely to be necessary given the growing amounts of system integration and parallel processing.

Multiple hardware contexts. Some reconfigurable systems have multiple contexts that use the same chip or region of a chip [47]. For instance, a system may alternate between high-performance and high-reliability implementations of an application, or may dynamically switch between IP cores implementing different protocols. Both of the leading FPGA vendors now support partial reconfiguration for the latter case. An area for future work is re-usability of PAC knowledge across contexts. Can knowledge of adaptations be extracted from a bitstream and leveraged in a different context? When does it make sense for an agent to optimize all contexts simultaneously rather than separately?

Multiple systems. A further line of inquiry is the nature of PAC cooperation across multiple systems. At a minimum, peers may be able to share logical data and self-test bitstreams, reducing the local memory overhead and server communication. Certain types of computations may be a poor physical fit for certain systems, and perhaps could be migrated to achieve a better fit. One can envision systems advertising their unique capabilities in a marketplace of computation. Research would be needed into the limits and possibilities of such an extension.

6.2.3 The Problem of Heat

Excessive heat is a fundamental physical problem for computing systems, causing exponential increases in wearout, sub-threshold current, and eventually, thermal noise errors [85]. Removing heat is itself expensive. Cavin *et al.* warn that continued scaling "will result in the generation of thermal loads that cannot be managed by any known heat removal technology…Radically new solutions for heat removal are necessary" [20].

Thermal hotspots do not always match what is predicted by simulation models and tools such as HotSpot, due to substrate variations, packaging defects, and unexpected workloads. Thus on-line sensing and fine-grained re-implementation of computations can play a role as an additional mitigation method.

Systems can in theory be cooled to ultra-low temperatures. However, low-temperature operation has been attempted in the past as a means to achieving high performance, and has run into at least two roadblocks. First, total system efficiency is limited by the energy required for cooling, as stipulated by Carnot's Theorem in thermodynamics. Second, while many circuit properties improve at low temperatures, some effects can worsen such as hot carrier injection. Thus new manufacturing technologies, and possibly non-CMOS technologies such as SiGe, may be required. A resurgence of interest in low-temperature computing is likely. Already, there are experimental quantum computers that operate at −269°C (4K). A further possibility is discussed below.

6.2.4 PAC in Space

A prime area for future study is physically-adaptive computing in space systems. NASA's FPGA-based SpaceCube 1.0 is currently flying on the International Space Station and undergoing reliability experiments. A new space-qualified 65 nm FPGA called the Xilinx Virtex-5QV is expected to enter production in 2011 [126]. This promising platform will have extra protection against SETs and SEUs and may be effective for mission critical computations such as command and data handling and flight control. The natures of physical variations, wearout, and fault mechanisms on this platform remain to be seen.

Each Virtex-5QV chip is expected to be very expensive and a full generation behind the commercial-grade FPGA chips such as the 40 nm Virtex-6. Thus there will be ongoing needs for higher-performing, lower-cost platforms, especially for non-mission-critical computations such as payload processing.

Regarding the problem of heat mentioned above, we pointed out that artificial cooling is costly. One fascinating possibility is to leverage environments with natural ultra-low temperatures. The coldest such places are beyond Earth. Certain shaded craters on the Moon are believed to be the coldest spots in the solar system, with temperatures below −223°C (50K). Combined with specialized cooling technology, it may even be possible to reach −263°C (10K) [84]. Computing in such environments is currently prohibitive, but that may well change. One likely scenario is for low-temperature computing technologies to be successfully realized first for ground-based systems. These systems will have unique advantages such as boosted performance or quantum capabilities, but ultimately will have limited efficiency in terrestrial environments due to cooling requirements. A subsequent step would be to find ways to deploy such technology in naturally cold space environments. Though adaptive computing in space sounds like a distant goal, it may be where the laws of thermodynamics lead.

6.3 Final Thoughts

Given the essential role digital systems play in society, I feel it is important to take steps, however modest, in the direction of physical adaptability. In my view, the opportunities and rewards are plentiful.

My experiences along this dissertation journey offer a small glimpse into the possibilities. Recounting just one stirring experience, I came to believe many months ago that, despite the conventional wisdom, local variations in upsetability were likely significant and that they could and should be uncovered. I subsequently implemented a method of self-test in an effort to shine light on this question. Upon turning on the experimental system for the first time, the early data showed no sign of variation, with patterns of all 0's (0x0 hexadecimal) or all 1's (0xF). After some initial concern, I spent a few hours fixing bugs, learning about the instrument, and adjusting the noise. Then, for the first time, some wonderful patterns began to emerge. Suddenly the system was reporting non-uniform characters…0xB…0x8…0x5…and many more. The nanoscale details began to reveal themselves in beautiful and repeatable variations. I felt privileged and fascinated to be seeing a tiny, previously hidden part of nature. As the results got stronger, I was elated that the variations were significant and measurable, representing a new opportunity for systems to improve themselves.

Now, the broader journey continues, toward better understanding of computation at nanoscales, and toward more effective, dependable digital systems. Legendary astronomer Carl Sagan once said, "Somewhere, something incredible is waiting to be known." This is undeniably true. I find it inspiring that not only will we know new and incredible things, but that our engineered systems in service of society will come to know themselves at microscopic scales, and with our help, climb to peaks unimagined.

BIBLIOGRAPHY

[1] P. Adell and G. Allen, "Assessing and mitigating radiation effects in Xilinx FPGAs," JPL Publication, Feb. 2008.

[2] T. Ahmed, P. Kundarewich and J. Anderson, "Packing techniques for Virtex-5 FPGAs," *ACM Trans. Reconfigurable Technology and Systems*, vol. 2, no. 3, Sep. 2009.

[3] A.H. Ajami, K. Banerjee, A. Mehrotra and M. Pedram, "Analysis of IR-drop scaling with implications for deep submicron P/G network designs," *Proc. Int'l Symp. Quality Electronic Design*, pp. 35–40, Mar. 2003.

[4] G. Allen *et al.*, "Virtex-4QV static SEU characterization summary," JPL Publ. 08–16, April 2008.

[5] G.R. Allen and G.M. Swift, "Single event effects test results for advanced field programmable gate arrays," *Proc. Radiation Effects Data Workshop*, pp. 115–120, 2006.

[6] A. Aloisio, P. Branchini, R. Cicalese, R. Giordano, V. Izzo and S. Loffredo, "FPGA implementation of a high-resolution time-to-digital converter," *Nuclear Science Symp. Conf. Record*, pp. 504–507, 2007.

[7] Altera Corp., "Increasing design functionality with partial and dynamic reconfiguration in 28-nm FPGAs," http://www.altera.com, July 2010.

[8] T. Austin, V. Bertacco, S. Mahlke and K. Cao, "Reliable systems on unreliable fabrics," *IEEE Design and Test*, vol. 25, no. 4, July 2008.

[9] A. Balasubramanian *et al.*, "Effects of random dopant fluctuations (RDF) on the single event vulnerability of 90 and 65 nm CMOS technologies," *IEEE Trans. Nuclear Science*, vol. 54, no. 6, pp. 2400–2406, Dec. 2007.

[10] A. Bernauer, D. Fritz, B. Sander, O. Bringmann and W. Rosenstiel, "Current state of ASoC design methodology," *Proc. Organic Computing - Controlled Self-organization*, 2008.

[11] V. Betz, "FPGAs at 28nm: meeting the challenge of modern systems-on-a-chip," *Proc. Int'l Conf. Field Programmable Logic and Applications*, Aug. 2010.

[12] A. Biswas *et al.*, "Computing Accurate AVFs using ACE Analysis on Performance Models: A Rebuttal," *Computer Architecture Letters*, vol. 7, no. 1, pp. 21–24, Jan. 2008.

[13] A. Biswas, N. Soundararajan, S. Mukherjee and S. Gurumurthi, "Quantized AVF: a means of capturing vulnerability variations over small windows of time," *Proc. IEEE Workshop on Silicon Errors in Logic – System Effects*, March 2009.

[14] E. Boemo and S. López-Buedo, "Thermal monitoring on FPGAs using ring-oscillators," *Proc. Int'l Workshop on Field-Programmable Logic and Applications*, pp. 69–78, 1997.

[15] J. Bongard, V. Zykov and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.

[16] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[17] S. Borkar, "Design perspectives on 22nm CMOS and beyond," *Proc. Design Automation Conf.*, pp. 93–94, 2009.

[18] G. Bronevetsky and B. de Supinski, "Soft Error Vulnerability of Iterative Linear Algebra Methods," *Proc. Workshop on Silicon Errors in Logic - System Effects*, April 2007.

[19] M. Caffrey *et al.*, "On-Orbit Flight Results from the Reconfigurable Cibola Flight Experiment Satellite (CFESat)," *Proc. Int'l Symp. Field-Programmable Custom Computing Machines*, pp. 3–10, 2009.

[20] R.K. Cavin and V.V. Zhirnov, "Future devices for information processing," *Proc. Solid-State Device Research Conf.*, pp. 7–12, Sep. 2005.

[21] P. Chaparro *et al.*, "Soft Error Protection Mechanisms for In-Order Cores," *Proc. Workshop on Silicon Errors in Logic - System Effects*, March 2008.

[22] P. Chen, M. Shie, Z.-Y. Zheng, Z.-F. Zheng and C. Chu, "A fully digital time-domain smart temperature sensor realized with 140 FPGA logic elements," *IEEE Trans. Circuits and Systems I*, vol. 54, no. 12, pp. 2661–2668, Dec. 2007.

[23] P. Chen, J.-S. Lai and P.-Y. Chen, "FPGA Vernier digital-to-time converter with 1.58 ps resolution and 59.3 minutes operation range," *IEEE Trans. Circuits and Systems*, vol. 57, no. 6, pp. 1134–1142, 2010.

[24] L. Cheng, J. Xiong, L. He and M. Hutton, "FPGA performance optimization via chipwise placement considering process variations," *Proc. Int'l Conf. Field Programmable Logic and Applications*, pp. 1–6, Aug. 2006.

[25] D. Clark and L. Weng, "Maximal and near-maximal shift register sequences: efficient event counters and easy discrete logarithms," *IEEE Trans. Computers*, vol. 43, no. 5, pp. 560–568, May 1994.

[26] J. Cong and M. Sarrafzadeh, "Incremental physical design," *Proc. Int'l Symp. Physical Design*, pp. 84–92, 2000.

[27] R. Conn Jr., "Method and apparatus for measuring localized temperatures on integrated circuits," U.S. Patent 6067508, May 23, 2000.

[28] S. Dighe *et al.*, "Within-die variation-aware dynamic-voltage-frequency scaling core mapping and thread hopping for an 80-core processor," *Proc. Int'l Solid-State Circuits Conf.*, pp. 174–175, 2010.

[29] S. Dobson, R. Sterritt, P. Nixon and M. Hinchey, "Fulfilling the vision of autonomic computing," *IEEE Computer*, vol. 43, no. 1, pp. 35–41. Jan. 2010.

[30] J.M. Emmert and J.A. Cheatham, "On-line incremental routing for interconnect fault tolerance in FPGAs minus the router," *Proc. Defect and Fault Tolerance in VLSI Systems*, pp. 149–157, 2001.

[31] T. Flatley, "Advanced hybrid on-board science data processor - SpaceCube 2.0," *Proc. Earth Science Technology Forum*, June 2010.

[32] J. Fletcher, M. Perlman, W. Rousey and A. Messner, "System for generating timing and control signals," U.S. Patent 3866022, Feb. 11, 1975.

[33] M. French, E. Anderson and D. Kang, "Autonomous system on a chip adaptation through partial runtime reconfiguration," *Proc. Int'l Sym. Field-Programmable Custom Computing Machines*, pp. 77–86, 2008.

[34] X. Fu, T. Li and J. Fortes, "Soft error vulnerability aware process variation mitigation," *Proc. Symp. High-Performance Computer Arch.*, Feb. 2009.

[35] J. Gal-Edd and C.C. Fatig, "L2-James Webb Space Telescope operationally friendly environment?" *Proc. IEEE Aerospace Conf.*, vol. 1, March 2004.

[36] M.G. Gericota *et al.*, "DRAFT: an on-line fault detection method for dynamic and partially reconfigurable FPGAs," *Proc. Int'l On-Line Testing Workshop*, pp. 34–36, 2001.

[37] B. Gill, N. Seifert and V. Zia, "Comparison of alpha-particle and neutron-induced combinational and sequential logic error rates at the 32nm technology node," *Proc. Int'l Reliability Physics Symp.*, pp. 199–205, Mar. 2009.

[38] B. Gojman and A. DeHon, "VMATCH: using logical variation to counteract physical variation in bottom-up, nanoscale systems," *Proc. Int'l. Conf. on Field-Programmable Technology*, pp. 78–87, Dec. 2009.

[39] S. Goyal and J. Carter, "A lightweight secure cyber foraging infrastructure for resource-constrained devices," *Proc. Workshop on Mobile Computing Systems and Applications*, pp. 186–195, Dec. 2004.

[40] J. Greco *et al.*, "Hardware/Software Interface for High-Performance Space Computing with FPGA Coprocessors," *Proc. IEEE Aerospace Conf.*, May 2006.

[41] Y. He and M. Ashtijou, "iBoard: A highly-capable, high-performance, reconfigurable FPGA-based platform," *Proc. NASA/ ESA Conf. Adaptive Hardware & Systems*, pp. 73–74, June 2010.

[42] T. Heijmen *et al.*, "A comprehensive study on the soft-error rate of flip-flops from 90-nm production libraries," *IEEE Trans. Device and Materials Reliability*, vol. 7, no. 1, pp. 84–96, March 2007.

[43] T. Heijmen, "Soft-error vulnerability of sub-100-nm flip-flops," *Proc. Int'l On-Line Testing Symp.*, pp. 247–252, 2008.

[44] J. Hellerstein, "Engineering autonomic systems," *Proc. Int'l Conf. Autonomic Computing*, pp. 75–76, June 2009.

[45] J. Holland, *Adaptation in Natural and Artificial Systems,* Ann Arbor, MI: University of Michigan Press, 1975.

[46] N.J. Howard, A.M. Tyrrell and N.M. Allinson, "The yield enhancement of field-programmable gate arrays," *IEEE Trans. VLSI Systems*, vol. 2, no. 1, pp. 115–123, 1994.

[47] C. Huang and F. Vahid, "Server-side coprocessor updating for mobile devices with FPGAs," *Proc. Int'l Symp. Field-Programmable Gate Arrays*, pp. 125–134, Feb. 2010.

[48] Z. Hyder and J. Wawrzynek, "Defect tolerance in multiple-FPGA systems," *Proc. Int'l Conf. Field Programmable Logic and Applications*, pp. 247–254, Aug. 2005.

[49] N. Jiang and M. Parashar, "Enabling autonomic power-aware management of instrumented data centers," *Proc. Int'l Symp. Parallel & Distributed Processing*, May 2009.

[50] P.H. Jones *et al*., "Adaptive thermoregulation for applications on reconfigurable devices," *Proc. Int'l Conf. on Field-Programmable Logic and Applications*, pp. 246–253, Aug. 2007.

[51] Intel Corp., Intel Technology Journal, vol. 7, no. 2, May 2003.

[52] International Technology Roadmap for Semiconductors, 2009 Edition, <http://www.itrs.net>.

[53] IWLS2005 Benchmarks, http://www.iwls.org/iwls2005/benchmarks.html, July 2010.

[54] K. Katsuki, M. Kotani, K. Kobayashi and H. Onodera, "A yield and speed enhancement scheme under within-die variations on 90nm LUT array," *Proc. Custom Integrated Circuits Conf.*, pp. 601–604, Sept. 2005.

[55] J. Kellington *et al.*, "IBM POWER6 processor soft error tolerance analysis using proton irradiation," *Proc. Workshop on Silicon Errors in Logic - System Effects*, April 2007.

[56] P. Kohlbrenner and K. Gaj, "An embedded true random number generator for FPGAs," *Proc. Int'l Symp. Field-Programmable Gate Arrays*, pp. 71–78, 2004.

[57] A. Krishnamoorthy and A. Detofsky, "Mapping variations in local temperature and local power supply voltage that are present during operation of an integrated circuit," U.S. Patent 7233163, Jun. 19, 2007.

[58] R. Kuhlman, "FPGA compilation on-site or in the cloud," *EETimes*, Aug. 9, 2010.

[59] A. Kumar and M. Anis, "FPGA design for timing yield under process variations," *IEEE Trans. VLSI Systems*, vol. 18, no. 3, pp. 423–435, March 2010.

[60] K. Kumar and Y. Lu, "Cloud computing for mobile users: can offloading computation save energy?" *IEEE Computer*, vol. 43, no. 4, pp. 51–56, April 2010.

[61] E. Kursun and C.Y. Cher, "Variation-aware thermal characterization and management of multi-core architectures," *Proc. Int'l. Conf. Computer Design*, pp. 280–285, Oct. 2008.

[62] V. Lakamraju and R. Tessier, "Tolerating operational faults in cluster-based FPGAs," *Proc. Int'l Symp. Field-Programmable Gate Arrays*, pp. 187–194, 2000.

[63] D. Lewis *et al.*, "Architectural enhancements in Stratix-III and Stratix-IV," *Proc. Int'l Symp. Field-Programmable Gate Arrays*, pp. 33–42, Feb. 2009.

[64] S. López-Buedo and E. Boemo, "A method for temperature measurement on reconfigurable systems," *Proc. Design of Circuit and Integrated Systems Conf.*, pp. 727–730, Nov. 1997.

[65] S. López-Buedo and E. Boemo, "Making visible the thermal behaviour of embedded microprocessors on FPGAs: a progress report," *Proc. Int'l Symp. Field-Programmable Gate Arrays*, pp. 79–86, 2004.

[66] P. Mangalagiri, S. Bae, R. Krishnan, Y. Xie and V. Narayanan, "Thermal-aware reliability analysis for platform FPGAs," *Proc. Int'l Conf. Computer Aided Design*, pp. 722–727, 2008.

[67] Y. Matsumoto, M. Hioki, T. Kawanami and H. Koike, "Suppression of intrinsic delay variation in FPGAs using multiple configurations," *ACM Trans. Reconfigurable Technology and Systems*, vol. 1, no. 1, March 2008.

[68] J. A. McCann and M. C. Huebscher, "Evaluation issues in autonomic computing," *Proc. Grid and Cooperative Computing Workshops*, pp. 597–608, 2004.

[69] S.O. Memik, R. Mukherjee, M. Ni and J. Long, "Optimizing thermal sensor allocation for microprocessors," *IEEE Trans. Computer-Aided Design of ICs and Systems*, vol. 27, no. 3, pp. 516–527, Mar. 2008.

[70] R. Merritt, "ARM CTO: power surge could create 'dark silicon'," *EETimes*, October 22, 2009.

[71] J.F. Meyer, "Computation-based reliability analysis," *IEEE Trans. Computers*, vol. C-25, no. 6, pp. 578–584, June 1976.

[72] S. Michalak, "Standards for SER in system components: a HPC perspective," *Proc. IEEE Workshop on Silicon Errors in Logic – System Effects*, March 2009.

[73] H. Mostafa, M. Anis and M. Elmasry, "Comparative analysis of process variation impact on flip-flops soft error rate," *Proc. Symp. Quality Electronic Design*, pp. 103–108, 2009.

[74] S. Mukherjee, *Architecture Design for Soft Errors*, Burlington, MA: Morgan Kauffman, 2008.

[75] G. Nabaa, N. Azizi and F. Najm, "An adaptive FPGA architecture with process variation compensation and reduced leakage," *Proc. Design Automation Conf.*, pp. 624–629, 2006.

[76] National Instruments, “FPGAs – under the hood,” White Paper, August 20, 2010.

[77] D.C. Ness, C.J. Hescott and D.J. Lilja, “Improving nanoelectronic designs using a stat. approach to identify key parameters in circuit level SEU simulations,” *Proc. Sym. Nanoscale Arch.*, pp. 46–53, Oct. 2007.

[78] D. Patterson, “The trouble with multicore,” *IEEE Spectrum*, vol. 47, no. 7, pp. 28–32/53, July 2010.

[79] S. Paul, S. Mukhopadhyay and S. Bhunia, “A variation-aware preferential design approach for memory based reconfigurable computing,” *Proc. Int’l Conf. Computer Aided Design*, pp. 180–193, 2009.

[80] I. Polian, J.P. Hayes, S. Kundu and B. Becker, “Transient fault characterization in dynamic noisy environments,” *Proc. Int’l Test Conf.*, pp. 1039–1048, 2005.

[81] G.M. Quénot, N. Paris and B. Zavidovique, “A temperature and voltage measurement cell for VLSI circuits,” *Proc. Euro ASIC ’91*, pp. 334–338, 1991.

[82] G. Reis *et al.*, “Design and evaluation of hybrid fault-detection systems,” *Proc. Int’l Symp. Computer Architecture*, pp. 148–159, June 2005.

[83] R. Rubin and A. DeHon, “Choose-your-own-adventure routing: lightweight load-time defect avoidance,” *Proc. Int’l Symp. Field-Programmable Gate Arrays*, pp. 23–32, 2009.

[84] R.E. Ryan, L.W. Underwood, R. McKellip, D.P. Brannon and K.J. Russell, “Exploiting lunar natural and augmented thermal environments for exploration and research,” *Proc. Lunar and Planetary Science Conf.*, 2008.

[85] F.C. Sabou, D. Kazazis, R.I. Bahar, J. Mundy, W.R. Patterson and A. Zaslavsky, “Markov chain analysis of thermally induced soft errors in subthreshold nanoscale CMOS circuits,” *IEEE Trans. Device and Materials Reliability*, vol. 9, no. 3, pp. 494–504, 2009.

[86] M. Santambrogio, H. Hoffman, J. Eastep and A. Agarwal, “Enabling technologies for self-aware adaptive systems,” *Proc. NASA/ESA Conf. Adaptive Hardware & Systems*, pp. 155–162, 2010.

[87] A. Sanyal *et al.*, “A built-in self-test scheme for soft error rate characterization,” *Proc. Int’l On-Line Testing Symp.*, pp. 65–70, 2008.

[88] B. Schmerl and D. Garlan, “Exploiting architectural design knowledge to support self-repairing systems,” *Proc. Int’l Conf. Software Engineering and Knowledge Engineering*, pp. 241–248, 2002.

[89] B. Schroeder, E. Pinheiro and W.-D. Weber, “DRAM errors in the wild: a large-scale field study,” *Proc. Int’l Conf. Measurement and Modeling of Computer Systems*, pp. 193–204, 2009.

[90] P. Sedcole and P.Y.K. Cheung, “Within-die delay variability in 90nm FPGAs and beyond,” *Proc. Int’l Conf. Field Programmable Technology*, pp. 97–104, Dec. 2006.

[91] P. Sedcole and P.Y.K. Cheung, "Parametric yield in FPGAs due to within-die delay variations: a quantitative analysis," *Proc. Int'l. Symp. Field-Programmable Gate Arrays*, pp. 178–187, 2007.

[92] P. Sedcole, E. Stott and P.Y.K. Cheung, "Compensating for variability in FPGAs by re-mapping and re-placement," *Proc. Int'l Conf. Field Programmable Logic and Applications*, pp. 613–616, 2009.

[93] N. Seifert and N. Tam, "Timing vulnerability factors of sequentials," *IEEE Trans. Device and Materials Reliability*, vol. 4, no. 3, pp. 516–522, 2004.

[94] A. Silburt, "Internet core router soft error specification," *Proc. IEEE Workshop on Silicon Errors in Logic – System Effects*, March 2009.

[95] J. Smith and T. Xia, "High-precision delay testing of Virtex-4 FPGA designs," *Proc. Midwest Symp. Circuits and Systems*, pp. 1360–1363, Aug. 2007.

[96] M. Smith, *Application-Specific Integrated Circuits*, Reading, MA: Addison-Wesley, 1997.

[97] P. Springer, "Assessing application vulnerability to radiation-induced SEUs in memory," NASA Technical Report, 2001.

[98] S. Srinivasan and V. Narayanan, "Variation aware placement for FPGAs," *Proc. Emerging VLSI Technology and Architectures*, March 2006.

[99] S. Srinivasan, "Toward increasing FGPA lifetime," *IEEE Trans. Dependable and Secure Computing*, vol. 5, no. 2, pp. 115–127, April–June 2008.

[100] E. Stott, J.S.J. Wong, P. Sedcole and P.Y.K. Cheung, "Degradation in FPGAs: measurement and modelling," *Proc. Int'l Symp. Field-Programmable Gate Arrays*, pp. 229–238, Feb. 2010.

[101] C. Stroud *et al.*, "On-line BIST and diagnosis of FPGA interconnect using roving STARs," *Proc. Int'l On-Line Testing Workshop*, pp. 27–33, 2001.

[102] P. Sundarajan, A. Gayasen, N. Vijaykrishnan and T. Tuan, "Thermal characterization and optimization in platform FPGAs," *Proc. Int'l Conf. Computer Aided Design*, pp. 443–447, 2006.

[103] D. Sylvester, D. Blaauw and E. Karl, "ElastIC: An adaptive self-healing architecture for unpredictable silicon," *IEEE Design & Test of Computers*, vol. 23, no. 6, pp. 484–490, June 2006.

[104] Synopsys, Inc., "Nanometer challenges," Synopsys Insight, no. 1, 2010.

[105] S. Trimberger, "Structures and methods of overcoming localized defects in programmable integrated circuits by routing during the programming thereof," U.S. Patent 7,251,804, Jul. 31, 2007.

[106] I.A. Troxel, M. Fehringer and M.T. Chenoweth, "Achieving multipurpose space imaging with the ARTEMIS reconfigurable payload processor," *Proc. IEEE Aerospace Conf.*, pp. 1–8, Mar. 2008.

[107] F. Vahid, G. Stitt and R. Lysecky, "Warp processing: dynamic translation of binaries to FPGA circuits," *IEEE Computer*, vol. 41, no. 7, pp. 40–46, July 2008.

[108] R. Velazco, R. Ecoffet and F. Faure, "How to characterize the problem of SEU in processors & representative errors observed on flight", *Proc. IEEE On-Line Testing Symp.*, pp. 303–308, 2005.

[109] N. Wang, A. Mahesri, S.J. Patel, "Examining ACE Analysis Reliability Estimates Using Fault-Injection," *Proc. Intl. Symp. Computer Architecture*, pp. 460–469, 2007.

[110] D. Wolpert and P. Ampadu, "A sensor to detect normal or reverse temperature dependence in nanoscale CMOS circuits," *Proc. Int'l Symp. Defect and Fault Tolerance in VLSI Systems*, pp. 193–201, Oct. 2009.

[111] H.Y. Wong, L. Cheng, Y. Lin and L. He, "FPGA device and architecture evaluation considering process variations," *Proc. Int'l Conf. Computer-Aided Design*, pp. 19–24, Nov. 2005.

[112] J.S.J. Wong, P. Sedcole and P.Y.K. Cheung, "Self-characterization of combinatorial circuit delays in FPGAs," *Proc. Int'l Conf. Field-Programmable Technology*, pp. 17–23, Dec. 2007.

[113] C. Wu and D. Verma, "A sensor placement algorithm for redundant covering based on Riesz energy minimization," *Proc. Int'l Symp. Circuits and Systems*, pp. 2074–2077, May 2008.

[114] L. Xiaodong *et al.*, "Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions," *Proc. Intl. Conf. Dependable Systems and Networks*, pp. 266–275, June 2007.

[115] Xilinx Inc., "EasyPath-6 FPGA FAQs," http://www.xilinx.com, Nov. 16, 2009.

[116] Xilinx Inc., White Paper: 7 Series FPGAs, WP373 (v1.0), http://www.xilinx.com, June 21, 2010.

[117] Xilinx Inc., "Partial reconfiguration of Virtex FPGAs in ISE 12," WP374 (v1.0), http://www.xilinx.com, July 23, 2010.

[118] Xilinx Inc., LogiCORE IP XPS HWICAP (v5.00a), DS586, July 23, 2010.

[119] Xilinx Inc., Floating-Point Operator v3.0 Product Specification, September 28, 2006.

[120] Xilinx Inc., CORDIC v3.0 Product Specification, April 28, 2005.

[121] Xilinx Inc., Fast Fourier Transform v5.0 Product Specification, October 10, 2007.

[122] Xilinx Inc., "Optimizing FPGA power with ISE design tools," Xcell Journal, Second Quarter 2007.

[123] Xilinx Inc., "Xilinx architects ARM-based processor-first, processor-centric device," Xcell Journal, Second Quarter 2010.

[124] Xilinx, Inc., "Linear Feedback Shift Registers in Virtex Devices," App. note XAPP210 (v1.3), Apr. 2007.

[125] Xilinx, Inc., "Xilinx UG192 Virtex-5 FPGA System Monitor User Guide."

[126] Xilinx Inc., "Virtex-5QV FPGAs: first high density rad-hard reconfigurable FPGAs for space applications," Press Backgrounder, July 19, 2010.

[127] L. Zhang, L. S. Bai, R. P. Dick, L. Shang and R. Joseph, "Process variation characterization of chip-level multiprocessors," *Proc. Design Automation Conf.*, pp. 694–697, July 2009.

[128] D. Zhu, R. Melhem and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," *Proc. Intl. Conf. Computer Aided Design*, pp. 35–40, 2004.

[129] K. Zhu, "Post-route LUT output polarity selection for timing optimization," *Proc. Int'l Symp. Field-Programmable Gate Arrays*, pp. 89–96, 2007.

[130] K.M. Zick and J.P. Hayes, "High-level vulnerability over space and time to insidious soft errors," *Proc. High-Level Design, Validation and Test Workshop*, pp. 161–168, Nov. 2008.

[131] K.M. Zick and J.P. Hayes, "On-line characterization and reconfiguration for single event upset variations," *Proc. Int'l On-Line Testing Symp.*, pp. 243–248, June 2009.

[132] K.M. Zick and J.P. Hayes, "On-line sensing for healthier FPGA systems," *Proc. Int'l Symp. Field-Programmable Gate Arrays*, pp. 239–248, Feb. 2010.

[133] K.M. Zick and J.P. Hayes, "Self-test and adaptation for random variations in reliability," *Proc. Int'l Conf. Field Programmable Logic and Applications*, pp. 193–198, Aug. 2010.

[134] K.M. Zick and J.P. Hayes, "Toward physically-adaptive computing," *Proc. IEEE Int'l Conf. Self-Adaptive and Self-Organizing Systems*, pp. 124–133, Sep. 2010.