2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems

Toward Physically-Adaptive Computing

Kenneth M. Zick and John P. Hayes
Computer Science & Engineering Division, EECS Department
University of Michigan
Ann Arbor, MI 48109 USA
email: {kzick, jhayes}@umich.edu

Abstract—**As semiconductor technology approaches the atomic scale, electronic systems are increasingly burdened by physical variations and uncertainty. Traditionally-designed systems lack an ability to adapt to these fine-grained effects and are thus becoming more inefficient, error-prone, and subject to early wearout. This paper describes the paradigm of physically-adaptive computing (PAC), in which systems learn physical parameters and adapt with fine granularity in the field. We outline an architecture for an adaptation agent and investigate two key aspects of the adaptive process: self-characterization and physical self-optimization. A case study is presented involving random variations in latch reliability. We conducted experiments on a model of a PAC system with physical data obtained from actual field-programmable gate array (FPGA) hardware. Our results show that across 15 benchmark circuits the mean time between failures improved by an average of 30% via low-cost self-adaptation and by 45% assuming assistance from a remote server. Physical self-adaptation and assisted adaptation will both play an important role in achieving computational systems with atomic-scale features.**

Keywords — self-adaptation; variation; sensing; reliability; single event upset; reconfiguration; FPGA

I. INTRODUCTION

For the past several decades, computations have generally been implemented without regard to unique physical variations within individual systems. Efficient computer systems have been created using one-size-fits-all architectures and programs. This separation of bits and atoms has been an essential aspect of the computer revolution, allowing a single design to be incorporated into millions of systems.

The advance of semiconductor technology to nano-scale feature sizes threatens this long-standing paradigm. An emerging and fundamental problem is physical variation. At the atomic scale, manufacturing effects such as random dopant fluctuations ensure that every system component and region is unique. The leading semiconductor roadmap projects that by 2011 the amount of variation will surge, with "3σ" values of delay, power, and leakage reaching 60%, 72%, and 255%, respectively [1]. Furthermore, systems experience temporal variations due to wearout, application activity, and environmental factors. The traditional nonadaptive design paradigm is oblivious to fine-grained variations and is thus leading to systems that are increasingly inefficient, noisy, and subject to early wearout.

A driving question is whether technological systems can feasibly adapt to match spatially unique, time-variant, uncertain physical circumstances. One adaptation approach is for the inherent variations of hardware to be mapped and for each system to be finely customized before being deployed. A proposal for such post-silicon customization is presented in [2], using variation-aware place and route and field-programmable gate arrays (FPGAs). However, there are high costs associated with fine-grained characterization using expensive testers, heavyweight computer-aided design cycles, and the handling of large numbers of unique configurations. Moreover, a pre-deployment approach cannot help with infant mortality, changes in workload, or other parameter shifts that occur in the field. As an example of parameter shifts, a large-scale study of memory reliability in Google data centers found that error rates changed unexpectedly after 10-20 months of operation [3].

In this paper we consider *physically-adaptive computing* (PAC), in which computational systems become finely adapted in the field to their physical circumstances, thereby improving system fitness. Some key research questions include the following. How can the relevant fine-grained parameters be characterized? How can systems generate better physical implementations given very limited system resources? What type of framework is needed to make physically-adaptive computing more practical?

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 including the absence of fine-grained physical variation, the lack of fine-grained reconfigurability, and the lack of tool support. All of these reasons are rapidly disappearing, suggesting a prime opportunity for deploying PAC.

The application domains where PAC is most relevant are those needing optimized levels of efficiency or reliability, combined with reconfigurability. Examples include FPGA-based on-board processing in spacecraft, UAVs, and many other embedded systems [4-6]. Another example domain is high-performance reconfigurable computing, where there are constraints on power, thermal effects, and lifetime reliability. Physical adaptation is relevant not only for today's leading-edge reconfigurable systems, which are built using CMOS technology, but also for future systems using emerging reconfigurable nano-technologies [7].

The contributions of this paper include:

- A conceptual framework for physically-adaptive computing
- A case study of self-adaptation for fine-grained physical variations in reliability
- FPGA-based experimental results showing 30% improvement in reliability using self-adaptation and 45% with assisted adaptation

The remainder of the paper is structured as follows. Background material is provided in Section II along with a survey of related work. In Section III we formulate the physically-adaptive computing problem and outline an adaptation architecture. A case study is introduced in Section IV. Experimental results from an emulated PAC system are presented in Section V followed by a discussion and conclusions in Sections VI and VII.

II. Background

In this section we briefly describe some key concepts and then survey related work.

A. Preliminaries

Parameter variations. Many of the key parameters in a computing system exhibit some form of variation. There are spatial variations in component parameters such as delay, and in operating conditions such as temperature. Spatial variation tends to be either random by component (e.g., for each transistor), spatially-correlated (e.g., covering a region of a chip), or a combination of both. Figure 1 shows spatially-correlated variations in ring oscillator frequency as measured on each of two FPGAs [8]. In addition, systems experience temporal variations caused by wearout or application phase behavior.

Reconfigurable hardware. Certain computing platforms allow for reconfiguration of the circuit components and interconnections. Prime examples include FPGAs which typically contain vast numbers of lookup tables (LUTs) and other logic, surrounded by networks of reconfigurable interconnect. The amount of logic can be enormous, reaching over 1 million LUTs on a single chip. Typically the logic is divided into small groups which we will call *slices*. FPGAs allow many key algorithms to be implemented efficiently and provide fine-grained flexibility in the field; thus they are popular for signal processing, communications, on-board processing, and a widening range of applications. Aside from FPGAs, there is a pronounced trend toward microprocessors with many processor cores; these many-core platforms provide a form of reconfigurability but are relatively coarse-grained. A third type of reconfigurable platform uses nano-electronics, though the technology is still in the research stage [7]. Note that we refer to all of these as reconfigurable platforms, and we refer to a specific physical instance as a *reconfigurable substrate*. Each substrate is considered to be physically unique.

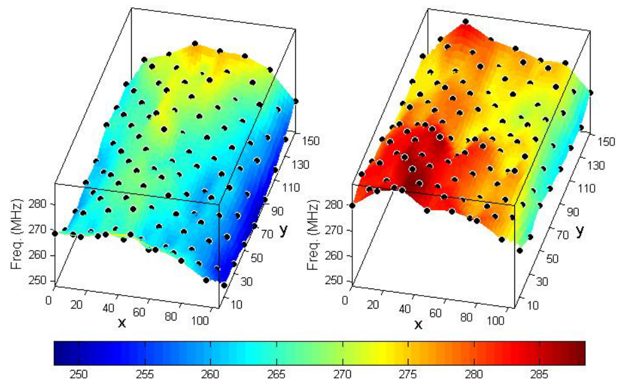


Figure 1. Spatially-correlated variations in ring oscillator frequency on two FPGAs, measured with reconfigurable logic-based sensors [8].

B. Related Work

Physically-adaptive computing is related to, but distinct from, several similar-sounding research areas including adaptive computing, autonomic computing, and cyber-physical systems. Distinguishing features of PAC include a focus on fine-grained physical phenomena and low-level structural adaptation. Adaptive computing research, on the other hand, has typically involved issues such as dynamically switching between previously-generated FPGA configurations, adaptive policies, and application-specific instruction sets. Some works in that area share with PAC a need for in-system implementation of computations [9] or on-board decision making. Examples of cognitive architectures for adaptive computing systems include: a Bayesian network cognitive algorithm [10], a genetic algorithm-based classifier system [11], and a reinforcement learning-based controller [12]. Autonomic computing is a study of self-management and other so-called self-* properties in computing systems [13]. Much of the work is conducted in the logical realm [14,15]. 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 [11] and thermal/power management in data centers [16]. 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 similar issues of sensing and adapting to a substrate prone to wearout/damage [17].

The concept of adapting to fine-grained variations has a connection to the well-studied area of defect handling. Defects are permanent hardware faults, and can be thought of as an extreme type of highly localized variation. There is a long history of research into testing for defects, as well as the use of spare resources to tolerate them. One method is to test for defects at manufacturing time and to permanently swap out a defective section of memory or FPGA logic for a spare one. There have been proposals for in-system handling of FPGA defects via self-test and re-routing using a lightweight routing tool [18-20]. 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 generic implementations and to allow each system to identify the most suitable alternative through trial and error [21-22]. This approach does not scale well for large systems or for fine-grained variations. Several works have focused on self-characterization of physical variation [23]. We proposed methods for measuring correlated variations in FPGA delays, leakage, voltage drop, etc. [8]. Another set of studies has considered variation-aware implementation techniques using signal polarity inversion [24], technology mapping [25], re-placement [2], and logical-physical mapping in a memory array [26]. Note that these techniques depend on accurate characterization data being available. A little preliminary research has addressed the combination of self-characterization and subsequent re-implementation of computations. Some of these studies have focused specifically on the problem of delay variations, such as in FPGAs [27] or in a post-CMOS reconfigurable technology [7]. We recently conducted a preliminary study of this kind in the context of reliability variations, but did not analyze actual circuits in that work [28].

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.

III. PAC Framework

In this section we present a high-level formulation of the adaptation problem, and then outline an architecture for adaptation.

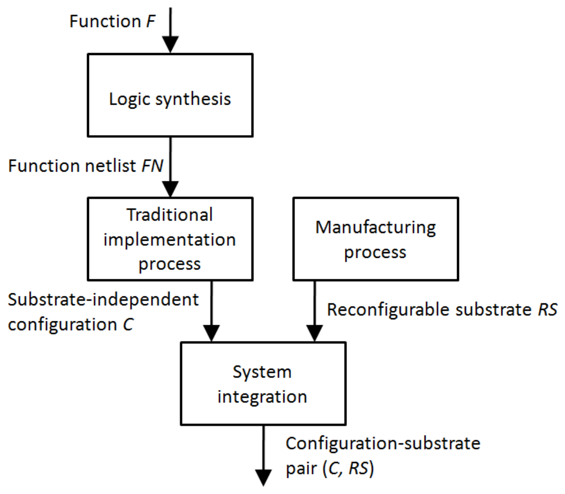
A. *Problem Formulation*

A physically-adaptive computing system consists of the following key elements: a reconfigurable substrate, a set of computational tasks, a set of parameters, and a process for adaptation.

Substrate. We define a *reconfigurable substrate RS* as a physical object able to support computation, such as an individual FPGA chip. The substrate contains configurable physical elements such as switches, memory cells, and interconnect. 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.

Computational tasks. A *computational function F* defines an application task. It can be expressed in various ways including at the behavioral or register-transfer level (RTL) 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 design flows, this is a netlist generated by a synthesis tool. A *configuration C* of a reconfigurable substrate specifies how to configure and connect the physical elements in order to implement a function netlist. 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 $C_1 \rightarrow C_2$. Computations of the function *F* are performed by a *configuration-substrate pair* (*C*, *RS*). Figure 2 illustrates a basic flow that occurs in the design and implementation of computations. Although such a flow is sometimes associated with FPGA "hardware" development, it is clear that with reconfigurable substrates the configuration can be considered a type of low-level software (sometimes called firmware, configware, etc.). In fact, the framework is general enough that it could be readily modified for "software"-centric systems, e.g., when computations are expressed as instruction-based programs.

Parameters. A configuration-substrate pair (*C*, *RS*) has an associated set of *parameters* $\{Par_1, Par_2 \ldots\}$. Parameters can be time-varying, for instance due to wearout. We separate the parameters into three classes: those that are specific to the substrate, the implemented logic, and 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 computer engineering would be sufficient. However, uncertainty is growing and is an inevitable 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, and uneven stress during operation. *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) and toggle rates. Logical parameters can exhibit high levels of variation, such as a 30x range of logical fanout [7], and order of magnitude differences in error propagation probabilities [29]. *Operational parameters* are those associated with the operation of a computing system. These emerge from the act


Function F
Logic synthesis
Function netlist FN
Traditional implementation process
Manufacturing process
Substrate-independent configuration C
Reconfigurable substrate RS
System integration
Configuration-substrate pair (C, RS)
Figure 2. Conceptual view of a conventional system design flow

of computing and depend upon the interaction of logical, substrate, and/or external environmental factors. Examples include power consumption, temperature, and fault/error rates. Operational parameter variations such as hot spots increase the need for physical adaptation. Certain operational parameters are also important for assessing the overall fitness of a system.

The *fitness* or *cost* of a configuration-substrate pair is a weighted function of the associated parameters:

$$Cost\,(C, RS) = f\,(Par_1, Par_2, \ldots)$$

In the conventional approach, cost functions do not account for the substrate. For example, Xilinx uses the following substrate-independent weighted cost function to evaluate different implementations [30]:

$$Cost\,(C, *) = w_1 \times (\text{wire length}) + w_2 \times (\text{timing}) + w_3 \times (\text{power})$$

A *physically-adaptive computing (PAC) system* is one in which parameters are learned and estimated over time and computational configurations are adapted to suit the physical landscape, leading to improved fitness. PAC design challenges include specifying the cost function, developing methods of characterization and re-implementation, and enabling autonomous adaptation.

B. Architecture for Physical Adaptation

We now describe some of the requirements for a PAC process, and outline an architecture for an agent controlling that process. In the domain of intelligent control, including much of autonomic computing, there is often an agent or controller engaged in a loop of sense-analyze-decide-act [13]. The PAC problem differs from many of the related problems in important ways. First, there are more extensive requirements for *active sensing*. The agent must decide which of many parameters to characterize, and take actions to obtain data, for instance by initiating an appropriate self-test procedure. This contrasts with systems in which feedback data is always available (automatic sensing). Second, the agent process only runs occasionally rather than continuously, in order to match the time scale of parametric shifts. This time scale can be much longer than the times for computational tasks or loading a configuration. Third, some portions of the physical adaptation process can require more resources than are available in the system.

This characteristic of needing 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 compute 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. Remote execution is well-understood in more abstract contexts such as cyber-foraging [31]. Now there are several indications that such an approach holds promise in the PAC context. First, there is an ongoing proliferation of network-connected embedded systems, and recent advances in cloud computing. For instance frameworks are being developed for evaluating the energy efficiency of local vs. remote execution [32]. Second, preliminary studies such as one by Hyder *et al*. [21] illustrate remote execution for a type of physical adaptation, using peers instead of a server. Third, missions such as the Cibola Flight Experiment have demonstrated that many of the pieces are already in place [33]. 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 our approach to assisted adaptation.

Given the requirements of PAC just mentioned, the main actions available to an agent can be categorized as follows. 1) Characterization. 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) Re-implementation. 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 [24] 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.

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 below, in which the focus is mainly on methods of characterization and re-implementation. A promising area for future work is to study the extent to which the decision procedure itself can be made adaptive. As mentioned in Related Work, a variety of cognitive algorithms have been proposed for use in adaptive computing systems [10-12].

A generic PAC system is sketched in Figure 3. 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 (such as a PowerPC or planned ARM core [34] in Xilinx FPGAs), or even on a soft core implemented in the reconfigurable logic. The main outputs of the agent process are the physically-adapted configurations associated with the computational tasks.

A simplified flow of the adaptation process is shown in Figure 4. In contrast with some previous flows of this type [20], the process need not be a linear progression of characterization-implementation-reconfiguration. The agent may need to repeat certain types of actions, schedule actions to be performed remotely, and juggle multiple overlapping adaptation processes.

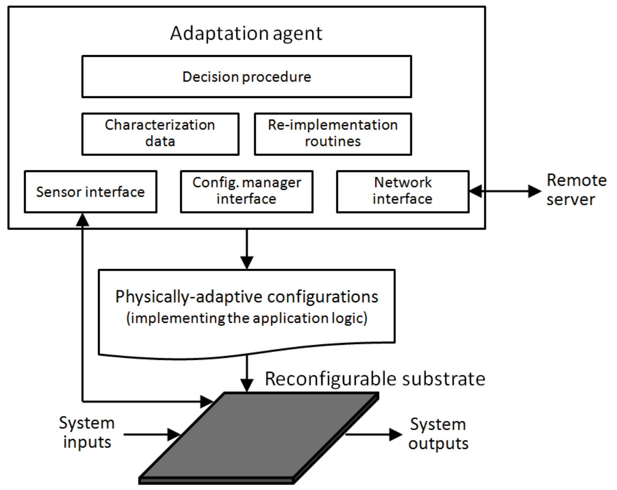


Figure 3. Sketch of a PAC system. The adaptation agent process runs on a processor in the system.

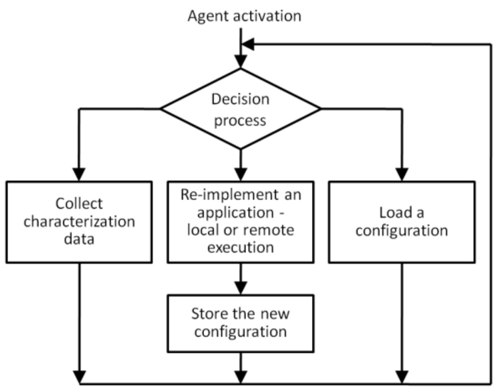

Agent activation
Decision process
Collect characterization data
Re-implement an application - local or remote execution
Load a configuration
Store the new configuration

Figure 4. 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.

IV. Case Study: Adaptation for Reliability

In this section we present an example of PAC in the context of transient faults.

A. *Variations in Reliability*

Transient faults in digital systems are one-time events that upset the state of a node. The resulting incorrect machine state is called a soft error, which can potentially propagate and become a system-level error [29]. Transient faults can be triggered by particle radiation, thermal noise [35], and other mechanisms.

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 and are often called *upsets*. Latches are typically used in a master-slave pair that forms a flip-flop. There are four main cases of upsets corresponding to the master latch in the 0 or 1 data state (abbreviated $m0$ and $m1$), and the slave latch in the 0 or 1 data state ($s0$ and $s1$). The master and slave latches generally have opposite 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. Due to random variations, each latch can have a different inherent susceptibility to transient faults. We refer to this property as *upsetability*, and define it as the probability of a latch incurring an upset in a specific noise environment. Upsetability values will often be normalized to a mean; for instance, a latch with twice the mean number of upsets will have a value of 2.0.

Whereas upsets in a memory array or a static configuration bit can often be efficiently detected and corrected without any system-level failures, upsets in latches are more difficult to handle. For further details regarding the importance of latch upsets and variations, see [28].

B. *Problem Statement*

We will now state the problem using the preceding PAC formalism. We start by assigning the function F to be each of a set of benchmarks functions (circuits) from the IWLS2005 OpenCores suite [36]. 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 RS_1 and RS_2 which are instances of a modern FPGA with 65nm 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 which 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 C has an associated matrix $\mathbf{x}$ which 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 estimated fault rate of a bit/flip-flop pair ij is essentially the fraction of time spent in a state times the fault rate associated with that state, summed over the four type of faults. 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 estimated fault rate:

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

We define the cost function for configuration-substrate pair (C, RS) to be the estimated total fault rate. Assuming that faults affect single latches, this is the sum of the component fault rates. The estimated MTBF is the inverse of the total fault rate, which is:

$$Cost(C, RS) = \sum_i \sum_j cost_{ij} x_{ij} \quad (2)$$

The goal for the PAC system is to characterize the necessary parameters and generate new configuration-substrate pairs (C, RS) that minimize cost (maximize fitness). A description of our experiments and results will be given next.

V. Experimental Results

We created an experimental FPGA-based system to test PAC principles, methods, and overhead. The system is capable of performing self-characterization of latch upsetability 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.

A. Self-Characterization

Our characterization method involves the autonomous injection of small amounts of charge into a latch via short pulses on the asynchronous set and reset lines. For a full description of this method see [28]. Self-characterization was performed for each of 1,024 flip-flops in a 32×32 array and for both substrates. For every flip-flop, the system characterized the four values of transient fault upsetability for the given noise environment. As can be seen with the $s1$ upsetability in Figure 5, significant variations were found. Each cell in the image corresponds to a flip-flop, and each vertical group of four cells corresponds to a slice.

The memory overhead for the characterization data and code (written in C for the MicroBlaze) was 64KB. The time required for characterization was previously found to be 3 minutes for 256 slices [28]. The vast majority of that time was spent on calibration of the noise levels for each slice.

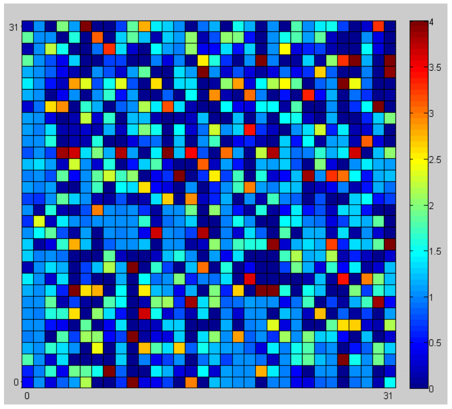


Figure 5. Random spatial variations in latch upsetability in a 32x32 array as measured via self-characterization. Shown are the ratios of $s1$ upsetability (slave latch, data = 1) to the mean value in the local slice, for substrate RS_1.

Slices were calibrated individually in an effort to reduce any experimental effects (e.g., voltage "droops") and thus improve accuracy. Although this calibration overhead scales linearly with the number of slices, it may be prohibitive for many applications. Areas for future study include minimizing the need for calibration and developing parallel methods that are more scalable.

Note 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.

B. 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 slices are shown in Table I. 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.

The cost of these original, substrate-independent implementations was calculated. This was done by extracting the slice coordinates and flip-flop sites of all state bits (using Xilinx Floorplanner), and then evaluating (2). Next, an adapted implementation with the lowest cost was found. The method used was local re-placement of the state bits within each slice. Since there are four flip-flop locations and up to four state bits that need to be placed, there are up

TABLE I. SYNTHESIZED BENCHMARK CIRCUITS

Benchmark circuit	No. of state bits	No. of LUTs	No. of slices	No. of slices w/ >0 state bits	Intra-slice 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

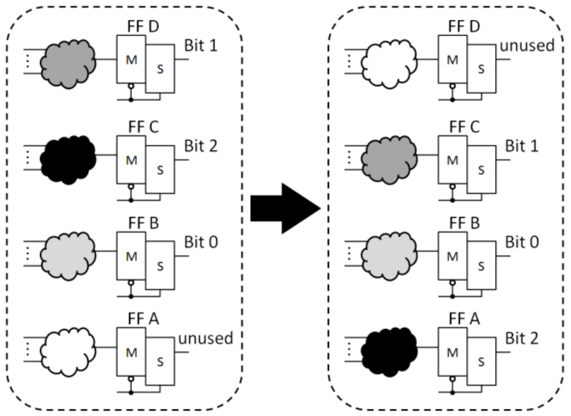


Figure 6. A slice of logic in its original configuration with state bits (0,1,2) placed at master-slave flip-flops B,D,C (left). The same slice after local re-ordering of logic (right).

to 4! = 24 alternative placements for each slice. Our re-implementation tool performed the variation-aware optimization within each of the 256 slices, needing only 256×24 evaluations of equation (2). The new total mean time between latch failures (MTBF) was then calculated by summing over all slices.

An example of intra-slice optimization is shown in Figure 6. Three state bits are placed among the four flip-flops in a slice. 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 slices was modified to achieve better balance. For instance, if 7 state bits were originally packed into 3 slices using 4, 2, and 1 flip-flops, our tool would re-pack the logic such that the distribution became 3, 2, and 2. This balancing decreases the utilization inside the slices 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 slices containing utilized flip-flops was held constant so as not to increase circuit area. Intra-slice 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 results are shown in Figure 7. 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%.

One circuit, des3_area, showed no improvement in this experiment. This is because the circuit is a regular datapath and leads to slices 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 slices, as shown in the right column of Table I, the gains increase. The least dense circuit, *spi*, saw improvements as high as 77%.

C. *Experiment 2: Physical and Logical 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-slice re-placement), or relied on assisted adaptation, in which case the aforementioned re-packing was performed along with intra-slice 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 10 trials, 15 benchmarks, and 2 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 8. 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%.

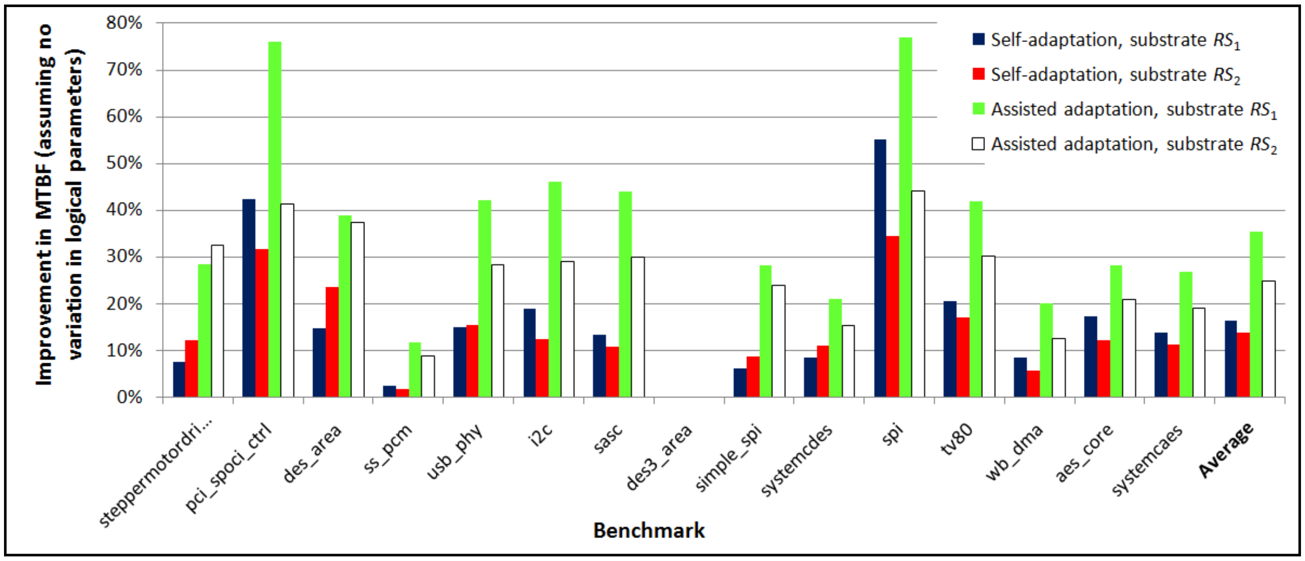


Figure 7. Results of experiment 1 showing the improvements in MTBF for self-adaptation and assisted adaptation relative to the non-adaptive case, assuming no variation in signal probabilities.

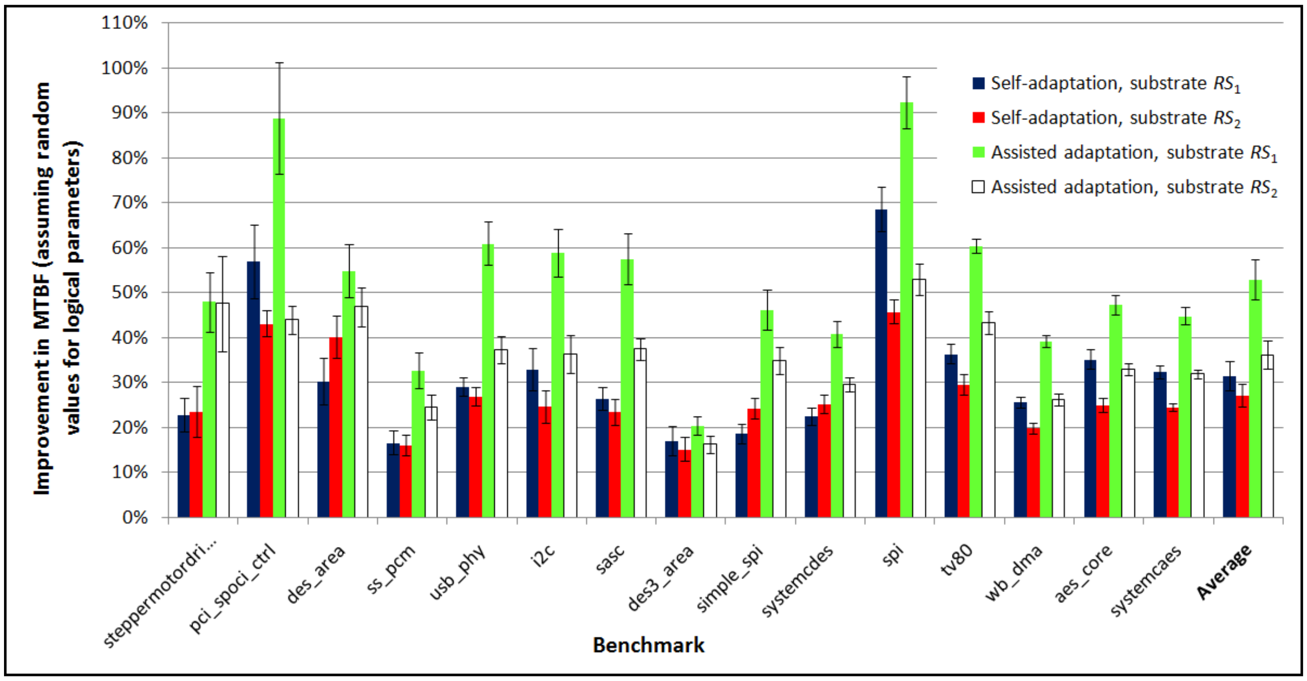


Figure 8. Results of experiment 2 showing the improvements in MTBF for self-adaptation and assisted adaptation relative to the non-adaptive case, assuming uniformly random signal probabilities. Error bars show the standard deviation across 10 trials.

VI. Discussion

The results illustrate some of the potential for physically-focused adaptation. One observation is that the value of PAC depends highly on the amount of variation in both logical and physical parameters. For instance, substrate RS_1 exhibits more random variation in latch reliability than does RS_2; the coefficient of variation is roughly 20% higher in relative terms [28]. As a consequence, the PAC benefits are generally larger when using RS_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%).

The potential for adaptation 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.

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

Note that the case study involved reducing the raw number of transient faults, a strategy called fault avoidance. Faults may or may not lead to a system-level error depending on error propagation. Thus there is an opportunity for improving reliability even further by leveraging variation in the error propagation probability [37]. A future challenge will be finding new ways to balance the variations of the many interacting system parameters.

Much work remains to be done to realize the vision of PAC. One limitation of the presented case study is the assumption that all logic can be locally 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 Altera) have announced new and more complete support for partial reconfiguration [38][39]. As physical issues come to the fore, vendor plans for additional fine-grained methods [20] are likely to come to fruition. A final limitation of the case study was that it involved optimization toward a single objective. In reality, there are typically multiple conflicting objectives (power, reliability, design effort, etc.) that require careful trade-offs.

The area of physically-adaptive systems has many interesting and open research questions involving generalized frameworks for adaptation, fusing together multiple types of characterization data, and how best to combine self- and assisted adaptation.

VII. CONCLUSIONS

This paper has demonstrated some of the rationale, requirements, and potential for physically-adaptive computing. The case study provides an example of self-characterization of two unique substrates, and the use of physical data to generate better implementations. Reliability was seen to improve by an average of 30% via self-adaptation and 45% via assisted adaptation. The PAC paradigm was illustrated here using modern FPGAs (using 65nm technology); based on the leading projections, its importance can be expected to increase for technologies on the near horizon. As feature sizes approach the atomic scale, fine-grained self- and assisted adaptation are likely to play important roles in technological systems.

ACKNOWLEGMENT

We would like to thank: Joseph Greathouse, Andrea Pellegrini, Chien-Chih Yu and the Adaptive Hardware & Systems Group for helpful comments; Xilinx Inc. and Sun Microsystems for equipment donations. This work was supported in part by a NASA GSRP Fellowship and by the National Science Foundation under grant CCF-0702276.

REFERENCES

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

[2] 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.

[3] 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.

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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] B. Schmerl and D. Garlan, "Exploiting architectural design knowledge to support self-repairing systems," *Int'l Conf. Software Engineering and Knowledge Engineering*, pp. 241-248, 2002.

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

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

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

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

[19] 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.

[20] 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.

[21] Z. Hyder and J. Wawrzynek, “Defect tolerance in multiple-FPGA systems,” *Int’l Conf. Field Programmable Logic and Applications*, pp. 247-254, Aug. 2005.

[22] 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.

[23] J.S.J. Wong, P. Sedcole and P.Y.K. Cheung, “Self-characterization of combinatorial circuit delays in FPGAs,” *Proc. Field-Programmable Technology*, pp. 17-23, Dec. 2007.

[24] K. Zhu, “Post-route LUT output polarity selection for timing optimization,” *Proc. Int’l Symp. Field Programmable Gate Arrays*, pp. 89-96, 2007.

[25] 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.

[26] 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.

[27] 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.

[28] 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*, Aug. 2010.

[29] 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.

[30] Xilinx Inc., “Optimizing FPGA power with ISE design tools,” Xcell Journal, Second Quarter 2007.

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

[32] 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.

[33] 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.

[34] Xilinx Inc., “Xilinx architects ARM-based processor-first, processor-centric device,” Xcell Journal, Second Quarter 2010.

[35] 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.

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

[37] K.M. Zick and J.P. Hayes, “On-line characterization and reconfiguration for single event upset variations,” *Proc. Int’l On-Line Testing Symposium*, pp. 243-248, June 2009.

[38] Xilinx Inc., http://www.xilinx.com.

[39] Altera Corp., http://www.altera.com.