Architecting a Reliable CMP Switch Architecture

KYPROS CONSTANTINIDES, STEPHEN PLAZA, JASON BLOME,
VALERIA BERTACCO, SCOTT MAHLKE, and TODD AUSTIN
University of Michigan
and
BIN ZHANG and MICHAEL ORSHANSKY
University of Texas at Austin

As silicon technologies move into the nanometer regime, transistor reliability is expected to wane as devices become subject to extreme process variation, particle-induced transient errors, and transistor wear-out. Unless these challenges are addressed, computer vendors can expect low yields and short mean-times-to-failure. In this article, we examine the challenges of designing complex computing systems in the presence of transient and permanent faults. We select one small aspect of a typical chip multiprocessor (CMP) system to study in detail, a single CMP router switch. Our goal is to design a *BulletProof* CMP switch architecture capable of tolerating significant levels of various types of defects. We first assess the vulnerability of the CMP switch to transient faults. To better understand the impact of these faults, we evaluate our CMP switch designs using circuit-level timing on detailed physical layouts. Our infrastructure represents a new level of fidelity in architectural-level fault analysis, as we can accurately track faults as they occur, noting whether they manifest or not, because of masking in the circuits, logic, or architecture. Our experimental results are quite illuminating. We find that transient faults, because of their fleeting nature, are of little concern for our CMP switch, even within large switch fabrics with fast clocks. Next, we develop a unified model of permanent faults, based on the time-tested bathtub curve. Using this convenient abstraction, we analyze the reliability versus area tradeoff across a wide spectrum of CMP switch designs, ranging from unprotected designs to fully protected designs with on-line repair and recovery capabilities. Protection is considered at multiple levels from the entire system down through arbitrary partitions of the design. We find that designs are attainable that can tolerate a larger number of defects with less overhead than naïve triple-modular redundancy, using

This article extends an earlier version that appeared in the 12th International Symposium on High Performance Computer Architecture (HPCA-12), February 2006.
This work was supported in part by grants from NSF and Gigascale Systems Research Center.
Authors' addresses: Kypros Constantinides, Stephen Plaza, Jason Blome, Valeria Bertacco, Scott Mahlke, and Todd Austin, Advanced Computer Architecture Lab, University of Michigan, Ann Arbor, MI 48109; email: {kypros, splaza, jblome, valeria, mahlke, austin}@umich.edu.
Bin Zhang and Michael Orshansky, Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, 78712; email: bzhang@ece.utexas.edu, orshansky@mail.utexas.edu.

domain-specific techniques, such as end-to-end error detection, resource sparing, automatic circuit decomposition, and iterative diagnosis and reconfiguration.

Categories and Subject Descriptors: B.8.1 [**Hardware**]: Performance and Reliability—*Reliability, Testing, and Fault-Tolerance*

General Terms: Reliability, Design

Additional Key Words and Phrases: reliability, defect-tolerance, CMP switch

ACM Reference Format:

Constantinides, K., Plaza, S., Blome, J., Bertacco, V., Mahlke, S., Austin, T., Zhang, B., and Orshansky, M. 2007. Architecting a reliable CMP switch architecture. ACM Trans. Architec. Code Optim. 4, 1, Article 2 (March 2007), 37 pages. DOI = 10.1145/1216544.1216545 http://doi.acm.org/10.1145/1216544.1216545.

1. INTRODUCTION

A critical aspect of any computer design is its reliability. Users expect a system to operate without failure when asked to perform a task. In reality, it is impossible to build a completely reliable system. Consequently, vendors target design failure rates that are imperceptibly small [Siewiorek and Swarz 1998]. Moreover, the failure rate of a population of parts in the field must exhibit a failure rate that does not prove too costly to service. The reliability of a system can be expressed as the mean-time-to-failure (MTTF). Computing system reliability targets are typically expressed as failures-in-time (FIT), or FIT rates, where one FIT represents one failure in a billion hours of operation.

In many systems today, reliability targets are achieved by employing a fault-avoidance design strategy. The sources of possible computing failures are assessed, and the necessary margins and guards are placed into the design to ensure it will meet the intended level of reliability. For example, most transistor failures (e.g., gate-oxide breakdown) can be reduced by limiting voltage, temperature, and frequency [J. E. D. E. Council 2002]. While these approaches have served manufacturers well for many technology generations, many device experts agree that transistor reliability will begin to wane in the nanometer regime. As devices become subject to extreme process variation, particle-induced transient errors, and transistor wear-out, it will likely no longer be possible to avoid these faults. Instead, computer designers will have to begin to directly address system reliability through fault-tolerant design techniques.

Figure 1 illustrates the fault-tolerant design space we focus on in this article. The horizontal axis lists the type of device-level faults that systems might experience. The source of failures are widespread, ranging from transient faults because of energetic particle strikes [Ziegler 1996] and electrical noise [Vrudhula et al. 2002], to permanent wear-out faults caused by electromigration [Hu and Rosenberg 1999], stress-migration [J. E. D. E. Council 2002], and dielectric breakdown [Wu et al. 2002]. The vertical axis of Fig. 1 lists design solutions to deal with faults. Design solutions range from ignoring any possible faults (as is done in many (current systems), to detecting and reporting faults, to detecting and correcting faults, and finally fault correction with repair capabilities. The last two rows of the table are the only solutions that can address permanent faults, with the solution of the last row being the only approach that

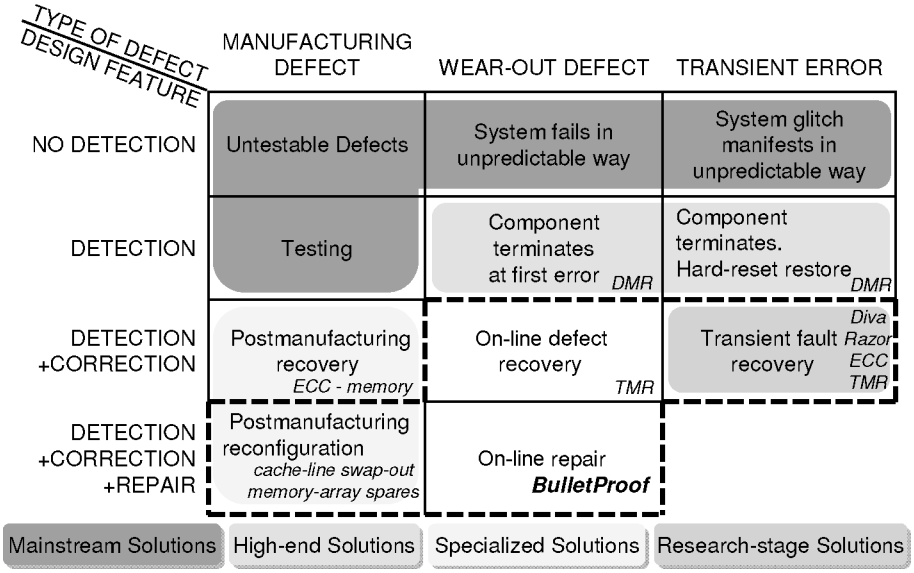


Fig. 1. *Reliable system design space*. The diagram shows a map of type of device-level faults in a digital system (horizontal axis) versus protection techniques against these faults (vertical axis). This work addresses the problems/solutions in the dash bordered area of the map.

maintains efficient operation after encountering a silicon defect. This project, called *BulletProof*, addresses the problems/solutions in the dash bordered area of the table.

In recent years, industry designers and academics have paid significant attention to building resistance to transient faults into their designs. A number of recent publications have suggested that transient faults, because of energetic particles, in particular, will grow in future technologies [Borkar et al. 2004; Mukherjee et al. 2005]. A variety of techniques have emerged to provide a capability to detect and correct these type of faults in storage, including parity or error correction codes (ECC) [Siewiorek and Swarz 1998], and logic, including dual or triple-modular spatial redundancy (DMR/TMR)[Siewiorek and Swarz 1998] or time-redundant computation [Smolens et al. 2004; Reinhardt and Mukherjee 2000; Mukherjee et al. 2002; Rotenberg 1999; Gomaa et al. 2003] or checkers, like DIVA [Weaver and Austin 2001]. Additional work has focused on the extent to which circuit timing, logic, architecture, and software are able to mask out the effects of transient faults, a process referred to as "derating" a design [Mukherjee et al. 2003; Wang et al. 2004].

In contrast, little attention has been paid to incorporating design tolerance for permanent faults, such as silicon defects and transistor wear-out. The typical approach used today is to reduce the likelihood of encountering silicon faults through postmanufacturing burn-in, a process that accelerates the aging process as devices are subjected to elevated temperature and voltage [Wu et al. 2002]. The burn-in process accelerates the failure of weak transistors, ensuring that, after burn-in, devices still working are composed of robust transistors. In addition, many computer vendors provide the ability to repair faulty

memory and cache cells, via the inclusion of spare storage cells [Spainhower and Gregg 1998; Bossen et al. 2002]. Recently, academics have begun to extend these techniques to support sparing for additional on-chip memory resources, such as branch predictors [Bower et al. 2004] and registers [Shivakumar et al. 2003].

1.1 Contributions of This Article

In this article, we emphasize the role of understanding in reliable microarchitecture design by performing a comprehensive design study of the effects of transient and permanent faults on a chip-multiprocessor switch design. The goal is to better understand the nature of faults, and to build into our designs a cost-effective means to tolerate these faults. Specifically, we make the following contributions:

- *We develop a high-performance and high-fidelity fault-modeling infrastructure.* We present two different setups for the infrastructure: one to evaluate the effects of transient faults and one for permanent faults. The fault-modeling infrastructure is sufficiently accurate to model asynchronous fault injection into a design at the transistor level, and then fully model the many possible ways that faults can be masked, through timing, logic, or microarchitectural effects.
- *We develop a high-level architect-friendly model of silicon failures, based on the time-tested bathtub curve.* The bathtub curve models the early-life failures of devices during burn-in, the infrequent failure of devices during the part's lifetime, and the breakdown of devices at the end of their normal operating lifetime. From this bathtub-curve model, we define the design space of interest, and fit previously published device-level reliability data to the model.
- *We introduce a low-cost chip-multiprocessor (CMP) switch-router architecture that incorporates system-level checking and recovery, component-level fault diagnosis, and spare-part reconfiguration.* Our design, called the *BulletProof* CMP switch, is capable of tolerating silicon defects, transient faults, and transistor wear-out. We evaluate a variety of BulletProof switch designs, and compare them to designs that utilize traditional fault-tolerance techniques, such as ECC and triple-modular redundancy. We find that our domain-specific fault-tolerance techniques are significantly more robust and less costly than traditional generic fault-tolerance techniques.

The remainder of this article is organized as follows. Section 2 highlights the important failure mechanisms for future process technologies. Section 3 presents our fault-simulation infrastructure for both transient and permanent faults and the statistical models used. Section 4 describes the baseline CMP switch architecture. Section 5 examines the exposure of the baseline design to transient faults while Section 6 evaluates the reliability of the baseline design for permanent faults. Section 7 introduces the techniques we have employed in our CMP switch designs to provide cost-effective tolerance of transient and permanent faults. In Section 8, we present a detailed trade-off analysis of the

resilience and cost of our CMP switch designs, plus a comparison to traditional fault-tolerant techniques, such as ECC and triple-modular redundancy (TMR). Finally, Section 9 gives conclusions and suggestions for future research directions.

2. AN ANALYSIS OF THE FAULT LANDSCAPE

As silicon technologies progress into the 65-nm regime and below, a number of failure factors rise in importance. In this section, we highlight these failure mechanisms and discuss the relevant trends for future process technologies.

2.1 Single-Event Upset (SEU)

There is growing concern about providing protection from soft errors caused by charged particles (such as neutrons and alpha particles) that strike the bulk silicon portion of a die [Ziegler 1996]. SEU events are caused when high-energy particles strike the P-N junction of a MOSFET, which generates a current pulse in the depletion region, resulting in a temporary loss of charge stored on the P-N junction. If the charge variation in the P-N junction is enough to alter the logic value corresponding to the voltage across the junction, a single-event upset (SEU) results. The final effect is a logic glitch that can potentially corrupt combinational logic computation or state bits. While a variety of studies have been performed that demonstrate the unlikeliness of such events [Weaver et al. 2004; Wang et al. 2004], concerns remain in the architecture and circuit communities. This concern is fueled by the trends of reduced supply voltage and increased transistor budgets, both of which exacerbate a design's vulnerability to SEU.

2.2 Process Variation

Another reliability challenge designers face is the design uncertainty that is created by increasing process variations. Process variations result from device dimension and doping concentration variation that occur during silicon fabrication. These variations are of particular concern, because their effects on devices are amplified as device dimensions shrink [Rao et al. 2003], resulting in structurally weak and poor performing devices. Designers are forced to deal with these variations by assuming worst-case device characteristics (usually, a three-sigma variation from typical conditions), which leads to overly conservative designs.

2.3 Manufacturing Defects

Deep submicron technologies are increasingly vulnerable to several fabrication-related failure mechanisms. For example, step coverage problems that occur during the metalization process may cause open circuits. A postmanufacturing [Murray and Hayes 1996] and a built-in self-test (BIST) [Al-Asaad and Hayes 1995] are two techniques to impress test vectors onto circuits in order to identify manufacturing defects. A more global approach to testing for defects is taken by IDDQ testing, which uses on-board current monitoring to detect short-circuits in the manufactured part. During IDDQ testing, any abnormally high current

spikes found during functional testing are indicative of short-circuit defects [Bohl et al. 1997].

2.4 Time-Dependent Wear-out

Technology scaling has adverse effects on the lifetime of transistor devices, as a result of time-dependent wear-out. There are three major failure modes for time-dependent wear-out: electromigration, hot-carrier degradation (HCD), and time-dependent oxide breakdown. Electromigration results from the mass transport of metal atoms in chip interconnects. The trends of higher current density in future technologies increases the severity of electromigration, leading to a higher probability of observing open and short-circuit nodes over time [Gupta and Kahng 2003]. HCD is the result of carriers being heated by strong electrical fields and, subsequently, being injected into the gate oxide. The trapped carriers cause the threshold voltage to shift, eventually leading to device failure. HCD is predicted to worsen for thinner oxide and shorter channel lengths [Ionescu et al. 2002]. Time-dependent oxide breakdown results from the extensive use of ultrathin oxide for high performance. The rate of defect generation in the oxide is proportional to the current density flowing through it and, therefore, it is increasing drastically as a result of relentless down-scaling [Stathis 2002].

2.5 Transistor Infant Mortality

Scaling has had adverse effects on the early failures of transistor devices. Traditionally, early transistor failures have been reduced through the use of burn-in. The burn-in process utilizes high voltage and temperature to accelerate the failure of weak devices, thereby ensuring that parts that survive burn-in only possess robust transistors. Unfortunately, burn-in is becoming less effective in the nanometer regime, as deep submicron devices are subject to thermal run-away effects, where increased temperature leads to increased leakage current and increased leakage current leads to yet higher temperatures. The end result is that aggressive burn-in will destroy even robust transistors. Consequently, vendors may soon have to relax the burn-in process, which will ultimately lead to more early failures for transistors in the field.

The quantity and diversity of faults makes the task of protecting against them seem, on the surface, a daunting task. However, as we will demonstrate in the following sections, we can conveniently bifurcate all faults into two classes: transient and permanent. Given these two abstractions, we need only determine an appropriate model for fault-arrival rates, and develop the infrastructure necessary to evaluate the effects of these faults.

3. A FAULT IMPACT-EVALUATION INFRASTRUCTURE

In this section, we present an accurate simulation infrastructure for evaluating a design's exposure to both transient and permanent faults. Again, we attack the problem by partitioning all faults into two classes: transient and permanent. Within each partition, we create an appropriate model for the relevant fault's

parameters, and develop the infrastructure necessary to evaluate their effects on a given design.

Reliable system design analysis places high demands on the analysis infrastructure. The simulation framework must accurately gauge detailed circuit phenomena to correctly simulate the introduction, propagation, and possible masking of faults, as they enter into the system under study. For example, a transient SEU fault, when injected into a combinational logic block, may or may not affect the latch at the end of the associated logic chain; whether or not this happens is purely because of the timing of the glitch's propagation and the arrival time of the clock at the associated latch inputs. The approach that has been taken in the past to analyze much of the work in reliable system design has been to utilize extremely simplistic analytical circuit models of microarchitectural components. The primary advantage of the analytical circuit models is flexibility and speed. However, the accuracy of these models is quite poor as they cannot capture the timing and logic-masking effect that are important in tracking fault propagation in complex computing systems. To address these concerns, our evaluation framework incorporates an event-driven logic simulator, permitting simulations that react to circuit-level reliability phenomena (such as transient, delay, and permanent faults) on a cycle-by-cycle basis while simulating, with sufficient speed, to examine entire workloads.

3.1 Simulation Methodology for Transient Faults

Figure 2a shows the simulation framework for evaluating the impact of transient faults on a digital design. We use an event-driven simulator to simulate in parallel two copies of the structural gate-level description of the design under evaluation (which we obtain by synthesizing a Verilog description of the design with the Synopsys CAD tools): one copy is kept intact (golden model), while the other is subjected to fault injection. The input stimuli used for the simulation consist of typical input traces that exercise the design.

The fault generator is capable of injecting voltage pulses of various durations at any gate of the design and flipping the value of any individual flip-flop of the design by forcing values on the design nets and latches during the simulation. Faults are uniformly distributed in time (when they occur), space (which net they affect), and duration of the event. At the end of each cycle, all outputs and sequential elements of the design are compared with the golden model. The fault analyzer distinguishes four cases: (1) if a mismatch is detected in the output lines, then an error has occurred; (2) if the mismatch is found in the sequential elements but not at the outputs, then the fault was microarchitecturally masked; we derive that all the other injected faults have been either (3) time masked or (4) logic masked. To discern between these two final cases, we rerun the simulation by injecting the same faults synchronized with the design's clock cycle using a one-cycle duration: the faults injected in this second simulation cannot be time-masked. Hence, the difference between these two analyses gives the correct partition between time- and logic-masked faults. Finally, to gain statistical confidence of the results, we run the simulations described for a thousand times in a Monte Carlo modeling framework.

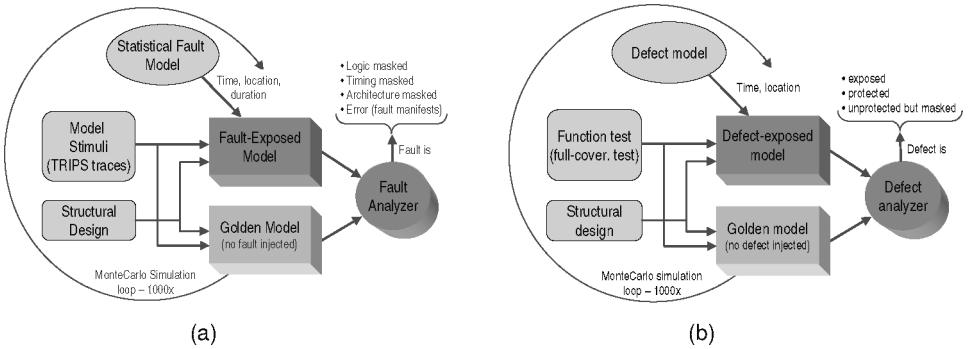


Fig. 2. Simulation infrastructure. (a) The simulation infrastructure to evaluate the impact of transient faults. Two models of the system are simulated in parallel: one is subjected to faults (uniformly distributed in time, space, and duration), while the other is kept intact. We provide realistic data traces to the design and evaluate, at the end of each clock cycle, if a fault has caused an error at the outputs. (b) The simulation infrastructure for permanent faults. The defect infrastructure once again uses two models of the system, simulated in parallel. Defects are uniformly distributed in time and space and the input stimuli is a full-coverage test that activates each internal circuit node of the system. A defect analyzer classifies faults based on the system response.

Soft errors can be caused by high-energy neutrons present in cosmic rays and alpha particles originated from radioactive decay of impurities in chip and packaging materials [Ziegler et al. 1996; Shivakumar et al. 2002]. Because of different packaging technologies, the soft-error rate caused by alpha particles can vary widely for processors within a particular technology generation. Further, the soft-error rate from these two radiation sources is additive and, thus, each one can be studied independently. Therefore, our transient fault model considers only transient faults caused by high-energy neutrons present in cosmic rays. The soft-error rate (SER) is determined not only by the flux rate of the high-energy neutrons, but also by the circuit elements' susceptibility to the particle strike:

$$SER \sim F \times exp\left(- \frac{Q_{crit}}{Q_s} \right)$$

where F is the flux rate of the high-energy neutrons, Q_s is a technology-dependent constant, and Q_{crit} is the critical charge, a measure of the circuit element's susceptibility to the particle strike. Q_{crit} and Q_s both decrease with feature size, but Q_{crit} decreases faster than Q_s [Shivakumar et al. 2002].

In our simulation framework, we use a pulse-based model for transient faults where the transient voltage pulses are classified into five classes based on their duration. The model uses a sixth class of faults to model the flip of a flip-flop's value, when the flip-flop is hit directly by an energetic particle strike. The transient fault generation is modeled using a six-variable random process, where random variables model the uniformly distributed arrival rate of each class of faults. For both current and future technology processes, the mean interarrival times for each class of faults were derived by data in Shivakumar et al. [2002] and by using accurate SPICE simulations to characterize Q_{crit} of various circuit elements.

3.2 Simulation Methodology for Permanent Faults

The simulation of permanent faults, or defects, is carried on a corresponding framework as shown in Fig. 2b, with three main differences. First, the defect generator considers only the time and location of defects. The duration of the event in this analysis is considered permanent. Once a permanent failure occurs, the design may or may not continue to function, depending on its internal structure and architecture. Second, the defect analyzer classifies each defect as (1) exposed, (2) protected, or (3) unprotected, but masked. In the context of defect evaluation, faults accumulate over time until when the design fails to operate correctly. The defect that brings the system to failure is the last injected defect in a simulation and it is classified as exposed. A defect may be protected if, for instance, it is the first one to hit a TMR-protected component. An unprotected, but masked defect, is one such that is masked only because it occurs in a portion of the design that has already failed and thus further failures in this region become innocuous. An example would be a defect hitting an already failed module of a TMR-protected component.

In the context of defects, we are concerned with studying the potential of a defect to impact the design outputs in any possible future execution. Thus, the permanent fault analysis framework differs in use of input stimuli. To gauge if a design impaired after a permanent defect, it is subjected to a full coverage test battery, crafted to excite all internal nodes of the design while observing the outputs. If any of the stimuli reveal an incorrect output, the implication is that there is at least one execution that can expose the defect, and thus such defect is considered exposed.

To derive a simple architect-friendly model of permanent failures, we utilize a time-tested model for silicon failure analysis. In the semiconductor industry, it is widely accepted that the failure rate for many systems follows what is known as the bathtub curve, as illustrated in Fig. 3. We will adopt this failure model for our research. Our goal with the bathtub-curve model is not to predict its exact shape and magnitude for the future (although we will fit published data to it to create “design scenarios”), but rather to utilize the bathtub curve to illuminate the potential design space for future fault-tolerant designs. The bathtub curve represents device failure rates over the entire lifetime of transistors and is characterized by three distinct regions.

- **Infant Period**: In this phase, failures occur very soon and thus the failure rate declines rapidly over time. These infant mortality failures are caused by latent manufacturing defects that surface quickly if a temperature or voltage stress is applied.
- **Grace Period**: When early failures are eliminated, the failure rate falls to a small constant value where failures occur sporadically because of the occasional breakdown of weak transistors or interconnect.
- **Breakdown Period**: During this period, failures occur with increasing frequency over time because of age-related wear-out. Many transistors will enter this period at roughly the same time, creating an avalanche effect and a quick rise in device failure rates.

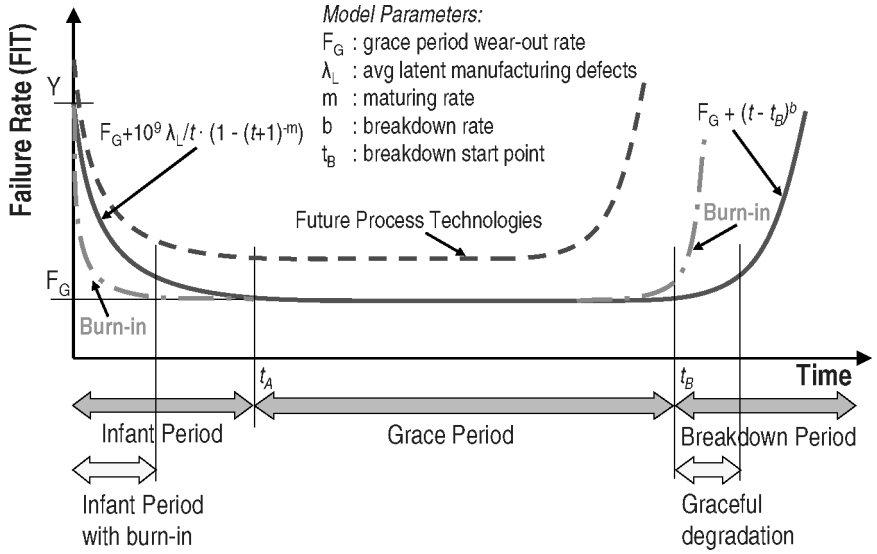


Fig. 3. Simple bathtub curve model of device defect exposure. The curve indicates the qualitative trend of failure rates for a silicon part over time. The initial operational phase and the "agedsilicon" phase are characterized by much higher failure rates.

With the respect to Fig. 3, the model is represented with the following equations:

$$F(t) = \begin{cases} F_G + \lambda_L \frac{10^9}{t}\left(1 - \frac{1}{(t+1)^m}\right), & \text{if } 0 \le t < t_A \\ F_G, & \text{if } t_A \le t < t_B \\ F_G + (t - t_B)^b, & \text{if } t_B \le t \end{cases}$$

(t is measured in hours)
where the parameters of the model are as follows:

- λ_L: average number of latent manufacturing defects per chip
- m: infant period maturing factor
- F_G: grace period failure rate
- t_B: breakdown period start point
- b: breakdown factor

In an effort to base our experiments off of published empirical fault data, we developed a baseline bathtub model based on published literature. Unfortunately, we were unable to locate a single-technology failure model that fully captured the lifetime of a silicon device, so for each period of the bathtub curve, we will use reference values from different sources.

3.2.1 *Latent Manufacturing Defects per Chip (λ_L).* Previous work [Barnett and Singh 2003], showed that the rate of latent manufacturing defects is determined by the formula $\lambda_L = \gamma\lambda_K$, where λ_K is the average number of "killer" defects per chip and γ is an empirically estimated parameter with typical values

between 0.01 and 0.02. The same work provides formulas for deriving the maximum number of latent manufacturing defects that may surface during the burn-in test. Based on these models, the average number of latent manufacturing defects per chip ($140\,mm^2$) for current technologies (λ_L) is approximately 0.005. In the literature, there are no clear trends how this value changes with technology scaling. Thus, we use the same rate for projections of future technologies.

3.2.2 *Grace Period Failure Rate (F_G).* For the grace period failure rate, we use reference data by Srinivasan et al. [2004]. Where a microarchitecture-level model was used to estimate workload-dependent processor hard failure rates at different technologies. The model used supports four main intrinsic failure mechanisms experienced by processors: electromigration, stress migration, time-dependent dielectric breakdown, and thermal cycling. For a predicted post-65-nm fabrication technology, we adopt their worst-case failure rate (F_G) of 55,000 FITs.

3.2.3 *Breakdown Period Start Point (t_B).* Previous work [Stathis 2002], estimates the time to dielectric breakdown using extrapolation from the measurement conditions (under stress) to normal operation conditions. We estimate the breakdown period start point (t_B) to be approximately 12 years for 65-nm CMOS at 1.0 V supply voltage. We were unable to find any predictions as to how this value will trend for fabrication technologies beyond 65-nm, but we conservatively assume that the breakdown period will be held to periods beyond the expected lifetime of the product. Thus, we need not address reliable operation in this period, other than to provide a limited amount of resilience to breakdown for the purpose of allowing the part to remain in operation until it can be replaced.

The maturing factor during the infant mortality period and the breakdown factor during the breakdown period used, are $m = 0.02$ and $b = 2.5$, respectively.

4. BASELINE CMP SWITCH ARCHITECTURE

The goal of the BulletProof project is to design a defect-tolerant chip-multiprocessor capable of tolerating significant levels of various types of defects. In this work, we address the design of one aspect of the system, a defect-tolerant CMP switch. The CMP switch is much less complex than a modern microprocessor, enabling us to understand the entire design and explore a large solution space. Further, this switch design contains many representative components of larger designs including finite state machines, buffers, control logic, and buses.

The baseline design, consists of a CMP switch, similar to the one described in Peh [2001]. This CMP switch provides wormhole routing pipelined at the flit level and implements credit-based flow control functionality for a two-dimensional torus network. In the switch pipeline, head flits will proceed through routing and virtual channel allocation stages, while all flits proceed through switch allocation and switch traversal stages. A high-level block diagram of the router architecture is depicted in Fig. 4a.

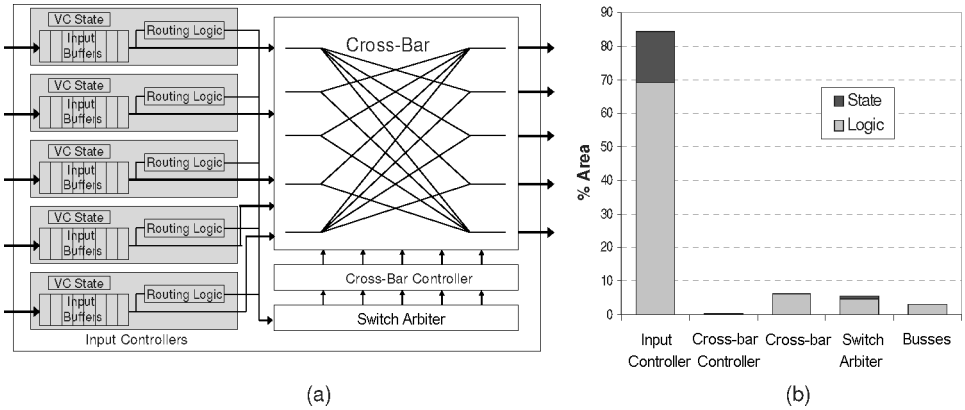


Fig. 4. Baseline CMP switch design. (a) A high-level block diagram for a wormhole interconnection switch is presented. It consists of five input controllers, a cross-bar, a switch arbiter, and a crossbar controller. (b) An area breakdown of the four switch modules and the input/output buses is given. The area is broken down into state and logic.

The implemented router is composed of four functional modules: the input controller, the switch arbiter, the crossbar controller, and the crossbar. The input controller is responsible for selecting the appropriate output virtual channel for each packet, maintaining virtual channel state information, and buffering flits as they arrive and await virtual channel allocation. Each input controller is enhanced with an 8-entry 32-bit buffer. The switch arbiter allocates virtual channels to the input controllers, using a priority matrix to ensure that starvation does not occur. The switch arbiter also implements flow control by tallying credit information used to determine the amount of available buffer space at downstream nodes. The crossbar controller is responsible for determining and setting the appropriate control signals so that allocated flits can pass from the input controllers to the appropriate output virtual channels through the interconnect provided by the crossbar.

The router design is specified in Verilog and was synthesized using the Synopsys Design Compiler and the IBM 0.13-um standard cell library to create a gate-level netlist, which consists of approximately 10 k gates. Figure 4b provides an area breakdown of the four modules and the input/output buses. The height of each bar in this figure represents the percentage of the total area for each module. Further, the bars are broken down into two segments indicating the fraction of area devoted to combinatorial logic and state elements in each module. In this design, the input controller is obviously dominant in area. There are five input controllers; thus, the fraction of area is magnified. The design is also heavily dominated by logic as compared to state: 84 logic versus 16% state.

5. ASSESSING THE SEU VULNERABILITY OF THE BASELINE CMP SWITCH

In this section, we first detail the various ways a transient fault might get masked and then we assess the SEU vulnerability of the baseline CMP switch architecture, using the simulation infrastructure described in Section 3.1.

5.1 Soft-Error Masking

Fortunately, not all transient faults affect the final outcome of a program. In order for a transient glitch caused by a particle strike in combinational logic to affect correct computation, it must first change the value of the latch at the end of the associated logic chain and then propagate to the design's output. There are five basic phenomena that might prevent the glitch from affecting the design's output, thereby masking the transient fault:

- **Logic Masking:** As shown in Fig. 5a, a faulty glitch is logically masked when it fails to affect the input value of a latch, because it gets blocked by a following gate whose output is completely determined by its other input values (i.e., a transient pulse exciting a two-input AND gate when its other input value is a logical zero). It is clear that the fewer levels of logic between two latches, the lower the probability that a faulty glitch will be logically masked. Therefore, it is expected that as the pipelines of microprocessors get deeper and clock frequencies get shorter, the levels of logic in a microprocessor's pipeline stages will become fewer and logic masking within a given pipeline stage will occur less frequently [Shivakumar et al. 2002]. Furthermore, as the pulse duration of the faulty glitch gets larger, the probability that it will be logically masked is lower. This is because the values of the other inputs of the blocking gate that determine its output value must remain unchanged for an extended period of time.
- **Timing Masking:** As shown in Fig. 5b, a faulty glitch is timing masked if it affects the input of a latch only in the period of time that the latch is not sensitive to its input value. It is clear that larger pulse durations lessen the probability of timing masking. Furthermore, the period of time that the latch is sensitive to its input value is determined by the technology's setup and hold times. Therefore, as microprocessors' clock frequencies get shorter and setup and hold times become a larger fraction of the clock period, it is expected that the timing masking will become a less frequent phenomenon.
- **Electrical Masking:** As shown in Fig. 5c, a faulty glitch is electrically masked if its pulse is attenuated by subsequent logic gates because of electrical properties and, as a result, it does not affect the input value of a latch. As with logic masking, electrical masking depends on the number of levels of logic between two latches. Hence, as the length of the faulty glitch gets larger (or the number of levels of logic become fewer), the probability that the transient fault will get masked is less. Furthermore, as transistors get smaller and faster, the effects on pulse attenuation by logic gates are reduced and electrical masking is also expected to reduce.
- **Microarchitectural Masking:** Even when a latch's value is altered by a transient fault (either due to a faulty transient glitch manifested in a combinational logic block that did not get masked and, therefore, successfully changed the latch's value or because of a particle strike directly flipping the latch's value) the transient fault can still be masked and be transparent to the application's correct execution as a result of microarchitectural masking. For example, if a register's bit is flipped by a particle strike and, subsequently,

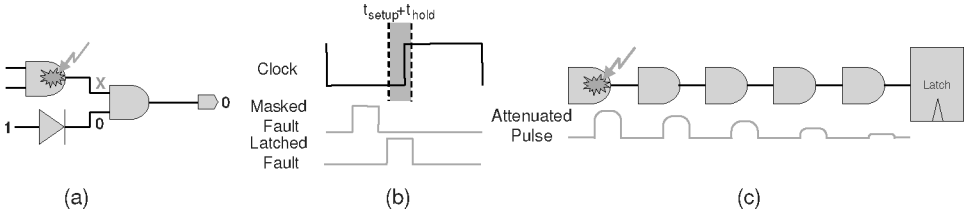


Fig. 5. Transient fault masking phenomena. (a) A particle strikes the first AND gate, but the formed faulty pulse is logically masked by the following AND gate. (b) The faults that do not reach the latch's input during the latching window get timing masked. (c) The fault's pulse gets attenuated by the subsequent logic gates chain and eventually does not affect the following latch.

the register's value gets overwritten by a new value without the wrong value ever having been read, then the fact that, for a period of time, the register's value was incorrect is transparent to the application's correct execution and the transient fault is successfully masked by microarchitectural phenomena. There is also the case where an incorrect value latched in a flip-flop is subsequently masked either by electrical, logic, or time-related masking in the next stage, and is thereby prevented from propagating to the design's output. Since the state of the design is incorrect for at least one cycle, we consider this case microarctitectural-related masking, no matter how the transient fault was subsequently masked.

- **Software Masking:** Even when a transient fault propagates an error to the output of the microprocessor, the error can be masked at the software level [Messer et al. 2004]. For example, when an error is propagated outside of the microprocessor's domain and causes an incorrect value at a memory location, which is then overwritten by the application or the operating system without having been used, then the error is software-masked and it is transparent to the correct execution of the application or the operating system. The quantitative analysis of software masking is out of the scope of this article, as it occurs outside of a microprocessor's domain.

These five masking phenomena significantly decrease the estimated raw soft-error rates for complex circuit designs but, at the same time, because their analysis requires accurate models that track cycle-by-cycle details of circuit activity the aforementioned phenomena, place high demands on the soft-error analysis infrastructure.

5.2 Exposure to Transient Faults

To evaluate the design's exposure to transient faults, we used the simulation infrastructure described in Section 3.1, along with realistic workloads from communication traces derived from the TRIPS architecture [Sankaralingam et al. 2003]. In these experiments, we simulate a single switch in a 5×5 on-chip meshlike operand network connecting ALUs and memory components. The switch chosen for simulation exhibited average communication traffic. The TRIPS network packets carry data (operands for instructions or addresses to memory) and status information associated with them. We used traffic traces

Table I. Simulated Benchmarks[a]

Benchmark	# of Comm. transactions	Clock cycles to completion	Mean Util. (%)	Sim. time (s)
ammp	64664	356452	18.14	758.53
art	44060	549648	8.01	531.64
bzip2	69014	686084	10.06	823.22
compress	60306	1304340	4.62	715.28
equake	67608	641020	8.99	664.00
gzip	76538	572473	13.37	909.39
m88ksim	58336	814768	7.16	689.42
mcf	76866	704404	10.91	920.78
mgrid	71786	258160	27.81	822.63
parser	80338	1429220	5.62	951.34
swim	47105	236272	19.94	540.58
twolf	59529	1009868	5.89	731.58
vortex	70848	439440	16.12	842.99
adpcm	79257	1454536	5.45	969.68
dct	70786	175120	40.42	836.07
hydro2d	62026	302728	20.49	728.78
mpeg2encode	58368	533684	10.94	680.13
tomcatv	51796	208952	24.79	602.62
turb3d	38695	416412	9.29	448.12
hi_util	40005	57150	70.00	505.05

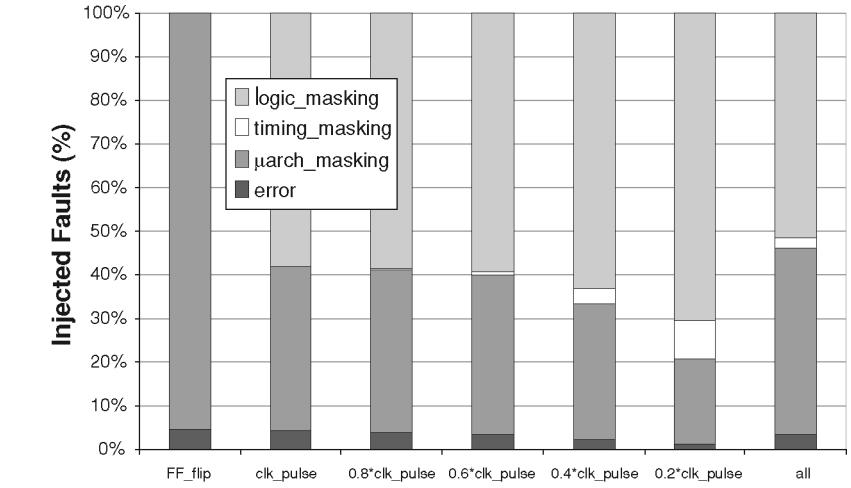
[a]The benchmark pool consists of 13 benchmarks from the SPEC2000 suite, 6 from the MediaBench suite, and 1 synthetic benchmark. For each benchmark, we list the number of communication transactions, the clock cycles needed to complete these transactions, the switch's mean utilization, and the simulation time (s).

for 13 benchmarks from the SPEC2000 benchmark suite, 6 benchmarks from the MediaBench suite, and 1 synthetic high-utilization traffic trace (hi_util), as shown in Table I.

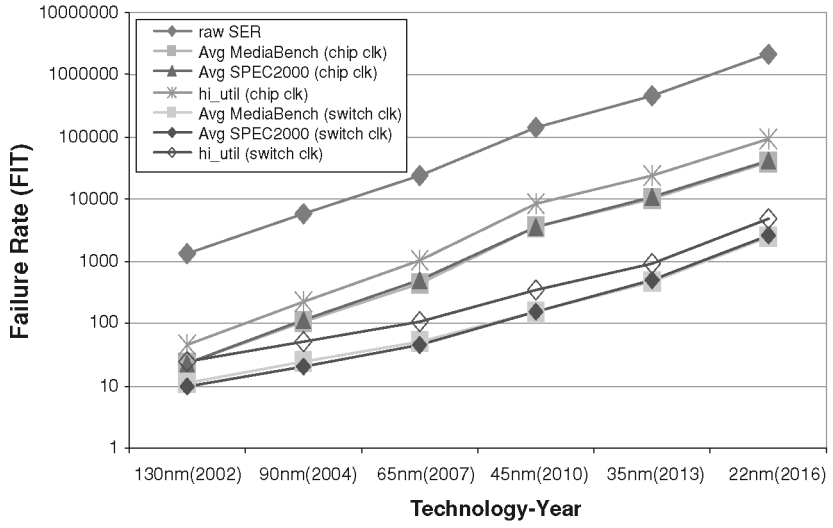
Each traffic trace consists of 32-bit packet communication transactions, where each communication transaction is specified by the incoming input channel, the header of the flit with the destination node (needed for the routing of the packet), the data of the packet, and the clock cycle that the packet is injected into the switch. The mean switch utilization for each traffic trace is specified by the ratio between the number of communication transactions and the number of clock cycles needed to complete all the communication transactions.

The maximum propagation delay of an injected fault to propagate to the design's output in the simulated CMP switch design is 160 clock cycles. Therefore, we let the CMP switch warm up for the first 10,000 cycles and then start injecting faults, in the design, with intervals of at least 200 clock cycles between each injected fault. For each injected fault, we keep track of its effects on the design by monitoring the design's state and outputs and comparing them with those of the golden model.

In Fig. 6a, we classify the injected faults into four categories: (1) the faults that caused an error, (2) faults that were microarchitecturaly masked, (3) faults that were timing masked, and (4) faults that were logically masked. This classification is presented per transient fault type. The first type (column) are



(a)



(b)

Fig. 6. Assessing the SEU vulnerability of the CMP switch. (a) The graph classifies the effect of transient faults on: logically masked time-masked, microarchitecturally-masked faults, and those faults that caused errors. The classification is devised for six different categories of transient faults with different characteristics, and for the combination of all the different fault categories. In part (b), the estimated failure rates resulting from transient faults are projected for six different process technologies, for designs with different clock frequencies, and for various workloads. The raw SER is projected as well.

state-bit flipping transient faults, while the next five are transient glitches with a pulse duration of 100, 80, 60, 40, and 20% of the design's clock period, respectively. The last column is the classification for all types of transient faults combined (weighted based on the transient faults model described in Section 3.1). The presented data are averages over all the SPEC2000 and MediaBench benchmarks.

When all transient fault types are combined, we observe that 51.7% of them are logic masked, 2.2% are timing masked, and 42.9% are microarchitectural masked. The remaining 3.2% of the injected transient faults propagate the fault at the output of the design and consequently cause an error in the application's execution.

The aspects of a design that are important in its tolerance to transient faults are the transistor density (number of transistors per area unit), the raw soft-error rate of a single device in the design, the design's area, and the design's clock frequency. All of these design parameters change with process technology scaling, except for the design area of microprocessor chips which, based on ITRS [2004] projections, will stay constant for future generation microprocessor designs. In order to derive a design's failure rate, we need to know all of the above along with the design's SER derating factor. Based on the design's masking derated SER, derived from simulations using our simulation infrastructure, and projections for the characteristics of future designs from ITRS [2004], we estimated the failure rates of a 5×5 on-chip meshlike interconnection network for current and future process technologies.

Figure 6b presents the estimated failure rates of six different process technologies for varied workloads. Across the different process technologies the architecture of the CMP switch design is kept the same. The vertical axis represents the design's failure rate in FITs, which is the number of failures in a billion hours of operation, and it is plotted in logarithmic scale. The top line is the raw SER, where each energetic particle strike that hits the design is assumed to cause an error in the application's execution. The other lines project the estimated failure rates for two different designs for varied workloads, considering fault masking. Since the design's clock frequency is one of the major design aspects that affect its tolerance to transient faults (and typical interconnection networks are clocked with much slower frequencies than microprocessors), we estimate the failure rates for two designs: one with projected clock frequencies for interconnection networks and a second with projected clock frequencies for microprocessors (though the architecture of the design is the same). As we can see, from the graph, the failure rates for the higher-clock frequency design are an order of magnitude larger than that of the lower-clock frequency design.

For each design, we present the failure rates for different workloads: the synthetic high-utilization benchmark, the average failure rate over the SPEC2000 benchmarks, and the average failure rate over the MediaBench benchmarks. The failure rates for the SPEC2000 and MediaBench benchmarks overlap, which indicates that realistic workloads exhibit similar masking effects to transient faults. The failure rate for the high-utilization benchmark is higher than that of the realistic benchmarks.

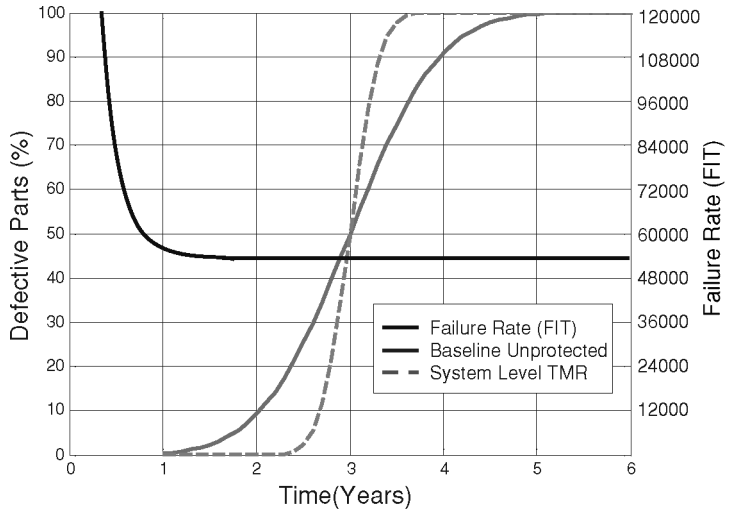


Fig. 7. Baseline design reliability. The graph superimposes the FIT rates of the bathtub model with the fault tolerance of two variants of the CMP switch design: a baseline unprotected version and a variant with a traditional TMR technique.

Clearly, for even switch-dominated designs, with high clock frequencies and unrealisticly high switch utilization, SEU-related FIT rates are extremely low. Considering that we estimate the post-65-nm fault rate resulting from transistor wear-out to be 55,000 FIT, SEU-related faults do not become a relatively major concern for any experiment. This result is because of the fleeting nature of transient fault glitches, which make it extremely difficult for them to influence the state of the system. In fact, at most, 3.2% of all faults result in a corrupted flit transmission. Our switch design is mostly comprised by combinational logic. Thus, most glitches are injected into combination logic and are subsequently masked. As a result of this initial study, we chose to focus the remainder of our design efforts and analysis at reducing the impact of silicon defects.

6. RELIABILITY OF THE BASELINE CMP SWITCH DESIGN

In this section, we evaluate the resiliency of the baseline CMP switch when exposed to hard silicon defects. In Fig. 7, we used the bathtub curve fitted for the post-65-nm technology node as derived in Section 3.2. The FIT rate of this curve is 55000 during the grace period, which corresponds to a mean time to first component failure (MTTF) of 2 years. We used this failure rate in our simulation framework for permanent failures and plotted the results.

The baseline CMP design does not deploy any protection technique against defects and one defect is sufficient to bring down the system. Consequently, the graph of Fig. 7 shows that in a large population of parts, 50% of the parts will be defective by the end of the second year after shipment, by the fourth year almost all parts will have failed. In this experiment, we have also analyzed a design variant that deploys triple-module-redundancy (TMR) at the full-system

level (i.e., three CMP switches with voting gates at their outputs). Designs with TMR applied at different granularities are evaluated in Section 8.

The TMR model used in this analysis is the classic TMR model which assumes that when a module fails, it starts producing incorrect outputs and, if two or more modules fail, the output of the TMR voter will be incorrect. This model is conservative in its reliability analysis because it does not take into account compensating faults. For example, if two faults affect two independent output bits, then the voter circuit should be able to correctly mask both faults. However, the benefit gained from accounting for compensating faults rapidly diminishes with a moderate number of defects, because the probabilities of fault independence are geometrically less likely. Further, though the switch itself demonstrated a moderate number of independent fault sites, submodules within the design tended to exhibit very little independence. Also, in Saxena and McCluskey [1998], it is demonstrated that even when TMR is applied on diversified designs (i.e., three modules with the same functionality, but different implementation), the probability of independence is small. Therefore, in our reliability analysis, we choose to implement the classical TMR model and for the rest of the article whenever TMR is applied, the classical TMR model is assumed.

From Fig. 7, the simulation-based analysis finds that system-level TMR provides very little reliability improvements over the baseline designs, because of the few number of defects that can be tolerated. Furthermore, the area of the TMR protected design is more than three times the size of the baseline design. The increase in area raises the probability of a defect being manifested in the design, which significantly affects the design's reliability. In the rest of the article, we propose and evaluate defect-tolerant techniques that are significantly more robust and less costly than traditional defect-tolerant techniques.

7. SELF-REPAIRING CMP SWITCH DESIGN

A design that is tolerant to permanent defects must provide mechanisms that perform four central activities related to faults: detection, diagnosis, repair, and recovery. Fault detection identifies that a defect has manifested as an error in some signal. Normal operation cannot continue after fault detection, as the hardware is not operating properly. Often, fault detection occurs at a macro-level. Thus it is followed by a diagnosis process to identify the specific location of the defect. Following diagnosis, the faulty portion of the design must be repaired to enable proper system functionality. Repair can be handled in many ways, including disabling, ignoring, or replacing the faulty component. Finally, the system must recover from the fault, purging any incorrect data and recomputing corrupted values. Recovery essentially makes the defect's manifestation transparent to the application's execution. In this section, we discuss a range of techniques that can be applied to the baseline switch to make it tolerant of permanent defects. The techniques differ in their approach and the level at which they are applied to the design.

In Dally et al. [1994] the authors present the reliable router (RR), a switching element design for improved performance and reliability within a mesh

interconnect. The design relies on an adaptive-routing algorithm, coupled with a link-level retransmission protocol in order to maintain service in the presence of a single node or link failure within the network. Our design differs from the RR in that our target domain involves a much higher fault rate and focuses on maintaining switch service in the face of faults rather than simply routing around faulty nodes or links. However, the two techniques can be combined to provide a more reliable multiprocessor interconnection network.

7.1 General Techniques

The most commonly used protection mechanisms are dual-and triple-modular redundancy, or DMR and TMR [Siewiorek and Swarz 1998]. These techniques employ spatial redundancy combined with a majority voter. With permanent faults, DMR provides only fault detection. Hence, a single fault in either of the redundant components will bring the system down. TMR is more effective, as it provides solutions to detection and recovery. In TMR, the majority voter identifies a malfunctioning hardware component and masks its effects on the primary outputs. Hence, recovery is trivial, since the defective component is always outvoted when it computes an incorrect value. Because of this restriction, TMR is inherently limited to tolerating a single permanent fault. Faults that manifest in either of the other two copies cannot be handled. DMR/TMR are applicable to both state and logic elements and, thus, are broadly applicable to our baseline switch design.

Storage or state elements are often protected by parity or error-correction codes (ECC) [Siewiorek and Swarz 1998]. ECC provides a lower overhead solution for state elements than TMR. Like TMR, ECC provides a unified solution to detection and recovery. Repair is again trivial as the parity computation masks the effects of permanent faults. In addition to the storage overhead of the actual parity bits, the computation of parity or ECC bits generally requires a tree of exclusive-ORs. This hardware has moderate overhead, but, more importantly, it can often be done in parallel, thus, not affecting latency. For our defect-tolerant switch, the application of ECC is limited because of the small fraction of area that holds state.

7.2 Level of Protection

The error resiliency achieved by employing redundancy is highly dependent on the granularity of the redundant partitions. In general, the larger the granularity of the redundant partitions, the less robust the design. However, as the granularity of the redundant partition becomes smaller, more logic is required to implement protection mechanisms. For example, with TMR, each output for a given partition requires a `MAJORITY` gate.

To illustrate these trade-offs, consider once again the baseline switch in Fig. 4a. TMR can be applied on the system-level where the whole switch is replicated and each output requires a single `MAJORITY` gate. A single permanent error makes one copy of the switch completely broken. However, the area overhead beyond the redundant partitions is limited to only a gate for each primary output. A slightly more resilient design considers partitioning, based

on the components that make up the switch. For instance, each of the five input controllers can have a redundant partition along with the arbiter, crossbar, and the crossbar controllers. This partitioning approach leaves the design more protected as a permanent defect in the input controller would make only that small partition broken and not the other four input controllers. The area penalty of this approach is slightly higher as the sum of the outputs, for each partition is greater than the switch, as a whole. However, the added unprotected logic is still insufficient to worsen the error resiliency of the design. Finally, if the design is partitioned at the gate level, each gate is in its own partition. In this scheme, the error resiliency for each partition is extremely high, because the target is very small. However, the overhead of this approach requires an extra gate for each gate in the switch design. Thus, the area would be four times the original design. In addition, because each added gate is unprotected, the susceptibility of this design to errors may be greater than designs protected at higher levels of granularity.

The previous analysis shows that the level of partitioning effects the error resiliency and the area overheads of the design. In this article, we introduce a technique called, *automatic cluster decomposition*, that generates partitions that minimizes area overhead and the amount of unprotected logic while maximizing error resiliency.

7.3 Automatic Cluster Decomposition

Automatic cluster decomposition (ACD) takes a netlist and creates partitions, with the end goal that each partition is approximately the same size and that there is a minimal amount of outputs required for each partition generated. Generating these partitions requires that the netlist be converted into a *hypergraph*[1] that can then be partitioned using a balanced-recursive min-cut algorithm.

The min-cut algorithm is based on multilevel-hypergraph partitioning. Specifically, we utilize the algorithm implemented by the *hMETIS* graph partitioning tool [Karypis et al. 1997]. As illustrated in Fig. 8, the algorithm is comprised of three phases: (1) the coarsening phase, (2) the initial partitioning phase, and (3) the uncoarsening and refinement phase. The goal of the coarsening phase is to construct a sequence of successively smaller hypergraphs so that a good bisection of the small hypergraph is not significantly worse than the bisection directly obtained for the original hypergraph. During the initial partitioning phase, a bisection of the coarsened hypergraph is computed. Finally, in the uncoarsening and refinement phase, the partitioning of the coarsest hypergraph is used to obtain a partitioning for the finer hypergraph. Using a partitioning refinement algorithm, the cuts between partitions are reduced and thus the quality of the partitioning is improved. For more details about the partitioning algorithm, we refer the reader to Karypis and Kumar [1998].

[1]A hypergraph is a generalization of a graph, where the set of edges is replaced by a set of hyperedges. A hyperedge extends the notion of an edge by allowing more than two vertices to be connected by a hyperedge.

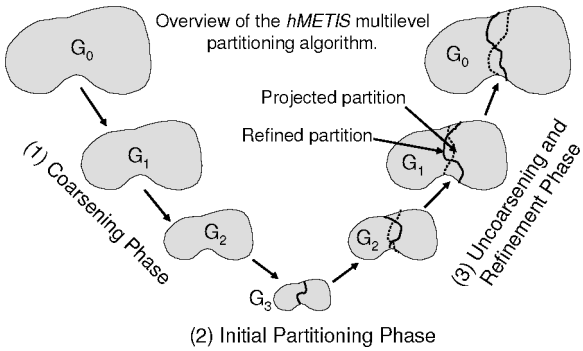


Fig. 8. Overview of the multilevel partitioning algorithm. The multilevel partitioning algorithm implemented by the *hMETIS* tool consists by three phases: (1) the coarsening phase, (2) the initial partitioning phase, and (3) the uncoarsening and refinement phase.

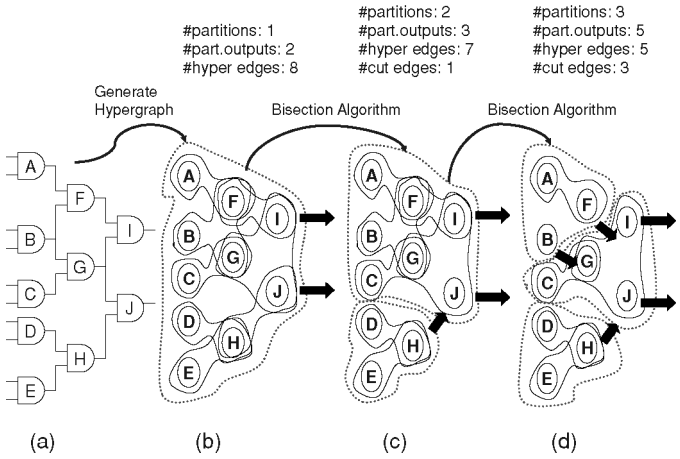


Fig. 9. The process of automatic cluster decomposition. In part (a) a sample netlist is shown with 2 primary outputs, along with its corresponding hypergraph in part (b). Part (c) shows the hypergraph after a min-cut bisection creating two unbalanced partitions. Part (d) shows the final 3-way partition resulting from a bisection of the largest partition.

Figure 9 shows an example of how these partitions are generated from the netlist of a design. First, the netlist pictured in (a) is used to generate the *hypergraph* shown in (b). For this example, we show a three-way partitioning of the circuit. In Fig. 9c, the hypergraph is bisected and the number of hyperedges cut is reported. Figure 9d shows the final partitioning assignment of the hypergraph, along with the number of hyperedges cut, which corresponds to the number of total outputs for all the partitions not including the original outputs of the system.

Table II shows the area, delay, and wire-length overheads for switch design configurations with different number of partitions. The first design configuration considers the whole design as one partition and the partition cut edges are, therefore, the outputs of the design. The second design configuration considers

Table II. Area, Delay and Wire-Length Overheads for Switch Design Configurations with Different Number of Partitions[a]

Description	Num. of partitions	Cut edges	Cut edges area overhead (%)	Critical path cut edges	Critical path delay (%)	Wire-length overhead (%)
System	1	165	1.57	0	0.00	4.85
Components	12	405	4.38	0	0.00	6.58
Gates	10540	10540	100	18	100	NA
ACD-2	2	293	2.79	0	0.00	5.42
ACD-4	4	395	3.78	1	5.56	9.17
ACD-8	8	445	4.29	1	5.56	4.37
ACD-16	16	457	4.48	2	11.11	2.22
ACD-32	32	497	5.01	3	16.67	1.49
ACD-64	64	558	5.9	3	16.67	6.47
ACD-128	128	746	8.29	4	22.22	17.09
ACD-256	256	1060	12.47	5	27.78	28.04

[a]The percentage overheads shown for the different switch-design configurations are over the baseline unpartitioned and unprotected design.

as partitions the twelve functional components of the switch (the sizes of the components are unequal). The third design configuration listed represents the limit of the design space where each individual gate in the design is considered as a different partition. The rest of the listed switch design configurations are designs with different number of partitions created by employing the ACD technique.

For generating the data reported in Table II, the initial router design was partitioned using the *hMETIS* tool. After optimal partitions were identified, multiplexers were added for each cut net because of the partitioning. The resulting design was then automatically placed and routed (APR) using the tool CAPO [Riess and Ettelt 1995]. The data reported in Table II suggests that the number of total cut edges, the area overhead because of cut edges, the cut edges in the critical path, and the critical path delay, are all increasing as the number of partitions in the design are increasing. On the other hand, the wire-length overheads show a slight decrease followed by an increase. These wire-length overheads result from the additional controlling wires to the partition selection multiplexers and to the wires needed for connecting the partition inputs/outputs to the added multiplexers. By having fewer and bigger partitions, the added multiplexers coresponding to each partition, span a larger area. Therefore, the partition controlling wires span longer distances, thus increasing the design's wire-length. However, by having more and smaller partitions, eventually the number of added multiplexers offset the benefit of having each partition's multiplexers close to each other.

In order to study the error resiliency provided to a switch-design configuration, we introduce a new metric, the *silicon protection factor (SPF)*, which gives us a more representative notion about the amount of protection that is offered to the system by a given defect-tolerance technique. Specifically, the SPF is computed by dividing the mean number of defects needed to cause a switch failure in the area overhead of the protection technique. The key advantage of the SPF metric is that it takes into account the size of the protected design and

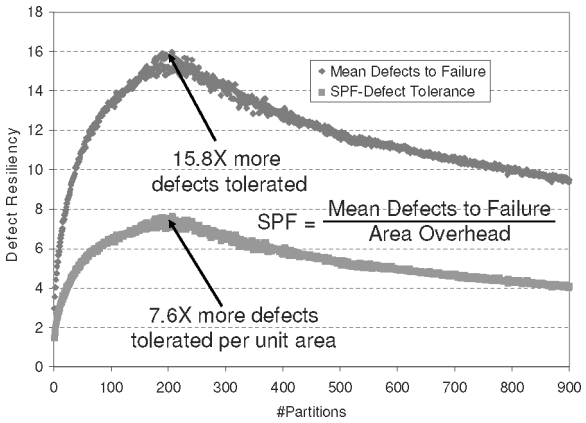


Fig. 10. Defect resiliency as a function of the number of partitions. As an example, we plot the mean defects to failure and SPF defect tolerance of a switch design configuration with one extra redundant partition for each partition in the design. The results are shown for a varying number of partitions generated by the ACD algorithm.

the degree to which unprotected logic is exposed to faults. In other words, the higher the SPF factor, the more resilient each transistor is to defects.

Figure 10 shows the dependency of the mean defects to failure and the SPF value over the number of decomposed partitions for a switch design configuration with one extra redundant partition for each partition in the design. We can see that for the given design configuration the peak SPF occurs around 200 partitions. As the per partition size decreases, the SPF value increases; as the number of cut edges per partition increases, the SPF value decreases. Therefore, the initial rise of the SPF occurs because the area per partition was decreasing as the number of decomposed partitions was getting larger. After the optimal point of 200 partitions, the overhead of the extra unprotected logic required for each cutting edge between partitions causes the SPF to start declining.

7.4 Resource Sparing

Instead of employing a TMR technique for providing defect tolerance to the switch design we use resource sparing for selected partitions of the switch. During the switch operation only one spare is active for each distinct partition of the switch. For each spare added in the design, there is an additional overhead for the interconnection and the required logic for enabling and disabling the spare. For resource sparing, we study two different techniques: dedicated sparing and shared sparing. In the dedicated-sparing technique, each spare is owned by a single partition and can be used only when the specific partition fails. When shared sparing is applied, one spare can be used to replace a set of partitions. In order for the shared sparing technique to be applied, it requires multiple identical partitions, such as the input controllers for the switch design. Furthermore, each shared spare requires additional interconnect and logic overhead because of the need of having the ability to replace more than one possible defective partitions.

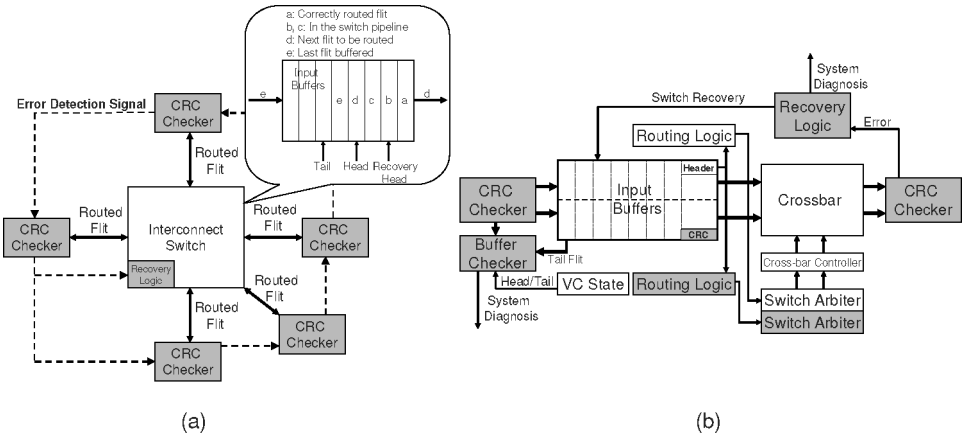


Fig. 11. End-to-End error detection and recovery mechanism. (a) The interconnection switch is enhanced by CRC and recovery logic for providing data-corrupting error detection. The input buffers are enhanced with an extra recovery head pointer to mark the last correctly checked flit. (b) A more detailed view of the switch with end-to-end error detection is shown. Flits are split into two parts, which are independently routed through the switch pipeline.

7.5 Domain-Specific Techniques

The properties of the wormhole router can be exploited to create domain-specific protection mechanisms. Here, we focus on one efficient design that employs end-to-end error detection, system diagnosis, and reconfiguration.

7.5.1 *End-to-End Error Detection and Recovery Mechanism.* Within our router design, errors can be separated into two major classes. The first class is comprised of data-corrupting errors, for example, a defect that alters the data of a routed flit, so that the routed flit is permanently corrupted. The second class is comprised of errors that cause incorrect operation, for example, a defect that causes a flit to be misrouted to a wrong output channel or to get lost and never reach any of the switch's output channels.

The first class of errors, the data-corrupting errors, can be addressed by adding cyclic redundancy checkers (CRC) at each one of the switch's five output channels, as shown in Fig. 11a. When an error is detected by a CRC checker, all CRC checkers are notified about the error detection and any further flit routing is blocked. The same error-detection signal used to notify the CRC checkers also notifies the switch's recovery logic. The switch's recovery logic logs the error occurrence by incrementing an error counter. In case the error counter surpasses a predefined threshold, the recovery logic signals the need for system diagnosis and reconfiguration.

In case the error counter is still below the predefined threshold, the switch recovers its operation from the last "checkpointed" state, by squashing all inflight flits and rerouting the corrupted flit and all following flits. This is accomplished by maintaining an extra recovery head pointer at the input buffers. As shown in Fig. 11a, each input buffer maintains an extra head pointer which indicates the last flit stored in the buffer which is not yet checked by a CRC checker. The

recovery head pointer is automatically incremented four cycles after the associated input controller grants access to the requested output channel, which is the latency needed to route the flit through the switch, once access to the destination channel is granted. In case of a switch recovery, the recovery head pointer is assigned to the head pointer for all five input buffers, and the switch recovers operation by starting rerouting the flits pointed by the head pointers. Further, the switch's credit backflow mechanism needs to be adjusted accordingly, since an input buffer is now considered full when the tail pointer reaches the recovery head pointer. In order for the switch's recovery logic to be able to distinguish soft from hard errors, the error counter is reset to zero at regular intervals.

The detection of functional errors is considerably more complicated, because of the need to be able to detect misrouted and lost flits. A critical issue for the recovery of the system is to assure that there is at least one uncorrupted copy for each flit in flight in the switch's pipeline. This uncorrupted flit can then be used during recovery. To accomplish this, we add a buffer checker unit to each input buffer. As shown in Fig. 11b, the buffer checker unit compares the CRC checked incoming flit with the last flit allocated into the input buffers (tail flit). Further, to guarantee the input buffer's correct functionality, the buffer checker also maintains a copy of the head and the tail pointers, which are compared with the input buffer's pointers whenever a new flit is allocated. In the case that the comparison fails, the buffer checker signals an allocation retry, to cover the case of a soft error. If the error persists, this means that there is a potential permanent error in the design, which signals the system diagnosis and reconfiguration procedures. By assuring that a correct copy of the flit is allocated into the input buffers and that the input buffer's head/tail pointers are maintained correctly, we guarantee that each flit entering the switch will correctly reach the head of the queue and be routed through the switch's pipeline.

To guarantee that a flit will get routed to the correct output channel, the flit is split into two parts, as shown in Fig. 11b. Each part will get its output channel requests from a different routing logic block, and access the requested output channel through a different switch arbiter. Finally, each part is independently routed through the crossbar. To accomplish this, we add an extra routing logic unit and an extra switch arbiter. The status bits in the input controllers that store the output channel reserved by the head flit are duplicated as well. Since the crossbar routes the flits at the bit-level, the only difference is that the responses to the crossbar controller from the switch arbiter will not be the same for all the flit bits, but the responses for the first and the second parts of the flit are fitted from the first and second switch arbiters, respectively. If a defect causes a flit to be misrouted, it follows that a single defect can impact only one of the two parts of the flit, and the error will be caught later at the CRC check.

The area overhead of the proposed error detection and recovery mechanism is limited to only 10% of the switch's area. The area overhead of the CRC checkers, the recovery logic and the buffer checker units is almost negligible. More specifically, the area of a single CRC checker is 0.1% of the switch's area and the area for the buffer checker and the recovery logic is much less significant. The area

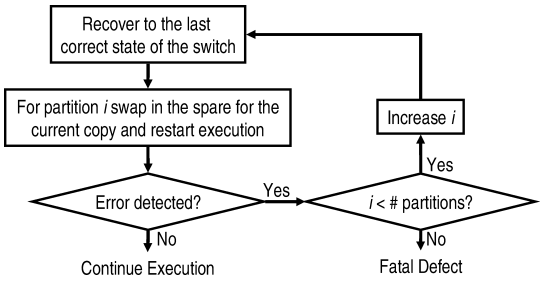


Fig. 12. Iterative system diagnosis and reconfiguration. The algorithm recovers to the last correct state, reconfigures the system, and replays the execution until no error is detected.

overhead of the proposed mechanism is dominated by the extra switch arbiter (5.7%), the extra routing logic units ($5 \times 0.5\% = 2.5\%$), and the additional CRC bits (1.5%). As we can see, the proposed error detection and recovery mechanism has a 10X times less area overhead than a naïve DMR implementation.

7.5.2 *System Diagnosis and Reconfiguration.* As a system diagnosis mechanism, we propose an iterative trial-and-error method, which recovers to the last correct state of the switch, reconfigures the system, and replays the execution until no error is detected. The general concept is to iterate through each spared partition of the switch and swap in the spare for the current copy. For each swap, the error detection and recovery mechanism performs a system replay. Eventually, the partition that happens to possess the current error will be disabled and its corresponding spare enabled. When this occurs, the system diagnosis mechanism will detect correct system behavior and terminate the replay mode. Using this approach, the faulty piece of logic is identified and correctly disabled. A flow chart of the system diagnosis and reconfiguration algorithm is shown in Fig. 12.

We also consider the use of built-in-self-test(BIST) as an alternative for providing system diagnosis. For each distinct partition in the design, we store in ROM automatically generated test vectors. During system diagnosis with BIST, the system operation is interrupted and these test vectors are applied to each partition of the system through scan chains to check its functionality correctness and locate the defective partition. Once the defective partition is located, in the case that there is a nondefective spare of the partition in the design, the defective partition is disabled by swapping it with the nondefective spare partition. After reconfiguration, the switch recovers to the last correct state and restarts execution. In the case that the design is not augmented with a nondefective spare, the defect is a fatal defect and the system will not be able to repair the defective switch.

Although the BIST approach is much faster than the iterative replay diagnosis technique, it comes with the extra overhead of additional scan logic needed for on-line testing, and storing on-chip the test vectors of each partition in the design. In order to generate overhead estimates for the BIST system diagnosis technique for several switch design configurations, we used the Synopsys TetraMAX ATPG tool. TetraMAX, provides a completely automated design flow

Table III. Built-In-Self-Test System Diagnosis Technique Overheads[a]

Design description	Num. of partitions	Scan-logic overhead (%)	Test vector bits	Test vectors overhead (%)	Total area overhead (%)
System	1	14.2	34,680	2.7	16.9
Components	12	12.2	22,568	4.3	16.5
ACD-2	2	15.6	7,216	0.6	16.2
ACD-4	4	14.6	28,864	1.9	16.5
ACD-8	8	14.7	57,640	3.8	18.5
ACD-16	16	14.7	115,115	7.1	21.8
ACD-32	32	15.0	229,984	14.5	29.5
ACD-64	64	15.6	460,128	29.8	45.4
ACD-128	128	17.6	920,360	59.6	77.2
ACD-256	256	18.7	1,839,520	217.3	236.0

[a]The overheads of the built in self-test system diagnosis technique consist by the additional scan logic needed for on-line testing and on-chip storage for test vectors. This table shows the area overhead estimates for various design configurations.

for adding the scan logic needed for enabling design on-line testing at the synthesized gate-level design model. It also provides automatic generated quality test vectors for high-fault coverage design testing. For estimating the area overhead of on-chip storing of the test vectors in ROM, we used the Synopsys ROM generator. These area overhead estimates are shown in Table III for various switch-design configurations that we consider in this work.

The test vectors generated for the presented configurations provide at least 90% fault coverage (in some cases, having 100% fault coverage is prohibitively expensive). This means that if the defect is not covered by the generated test vectors, then after running a full system diagnosis, the system will not be able to locate which partition is defective and thus it will not be able to recover. Unlike the BIST technique, the iterative replay system diagnosis technique can guarantee that if there is a nondefective spare of the defective partition in the design, then the system will be able to locate the defective partition and recover.

The test vector bits shown in Table III account for the total test vector bits for all the partitions in the design. Even though the overhead of the BIST system diagnosis technique is relatively low for design configurations with moderate number of partitions, the overhead is significantly increased for design configuration with more than 64 partitions, reaching up to 236% for a design configuration with 256 partitions. As shown in Section 7.3, these are the design configurations that lead to higher defect resiliency.

Both the iterative replay and BIST techniques can be implemented as a separate module from the switch and the area overhead for their implementation can be shared by a wide number of switches in a chip multiprocessor design, thus mitigating the overhead of system diagnosis. Further results comparing the two system diagnosis techniques will be presented in Section 8.

8. EXPERIMENTAL RESULTS

To evaluate the effectiveness of our domain-specific defect tolerance techniques in protecting the switch-design, we simulated 38 different switch-design

Table IV. Mnemonic Table for Design Configurations[a]

Mnemonic group	Mnemonic	Description
Level of applying defect-tolerance technique	S	System level
	C	Component level
	G	Gate level
	S+CL	System level clusters (clustering is applied at the whole design)
	C+CL	Component level clusters (clusters are limited only within components)
Defect-tolerance techniques (can be applied in combinations)	TMR	Triple Modular Redundancy
	#SP	# dedicated spares for each partition
	#SH(X)	# shared spares for partition of type X
	ECC	Error-correction codes applied at state
System diagnosis technique	IR	Iterative replay
	BIST	Built-in-self-test
Example configurations	S+CL_1SP_IR	System level clusters with one spare for each partition and iterative replay.
	C_2SH(IC)+1SP_BIST	Component level with two shared input controllers and one dedicate spare for the rest of the components. BIST for system diagnosis.
	C+CL_TMR+ECC	Component level clusters TMR with ECC protected state.

[a]For each portion of the naming convention, we show the possible mnemonics with the related description. The last portion provides some example design configurations.

configurations with both traditional and domain specific fault-tolerant design techniques. Each configuration providing a defect-tolerant switch design is characterized by three parameters: level of protection, techniques applied, and system diagnosis method. For each configuration, we use the following naming convention:

$$< level > \_ < technique > \_ < diagnosis >$$

The configurations using TMR as the defect tolerance technique do not use the end-to-end error detection, recovery, and system diagnosis techniques, since TMR fully incorporates error detection, diagnosis, and recovery. All other configurations use the end-to-end error detection and recovery technique, along with either iterative replay or BIST for system diagnosis. Table IV describes the choices that we considered in our simulated configurations for the three parameters, and it gives some example configurations along with their naming conventions.

In Table V, we list the design configurations that we simulated. For each simulated design configuration, we provide the area overhead needed for implementing the specific design. This area overhead includes the extra area needed for the spare units, the majority gates, the logic for enabling and disabling spare units, the logic for the end-to-end error detection, and overheads for recovery and system diagnosis (different configurations have different requirements for the extra logic added). We note that the design configurations with the higher area overheads are the ones applying BIST for system diagnosis. This is due to the extra area needed for storing the test vectors necessary for self-testing each distinct partition in the design, along with the additional interconnection

Table V. Results of the Evaluated Designs[a]

Key	Design configuration	Area o.head	Defects	SPF	*Part.	%Dly
1	S_TMR	3.02	2.49	0.82	1	0.00
2	S+CL_TMR	3.08	16.78	5.45	241	22.22
3	S+CL_TMR+ECC	3.07	6.92	2.25	185	27.78
4	C_TMR	3.04	4.68	1.54	12	0.00
5	C+CL_TMR	3.09	15.86	5.13	223	18.75
6	C+CL_TMR+ECC	3.11	6.25	2.01	298	25.00
7	O_TMR	4.00	4.00	1.00	105.40	100.00
8	S_1SP_IR	2.22	3.27	1.47	1	0.00
9	S+CL_1SP_IR	2.30	17.53	7.63	206	22.22
10	S+CL_1SP_BIST	3.16	17.53	5.54	206	22.22
11	S+CL_1SP+ECC IR	2.48	5.96	2.41	183	27.78
12	S+CL_1SP+ECC_BIST	3.34	5.96	1.78	183	27.78
13	C_1SP_IR	2.24	5.87	2.62	12	0.00
14	C_1SP_BIST	2.79	5.87	2.62	12	0.00
15	C+CL_1SP_IR	2.33	16.04	6.88	223	18.75
16	C+CL_1SP_ECC_IR	2.51	5.34	2.13	138	25.00
17	S_2SP_IR	3.32	5.95	1.79	1	0.00
18	S+CL_2SP_IR	3.42	37.99	11.11	206	22.22
19	S+CL_2SP_BIST	4.29	37.99	8.86	206	22.22
20	S+CL_2SP+ECC IR	3.39	8.64	2.55	118	22.22
21	C_2SP_IR	3.36	13.07	3.90	12	0.00
22	C_2SP_BIST	3.90	13.07	3.35	12	0.00
23	C+CL_2SP_IR	3.44	32.33	9.39	208	18.75
24	C_CL_2SP_BIST	4.31	32.33	7.50	208	18.75
25	C+CL_2SP+ECC_IR	3.41	7.49	2.20	103	25.00
26	C_2SH(IC)_IR	1.52	3.15	2.07	12	0.00
27	C_3SH(IC)_IR	1.71	4.14	2.43	12	0.00
28	C_4SH(IC)_IR	1.89	5.02	2.65	12	0.00
29	C_5SH(IC)_IR	2.08	5.90	2.84	12	0.00
30	C 2SH(IC)+1SP_IR	1.74	4.40	2.53	12	0.00
31	C_3SH(IC)+1SP_IR	1.93	5.79	3.01	12	0.00
32	C_4SH(IC)+1SP_IR	2.12	7.10	3.34	12	0.00
33	C_5SH(IC)+1SP_IR	2.41	8.39	3.48	12	0.00
34	C_2SH(IC)+2SP_IR	1.93	5.01	2.60	12	0.00
35	C_3SH(IC)+2SP_IR	2.12	6.57	3.09	12	0.00
36	C_4SH(IC)+2SP_IR	2.30	8.10	3.52	12	0.00
37	C_5SH(IC)+2SP_IR	2.50	9.58	3.84	12	0.00
38	S_ECC	1.18	1.16	0.98	12	0. 00

[a]For each design configuration, we report the mnemonic, the area factor over the baseline design, the number of defects that can be tolerated, the SPF, the number of partitions, and an estimate of the impact on the system delay.

and logic needed for the scan chains. Another design configuration with high area overhead is the one where TMR is applied at the gate level because of the extra voting gate needed for each gate in the baseline switch design. On the other hand, designs with shared spares achieve low area overhead (under two), since not every part of the switch is duplicated. The area overhead for the rest of the design configurations depends on the amount of spares per partition.

In the fourth column of the table, we provide the mean number of defects to failure for each design configuration. The design configurations providing high mean number of defects to failure are the ones employing the ACD (automatic cluster decomposition) technique. Another point of interest is that techniques employing ECC, even when coupled with ACD perform poorly. Although state is traditionally protected by ECC, when a design is primarily combinational logic, like our switch, the cost of separating the state from the logic exceeds the protection given to the state elements. In other words, if the state is not considered in the ACD analysis and is, therefore, not part of any of the spared partitions, the boundary between the state and the spared partitions must have some unprotected interconnection logic. This added logic, coupled with the unprotected logic required by ECC, makes ECC undesirable in a logic-dominated design.

The SPF values (see Section 7.3) for each design are presented in the fifth column of Table V. The highest SPFs are given by the design configurations that employ automatic cluster decomposition, with the highest being design *S+CL_2SP_IR* at 11.11. Even though design *S+CL_2SP_BIST* uses the same sparing strategies, the area overhead added from BIST decreases the design's SPF significantly. It is interesting that two design configurations have SPFs lower than one. The first one is TMR applied at the system level, which can tolerate an average of 2.5 defects, but the area overhead is more than triple, thus making the new design less defect tolerant than the baseline switch design by 18%. The second one is where the state is protected by ECC. Since our design is logic-dominated and the protected fraction of the design is very small, the extra logic required for applying ECC (which is unprotected) is larger than the actual protected area. Thus, this technique makes the specific design less defect-tolerant than the baseline unprotected design by 2%.

The sixth column in the table shows the number of distinct partitions for each design configuration. As shown in Section 7.3, this parameter is very important for the configurations employing ACD, since the SPF value provided by a given design configurations is greatly dependent on the number of partitions in the decomposed design.

The final column in Table V, %Delay, gives the percentage increase of the critical path delay in the switch. Our results show that for the best designs, we always achieve a delay increase of less than 25%. The designs that involve ACD involve the greatest increase in delay because the partitions generated frequently split up the critical paths. Designs with minimal amount of clustering, such as the *C_2SH(IC)_IR*, achieve no overhead as no interconnection logic is added to any of the critical paths. In general, our results indicate that achieving high SPF factors require slight delay penalty; however, in principle, the ACD strategy could be used to try to minimize the number of critical paths that are partitioned.

The graph in Fig. 13 shows the trade-off between defect tolerance and area overhead. The horizontal axis of the graph represents the defect tolerance provided from a design configuration in SPFs and the vertical axis the area overhead of the design configuration. The further to the right a design configuration

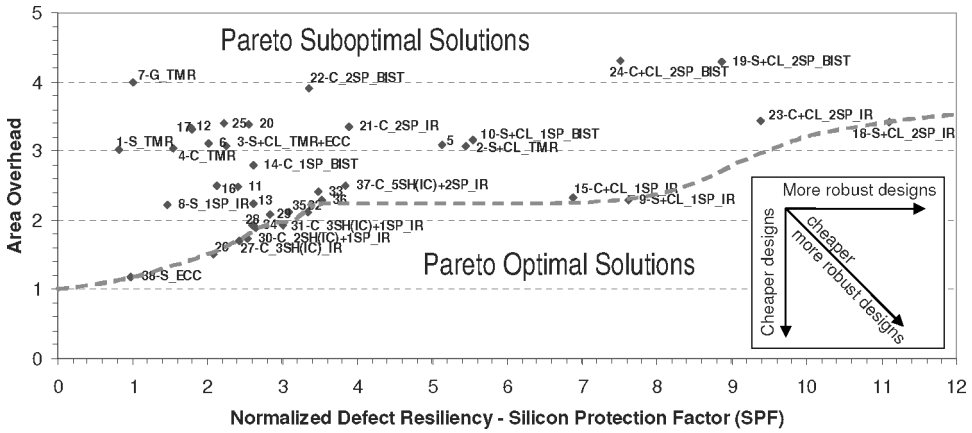


Fig. 13. Pareto chart of the explored solutions. The design evaluated are plotted on an area versus SPF chart. The dashed line across the chart connects the set of optimal solutions. (See Table IV for explanations of design points.)

lies, the higher the defect tolerance it provides, while the lower it is, the lower the implementation cost.

At the lower left corner, is the design configuration *S_ECC* providing ECC protection to the state. This is the cheapest design configuration, but it does not provide any considerable defect tolerance to the switch design. The rightmost design configuration, *S+CL_2SP_IR*, provides a defect tolerance of 11.11 SPF, by employing automatic cluster decomposition at the system level with 206 partitions and two extra spares for each partition, along with iterative replay for system diagnosis. The area cost for implementing this design configuration is 3.42X and provides the better trade-off between area required and offered defect protection. Design configurations with moderate SPFs, but with much less cost in area overhead, are: *C_3SH(IC)_IR*, *C_2SH(IC)+1SP_IR*,*C_3SH(IC)+1SP_IR*, and*C_2SH(IC)+2SP_IR*. These design configurations use shared spares of input controllers along with dedicated spares for the other components in the switch design, keeping the area cost less than 2X, but offering SPFs of 2.5 to 3 at the same time. Such designs are interesting, since they keep the implementation cost at low levels and provide an attractive solution for defect tolerance.

Other two interesting design configurations are *C+CL_1SP_IR* and *S+CL_1SP_IR*. These two designs use the same technique, automatic cluster decomposition with one spare for each partition, with the difference that design *S+CL_1SP_IR* applies the ACD technique on the system level, and design *C+CL_1SP_IR* at the component level. The area cost of the two designs is almost the same, but *S+CL_1SP_IR* provides 11% more SPF. The same argument also holds for designs *S+CL_2SP_IR* and *C+CL_2SP_IR*. This suggests that applying the ACD technique at the system level can offer more effective defect tolerance at the same cost in area.

In Fig. 14, we present how some of the design configurations affect the lifetime of the switch design for a future post 65-nm technology where the mean

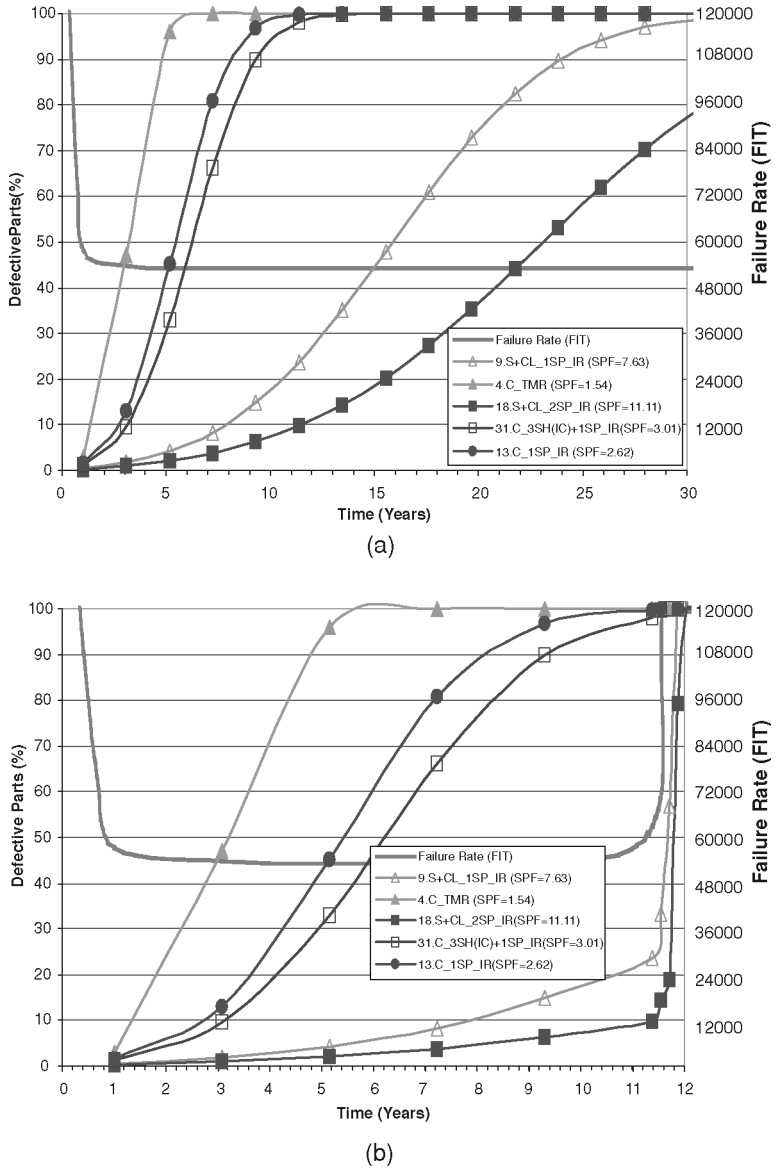


Fig. 14. Fault-tolerance of some interesting design configurations. (a) Superimposion the FIT rate of the bathtub model with the percentage of defective parts over time. (b) The breakdown period is accounted for.

time between hard transistor failures is 2 years (i.e., a failure rate of 55,000 FITs). The graph's horizontal axis represents the years that the switch design is operating. The vertical axis represents the percentage of defected parts over a population of switches (left axis) and the baseline switch's failure rate (right axis). The baseline switch's lifetime failure rate for the given technology is presented by the darker thick line, forming the bathtub curve. In Fig. 14a, it only

forms a part of the bathtub curve since, for this graph, we assume that the design's breakdown occurs after 30 years. For each design configured presented in the graph, there is a line showing the failing rate of switch parts over time. This line starts from year one, since we assume that the first year of a parts lifetime is consumed during the accelerated testing (burn-in) procedure, and that shipped parts are already at their first year of lifetime with a constant failure rate.

From the graph in Fig. 14a, we can observe that when applying TMR at the component level, 25% of the shipped parts will become defective by the first year and 75% after the first 3 years. On the other hand, when TMR is applied in a design configuration where automated cluster decomposition was performed at the system level with two spares for each partition, the 25% of the shipped parts will be defected after 16 years and the 75% after 29 years. If we define the lifetime of a manufactured product as the period of time where 10% of the manufactured parts become defective, then the clustering design configuration *S+CL_2SP_IR* increases the switch's lifetime by 26X over the TMR design configuration *C_TMR*.

System designers, can choose a defect tolerance technique that best matches with their design's specifications. For example, the design configuration *S +CL_1SP_IR*, where automatic clustering decomposition is applied at system level with one dedicated spare for each partition, where 10% of the parts will get defected after 7 years but with 48% less cost in area than design configuration *S+CL_2SP_IR* might be a more attractive solution.

The same data as in Fig. 14a, is presented in Fig. 14b, with the difference that here we assume that the breakdown for the switch design starts after 10 years of being shipped. For the first three design configurations, there is no difference since by that time all of the parts become defective. For the other two design configurations, what is interesting to observe is that even after the breakdown point where the failure rates increase with an exponential rate, most of the parts will be able to provide the user a warning time window of a month before failure. This is a very important feature for a design configuration, especially for designs that require high levels of dependability.

9. CONCLUSIONS AND FUTURE DIRECTIONS

As silicon technologies continue to scale, transistor reliability is becoming an increasingly important issue. Devices are becoming subject to extreme process variation, transistor wear-out, and manufacturing defects. As a result, it will likely be no longer possible to create fault-free designs. In this article, we investigate the design of a defect-tolerant CMP network switch. To accomplish this design, we first develop a high-level, architect-friendly model of silicon failures, based on the time-tested bathtub curve. Based on this model, we explore the design space of defect-tolerant CMP switch designs and the resulting trade-off between defect tolerance and area overhead. We find that traditional mechanisms, such as triple modular redundancy and error correction codes, are insufficient for tolerating moderate numbers of defects. Rather, domain-specific techniques that include end-to-end error detection, resource sparing,

and iterative diagnosis/reconfiguration are more effective. Further, decomposing the netlist of the switch into modest-sized clusters is the most effective granularity to apply the protection techniques.

This work provides a solid foundation for future exploration in the area of defect-tolerant design. We plan to investigate the use of spare components, based on wearout profiles to provide more sparing for the most vulnerable components. Further, a CMP switch is only a first step toward the overreaching goal of designing a defect-tolerant CMP system.

ACKNOWLEDGMENTS

We would like to acknowledge Li-Shiuan Peh for providing us access to CMP switch models, Doug Burger for providing CMP network reference traces, and the anonymous reviewers for providing useful comments on this article. This work was supported in part by grants from NSF and Gigascale Systems Research Center.

REFERENCES

AL-ASAAD, H. AND HAYES, J. P. 1995. Design verification via simulation and automatic test pattern generation. In *International Conference on Computer Aided Design (ICCAD '95)*. IEEE Computer Society Press, Washington, D.C. 174–180.

BARNETT, T. S. AND SINGH, A. D. 2003. Relating yield models to burn-in fall-out in time. In *Proceedings of International Test Conference (ITC)*. 77–84.

BOHL, E., LINDENKREUZ, T., AND STEPHAN, R. 1997. The fail-stop controller AE11. In *International Test Conference (ITC '97)*. IEEE, Washington, D.C. 567–577.

BORKAR, S., KARNIK, T., AND DE, V. 2004. Design and reliability challenges in nanometer technologies. In *Proceedings of the Design Automation Conference (DAC)*. 75.

BOSSEN, D. C., KITAMORN, A., REICK, K., AND FLOYD, M. S. 2002. Fault-tolerant design of the IBM pseries 690 system using POWER4 processor technology. *IBM Journal of Research and Development 46*, 1, 77–86.

BOWER, F. A., SHEALY, P. G., OZEV, S., AND SORIN, D. J. 2004. Tolerating hard faults in microprocessor array structures. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*. 51–60.

DALLY, W. J., DENNISON, L. R., HARRIS, D., KAN, K., AND XANTHOPOULOS, T. 1994. The reliable router: A reliable and high-performance communication substrate for parallel computers. In *Proceedings of International Workshop on Parallel Computer Routing and Communication (PCRCW)*. 241–255.

GOMAA, M., SCARBROUGH, C., VIJAYKUMAR, T. N., AND POMERANZ, I. 2003. Transient-fault recovery for chip multiprocessors. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-03)*. 98–109.

GUPTA, P. AND KAHNG, A. B. 2003. Manufacturing-aware physical design. In *Proc. of International Conference on Computer-Aided Design (ICCAD)*. 681–688.

HU, C. K. AND ROSENBERG, R. 1999. Scaling the effect on electromigration in on-chip CU wiring. In *International Electron Devices Meeting*.

IONESCU, A. M., DECLERCQ, M. J., MAHAPATRA, S., BANERJEE, K., AND GAUTIER, J. 2002. Few electron devices: towards hybrid CMOS-SET integrated circuits. In *Proceedings of the Design Automation Conference (DAC)*. 88–93.

ITRS. 2004. International technology roadmap for semiconductors (ITRS) 2004 update. Document available at *http://public.itrs.net/*.

J. E. D. E. COUNCIL. 2002. Failure mechanisms and models for semiconductor devices. JEDEC Publication JEP122-A.

KARYPIS, G. AND KUMAR, V. 1998. hMETIS: A hypergraph partitioning package.

KARYPIS, G., AGGARWAL, R., KUMAR, V., AND SHEKHAR, S. 1997. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings of the Design Automation Conference (DAC)*. 526–529.

MESSER, A., BERNADAT, P., FU, G., CHEN, D., DIMITRIJEVIC, Z., LIE, D. J. F., MANNARU, D., RISKA, A., AND MILOJICIC, D. S. 2004. Susceptibility of commodity systems and software to memory soft errors. *IEEE Trans. Computers 53*, 12, 1557–1568.

MUKHERJEE, S. S., KONTZ, M., AND REINHARDT, S. K. 2002. Detailed design and implementation of redundant multithreading alternatives. In *Proceedings of the 29th International Symposium on Computer Architecture (ISCA-02)*. 99–110.

MUKHERJEE, S. S., WEAVER, C., EMER, J. S., REINHARDT, S. K., AND AUSTIN, T. M. 2003. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings International Symposium on Microarchitecture (MICRO)*. 29–42.

MUKHERJEE, S. S., EMER, J. S., AND REINHARDT, S. K. 2005. The soft error problem: An architectural perspective. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*. 243–247.

MURRAY, B. T. AND HAYES, J. P. 1996. Testing ICs: Getting to the core of the problem. *IEEE Computer 29*, 11, 32–38.

PEH, L.-S. 2001. Flow control and micro-architectural mechanisms for extending the performance of interconnection networks. Ph.D. thesis, Stanford University.

RAO, R., SRIVASTAVA, A., BLAAUW, D., AND SYLVESTER, D. 2003. Statistical estimation of leakage current considering inter- and intra-die process variation. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. 84–89.

REINHARDT, S. K. AND MUKHERJEE, S. S. 2000. Transient fault detection via simultaneous multithreading. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*. 25–36.

RIESS, B. M. AND ETTELT, G. G. 1995. Speed: Fast and efficient timing driven placement. In *Proceedings of International Symposium on Circuits and Systems (ISCAS)*. 377–380.

ROTENBERG, E. 1999. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In *Proceedings of International Symposium on Fault-Tolerant Computing (FTCS)*. 84–91.

SANKARALINGAM, K., NAGARAJAN, R., LIU, H., KIM, C., HUH, J., BURGER, D., KECKLER, S. W., AND MOORE, C. R. 2003. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-03)*. 422–433.

SAXENA, N. AND MCCLUSKEY, E. 1998. Dependable adaptive computing systems. In *IEEE Systems, Man, and Cybernetics Conference*.

SHIVAKUMAR, P., KISTLER, M., KECKLER, S. W., BURGER, D., AND ALVISI, L. 2002. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. 389–398.

SHIVAKUMAR, P., KECKLER, S. W., MOORE, C. R., AND BURGER, D. 2003. Exploiting microarchitectural redundancy for defect tolerance. In *Proceedings of International Conference on Computer Design (ICCD)*. 481–488.

SIEWIOREK, D. P. AND SWARZ, R. S. 1998. *Reliable Computer Systems: Design and Evaluation*, 3rd ed. AK Peters Ltd, Wellesly, MA, USA.

SMOLENS, J. C., GOLD, B. T., KIM, J., FALSAFI, B., HOE, J. C., AND NOWATZYK, A. 2004. Fingerprinting: bounding soft-error detection latency and bandwidth. In *Proc. of the Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 224–234.

SPAINHOWER, L. AND GREGG, T. A. 1998. G4: A fault-tolerant CMOS mainframe. In *Proc. of International Symposium on Fault-Tolerant Computing (FTCS)*. 432–440.

SRINIVASAN, J., ADVE, S. V., BOSE, P., AND RIVERS, J. A. 2004. The impact of technology scaling on lifetime reliability. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*. 177.

STATHIS, J. H. 2002. Reliability limits for the gate insulator in CMOS technology. *IBM Journal of Research and Development 46*, 2-3, 265–286.

VRUDHULA, S. B. K., BLAAUW, D., AND SIRICHOTIYAKUL, S. 2002. Estimation of the likelihood of capacitive coupling noise. In *Proceedings of the Design Automation Conference (DAC)*. 653–658.

WANG, N. J., QUEK, J., RAFACZ, T. M., AND PATEL, S. J. 2004. Characterizing the effects of transient faults on a high-performance processor pipeline. In *Proceedings of International Conference on Dependable Systems and Networks (DSN)*. 61.

WEAVER, C. AND AUSTIN, T. 2001. A fault tolerant approach to microprocessor design. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN '01)*. IEEE, Washington, D.C. 411–420.

WEAVER, C., EMER, J. S., MUKHERJEE, S. S., AND REINHARDT, S. K. 2004. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. 264–275.

WU, E. ET AL. 2002. Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate dioxides. *Solid State Electronics Journal*.

ZIEGLER, J. F. 1996. Terrestrial cosmic rays. *IBM Journal of Research and Development 40*, 1 (Jan.), 19–39.

ZIEGLER, J. F. ET AL. 1996. IBM experiments in soft fails in computer electronics (1978–1994). *IBM Journal of Research and Development 40*, 1 (Jan.), 3–18.

Received March 2006; revised July 2006; accepted August 2006