

BulletProof: A Defect-Tolerant CMP Switch Architecture

Kypros Constantinides[‡] Stephen Plaza[‡] Jason Blome[‡] Bin Zhang[†]
Valeria Bertacco[‡] Scott Mahlke[‡] Todd Austin[‡] Michael Orshansky[†]

[‡]Advanced Computer Architecture Lab
University of Michigan
Ann Arbor, MI 48109
{kypros, splaza, jblome, valeria,
mahlke, austin}@umich.edu

[†]Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX, 78712
bzhang@ece.utexas.edu
orshansky@mail.utexas.edu

Abstract

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 paper, 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 multi-processor (CMP) system to study in detail, a single CMP router switch. To start, we develop a unified model of 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 online repair and recovery capabilities. Protection is considered at multiple levels from the entire system down through arbitrary partitions of the design. To better understand the impact of these faults, we evaluate our CMP switch designs using circuit-level timing on detailed physical layouts. Our experimental results are quite illuminating. We find that designs are attainable that can tolerate a larger number of defects with less overhead than naïve triple-modular redundancy, using domain-specific techniques such as end-to-end error detection, resource sparing, automatic circuit decomposition, and iterative diagnosis and reconfiguration.

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 [23]. 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, 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 [8]. 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 wearout, 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 paper. The horizontal axis lists the type of device-level faults that systems might experience. The source of failures are widespread, ranging from transient faults due to energetic particle strikes [32] and electrical noise [28], to permanent wearout faults caused by electromigration [13], stress-migration [8], and dielectric breakdown [10]. The vertical axis of Figure 1 lists design solutions to deal with faults. Design solutions range from ignoring any possible faults (as is done in many systems today), to detecting and reporting faults, to detecting and correcting faults, and finally fault correction with repair capabilities. The final two design solutions are the only solutions that can address permanent faults, with the final solution being the only approach that maintains efficient operation after encountering a silicon defect.

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, due to energetic particles in particular, will grow in future technologies [5, 16]. 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) [23], and logic, including dual or triple-modular spatial redundancy [23] or time-redundant computation [24] or checkers [30]. 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 [7, 17, 29].

In contrast, little attention has been paid to incorporating design tolerance for permanent faults, such as silicon defects and transistor wearout. The typical approach used today is to reduce the likelihood of encountering silicon faults through post-manufacturing burn-in, a process that accelerates the aging process as devices are subjected to elevated temperature and voltage [10]. The burn-in process accelerates the failure of weak transistors, ensuring that, after burn-in, devices still working are composed of robust transistors. Additionally, many computer vendors provide the ability to repair faulty memory and cache cells, via the in-

TYPE OF DEFECT DESIGN FEATURE	MANUFACTURING DEFECT			WEAR-OUT DEFECT	TRANSIENT ERROR
	NO-DETECTION	Unstable Defects	System fails in unpredictable way	System glitch manifests in unpredictable way	
DETECTION	Testing	Component terminates at first error	Component terminates. Hard-reset restores	DMR	DMR
DETECTION +CORRECTION	Post-manufacturing recovery ECC · memory	Online defect recovery	Transient fault recovery	TMR	Diva Razor ECC TMR
DETECTION +CORRECTION +REPAIR	Post-manufacturing reconfiguration cache-line swap-out memory-array spares	Online repair		Bulletproof	

Mainstream Solutions | High-end Solutions | Specialized Solutions | Research-stage Solutions

Figure 1: *Reliable System Design Space.* The diagram shows a map of type of device-level faults in a digital system (horizontal axis) vs. protection techniques against these faults (vertical axis). This work addresses the problems/solutions in the dark shaded area of the map.

clusion of spare storage cells [25]. Recently, academics have begun to extend these techniques to support sparing for additional on-chip memory resources such as branch predictors [6] and registers [22].

1.1 Contributions of This Paper

In this paper, we push forward the understanding in reliable microarchitecture design by performing a comprehensive design study of the effects of 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-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 we 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 *BulletProof*, is capable of tolerating silicon defects, transient faults, and transistor wearout. 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 paper is organized as follows. Section 2 gives additional background on the faults of interest in this study and introduces our architect-friendly fault model based on the bathtub curve. Section 3 presents our fault simulation infrastructure, and examines the exposure of the baseline design to permanent faults. Section 4 introduces the techniques we have employed in our CMP switch designs to provide cost-effective tolerance of transient and permanent faults. In Section 5, 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 6 gives conclusions and suggestions for future research directions.

2. An Analysis of the Fault Landscape

As silicon technologies progress into the 65nm 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.

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 [32]. The effect of SEU 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 [31, 29], 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.

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 [20], resulting in structurally weak and poor performing devices. Designers are forced to deal with these variations by assuming worst-case device characteristics (usually, a 3-sigma variation from typical conditions), which leads to overly conservative designs.

Manufacturing Defects. Deep sub-micron 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. Post-manufacturing test [18] and built-in self-test (BIST) [1] 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 [4].

Gate Oxide Wearout. Technology scaling has adverse effects on the lifetime of transistor devices, due to time-dependent wearout. There are three major failure modes for time-dependent wearout: electromigration, hot carrier degradation (HCD), and time-dependent oxide breakdown. Electro-migration 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 [11]. 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 [14]. Time-dependent oxide breakdown is due to the extensive use of ultra-thin oxide for high performance. The rate of defect generation in the oxide is proportional to the current density flowing through it, and therefore is increasing drastically

as a result of relentless down-scaling [27].

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 sub-micron 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 results 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.

2.1 The Bathtub: A Generic Model for Semiconductor Hard Failures

To derive a simple architect-friendly model of failures, we step back and return to the basics. 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 Figure 2. We will adopt this time-tested 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 it 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 due to the occasional breakdown of weak transistors or interconnect.
- **Breakdown Period:** During this period, failures occur with increasing frequency over time due to age-related wearout. Many devices will enter this period at roughly the same time, creating an avalanche effect and a quick rise in device failure rates.

With the respect to Figure 2, the model is represented with the following equations:

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

(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,

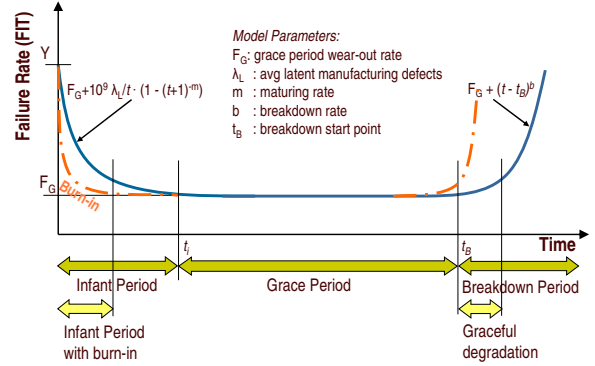


Figure 2: Simple bathtub curve model of device defect exposure. The curve indicates the qualitative trend of failure rates for a silicon part over time. The initial operational phase and the “aged-silicon” phase are characterized by much higher failure rates.

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.

Latent Manufacturing Defects per Chip (λ_L): Previous work [3], 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 burn-in test. Based on these models, the average number of latent manufacturing defects per chip ($140mm^2$) 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.

Grace Period Failure Rate (F_G): For the grace period failure rate, we use reference data by [26]. In [26], 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-dependence dielectric breakdown, and thermal cycling. For a predicted post-65nm fabrication technology, we adopt their worst-case failure rate (F_G) of 55,000 FITs.

Breakdown Period Start Point (t_B): Previous work [27], 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 65nm CMOS at 1.0V supply voltage. We were unable to find any predictions as to how this value will trend for fabrication technologies beyond 65nm, 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.

3. A Fault Impact Evaluation Infrastructure

In [7], we introduced a high-fidelity simulation infrastructure for quantifying various derating effects on a design’s overall soft-error rate. This infrastructure takes into account circuit level phenomena, such as time-related, logic-related and microarchitectural-related fault masking. Since many tolerance techniques for permanent errors can be adapted to also provide soft error tolerance, the remainder of this work concentrates on the exploration of various defect tolerance techniques.

3.1 Simulation Methodology for Permanent Faults

In this section, we present our simulation infrastructure for evaluating the impact of silicon defects. We create a model for permanent fault parameters, and develop the infrastructure necessary to evaluate the effects on a given design.

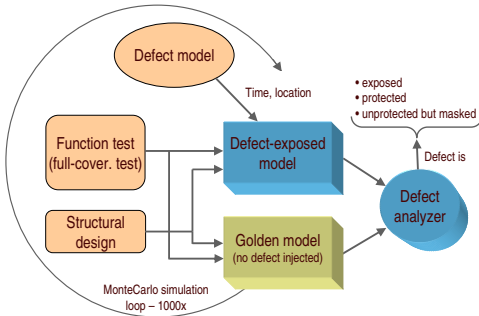


Figure 3: Simulation infrastructure for permanent faults. The defect infrastructure uses two models of the system, simulated in parallel. Defects are uniformly distributed in time and space and the input stimuli is a full coverage test that activates each internal circuit node of the system. A fault analyzer classifies defects based on the system response.

Figure 3 shows our simulation framework for evaluating the impact of silicon defects on a digital design. The framework consists of an event-driven simulator that simulates two copies of the structural, gate-level description of the design in parallel. Of these two designs, one copy is kept intact (golden model), while the other is subject to fault injection (defect-exposed model). The structural specification of our design was synthesized from a Verilog description using the Synopsys Design Compiler.

Our silicon defect model distributes defects in the design uniformly in time of occurrence and spatial location. Once a permanent failure occurs, the design may or may not continue to function depending on the circuit’s internal structure and the system architecture. The defect analyzer classifies each defect as exposed, protected or unprotected but masked. In the context of defect evaluation, faults accumulate over time until the design fails to operate correctly. The defect that brings the system to failure is the last injected defect in each experiment and it is classified as exposed. A defect may be protected if, for instance, it is the first one to occur in a triple-module-redundant design. An unprotected but masked defect is a defect that it is masked because it occurs in a portion of the design that has already failed, for

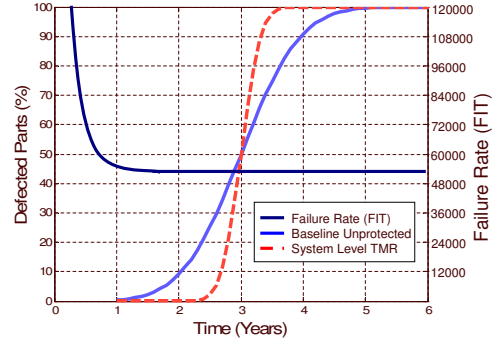


Figure 4: Baseline design reliability. The graph superimposes the FIT rates of the bathtub model with the fault tolerance of two variants of the CMP switch design: a baseline unprotected version and a variant with a traditional TMR technique.

example a defect hitting an already failed module of a TMR design.

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 input stimuli is a full coverage test, crafted to excite all internal nodes of the design while observing the outputs. If any of the stimuli impact the output correctness, the implication is that there is at least one execution that can expose the defect, and thus such defect is considered exposed.

Finally, to gain statistical confidence of the results, we run the simulations described many times in a Monte Carlo modeling framework.

3.2 Reliability of the Baseline CMP Switch Design

The first experiment is an evaluation of the reliability of the baseline CMP switch design. In Figure 4, we used the bathtub curve fitted for the post-65nm technology node as derived in Section 2. The FIT rate of this curve is 55000 during the grace period, which corresponds to a mean time to failure (MTTF) of 2 years. We used this failure rate in our simulation framework for permanent failures and we 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 Figure 4 shows that in a large parts population, 50% of the parts will be defective by the end of the second year after shipment, and by the fourth year almost all parts will have failed. In this experiment, we have also analyzed a design variant which 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 at Section 5) to present better defect tolerance.

The TMR model used in this analysis is the classical 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 multiplied.

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 [21], 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 paper whenever TMR is applied, the classical TMR model is assumed.

From Figure 4, the simulation-based analysis finds that TMR provides very little reliability improvements over the baseline designs, due to the few number of defects that can be tolerated by system-level TMR. Furthermore, the area of the TMR protected design is more than three times the area 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 paper, we propose and evaluate defect-tolerant techniques that are significantly more robust and less costly than traditional defect-tolerant techniques.

4. Self-repairing CMP Switch Design

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.

4.1 Baseline Design

The baseline design, consists of a CMP switch similar to the one described in [19]. 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 Figure 5.

The implemented router is composed of four functional modules: the input controller, the switch arbiter, the cross-bar 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 to create a gate-

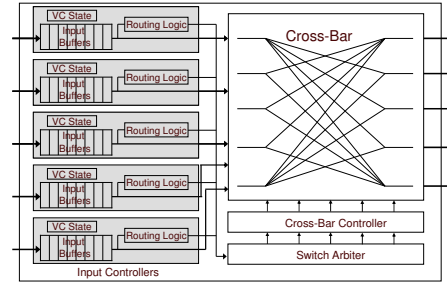


Figure 5: Baseline CMP switch design. A high level block diagram for a wormhole interconnection switch is presented. It consists of 5 input controllers, a cross-bar, a switch arbiter and a cross-bar controller.

level netlist, which consists of approximately 10k gates. This router design consists of five input controllers which dominate the design’s area (86%). Also, the design is heavily dominated by combinational logic, which represents 84% of the total area, making it critical to choose protection techniques that can tolerate errors in logic effectively.

4.2 Protection Mechanisms

A design that is tolerant to permanent defects must provide mechanisms that perform four central actions 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 [9] 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 and provide a higher reliability multiprocessor interconnection network.

4.2.1 General Techniques

The most commonly used protection mechanisms are dual and triple modular redundancy, or DMR and TMR [23]. 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, recovery, and repair. In TMR, the majority voter identifies a malfunctioning hardware component and masks its affects on the primary outputs. Hence, repair is trivial since the defective component is always just simply outvoted when it computes an incorrect value. Due to 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) [23]. 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 due to the small fraction of area that holds state.

4.2.2 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, resource sparing, system diagnosis, and reconfiguration.

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 functional incorrectness, 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 Figure 6(a). When an error is detected by a CRC checker, all CRC checkers are notified about the error detection and block any further flit routing. 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 Figure 6(a), 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

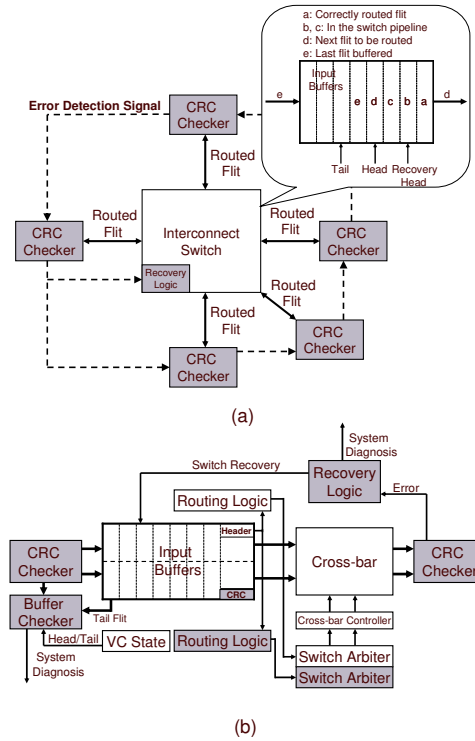


Figure 6: End-to-End error detection and recovery mechanism. In part (a) the interconnection switch is enhanced by Cyclic Redundancy Checkers (CRC) and recovery logic for providing data corrupting error detection. The input buffers are enhanced with an extra recovery head pointer to mark the last correctly checked flit. In part (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.

switch recovery, the recovery head pointer is assigned to the head pointer for all five input buffers, and the switch recovers operations 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 errors causing functional incorrectness 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 Figure 6(b), 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, and it 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 Figure 6(b). 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 routed through the cross-bar independently. 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 cross-bar routes the flits at the bit-level, the only difference is that the responses to the cross-bar 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 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.

Resource Sparing. 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 its need of having the ability to replace more than one possible defective partitions.

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

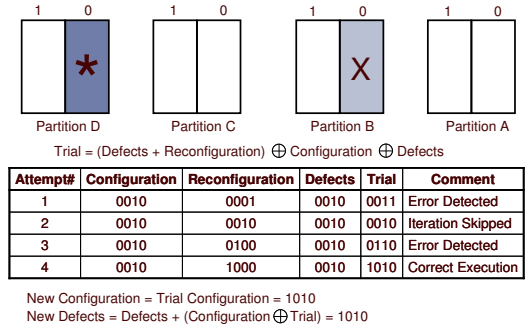


Figure 7: Example system diagnosis and reconfiguration. This example shows the system with four partitions and one spare for each partition. The first spare of partition B contains a previously detected and corrected defect, thus the latest error in execution is caused by the defect in the first spare of partition D.

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.

In order for the system diagnosis to operate, it maintains a set of bit vectors as follows:

- Configuration Vector: It indicates which spare partitions are enabled.
- Reconfiguration Vector: It keeps track of which configurations have been tried and indicates the next configuration to be tried. It gets updated at each iteration of system diagnosis.
- Defects Vector: It keeps track of which spare partitions are defected.
- Trial Vector: Indicates which spare partitions are enabled for a specific system diagnosis iteration.

Figure 7, demonstrates an example where system diagnosis is applied on a system with four partitions and two copies (one spare) for each partition. The first copy of partition B has a detected defect (mapped at the Defects Vector). The defect in the first copy of partition D is a recently manifested defect and is the one that caused erroneous execution. Once the error is detected, the system recovers to the last correct state using the mechanism described in the previous section (see error detection and recovery mechanism), and it initializes the Reconfiguration Vector. Next, the Trial Vector is computed using the Configuration, Reconfiguration, and Defects vectors. In case the Trial Vector is the same with the Configuration Vector (attempt 2), due to a defected spare, the iteration is skipped. Otherwise, the Trial Vector is used as the current Configuration vector, indicating which spare partitions will be enabled for the current trial. The execution is then replayed from the recovery point until the error detection point. In case the error is detected, a new trial is initiated by updating the Reconfiguration Vector and re-computing the Trial Vector. In case no error is detected, meaning that the trial configuration is a working configuration, the Trial Vector is copied to the Configuration Vector and the Defects Vector is updated with the located defected copy. If all the trial configurations are exhausted, which are equal to the number of partitions, and no working configuration was found, then the defect was a fatal defect and the

system won't be able to recover. The example implementation of the system diagnosis mechanism demonstrated in Figure 7, can be adapted accordingly for designs with more partitions and more spares.

We also consider the Built-In-Self-Test (BIST) technique 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, these test vectors are applied to each partition of the system through scan chains to check its functionality correctness and locate the defected partition.

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 possible chip multiprocessor design.

4.2.3 Level of Protection

The error resiliency achieved by implementing one of the protection techniques (e.g., TMR or sparing) is highly dependent on the granularity of the partitions. In general, the larger the granularity of the partitions, the less robust the design. However, as the granularity of the partition becomes smaller, more logic is required. For TMR, each output for a given partition requires a MAJORITY gate. Since each added MAJORITY gate is unshielded from permanent defects, poorly constructed small partitions can make a design less error resilient than designs with larger partitions.

To illustrate these trade-offs, consider the baseline switch again in Figure 5. Sparing and TMR can be done on the system-level where the whole switch is replicated and each output requires some extra logic like a MUX or MAJORITY gate. A single permanent error makes one copy of the switch completely broken. However, the area overhead beyond the spares 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 spare along with the arbiter, cross-bar, and the cross-bar 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. There is a small area penalty for this approach 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, consider partitioning at the gate-level. In this approach, 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 for TMR, the area would be four times the original design. In addition, because each added gate is unprotected, the susceptibility of this design to errors is actually greater than the larger partitions used in the component-based partitioning.

The previous analysis shows that the level of partitioning effects the error resiliency and the area overheads of the design. In this paper, we introduce a technique called, *Automatic Cluster Decompositions*, that generates partitions that minimizes area overhead while maximizing error resiliency.

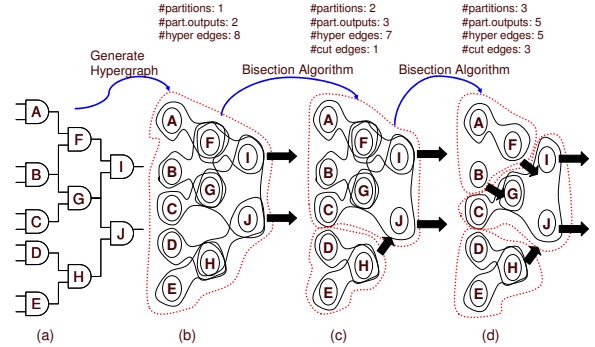


Figure 8: The process of automatic cluster decomposition. In part (a) a sample netlist is shown with 2 primary outputs, along with its corresponding hypergraph in part (b). Part (c) shows the hypergraph after a min-cut bisection creating two unbalanced partitions. Part (d) shows the final 3-way partitioning resulting from a bisection of the largest partition.

4.2.4 Automatic Cluster Decomposition

Automatic Cluster Decomposition 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 graph that can then be partitioned using a balanced-recursive min-cut algorithm [15, 12] that has found use in fields like VLSI [2].

Figure 8 shows how these partitions are generated from the netlist of a design. First, the netlist pictured in part (a) is used to generate a *hypergraph* shown in part (b). A hypergraph is an extension of a normal graph where one edge can connect multiple vertices. In the figure, each vertex represents a separate net in the design. A *hyperedge* is drawn around each net and its corresponding fanout. If that net is placed in a different partition than one of its fanout, that net becomes an output for its partition thus increasing the overhead of the partition. Thus the goal of the partitioning algorithm is to minimize the number of hyperedges that are cut. For this example, we show a 3-way partitioning of the circuit. The algorithm in [12] performs a recursive min-cut operation where the original circuit is bisected and then one of these partitions is bisected again. In Figure 8(c), the hypergraph is bisected and the number of hyperedges cut is reported. Notice that one of those pieces is twice the size of the other one. Because 3-way partitioning is desired, one piece is slightly larger so that the final partitions are fairly balanced where each partition has about the same number of vertices/nets. Figure 8(d) 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. In practice, several iterations of [12] are run as the algorithm is heuristically-based and requires many runs for optimal partitions. Also, some imbalance in partition sizes is tolerated if the number of hyperedges cut is significantly smaller as a result.

Once the hypergraph is partitioned, several different replication strategies can be used such as sparing and TMR. In Figure 9, an example of performing 3-way partitioning over an arbitrary piece of logic using one spare per partition is shown. In part (a), sparing is performed at the system-level.

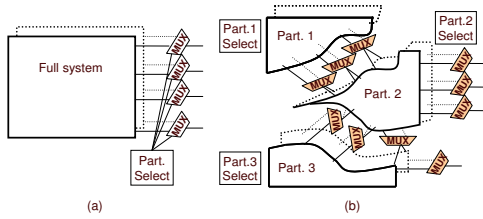


Figure 9: Examples of one spare systems. In part (a) sparing without any cluster decomposition is shown. In part (b) sparing is applied to a three-way partition. Cluster decomposition increases MUX interconnect overhead, but provides higher protection due to the smaller granularity of the sparing.

The outputs of both identical units are fed into a MUX where a register value determines which unit is active and which one is inactive. In part (b), the circuit is partitioned into three pieces. Notice, that the outputs for each boundary must now have a MUX associated with it. Also, each partition requires a register to determine which copy is active. Thus 3-way partitioning requires a total of three registers and the number of MUXes corresponding to the number of outputs generated by the partitioning.

4.3 Switch Designs

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 give a name convention as follows: `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 inherently provides 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 1, describes the choices that we considered in our simulated configurations for the three parameters, and it gives some example configurations along with their name conventions.

5. Experimental Results

To evaluate the effectiveness of our domain specific defect tolerance techniques in protecting the switch design, we simulated various design configurations with both traditional and domain specific techniques. To assess the effectiveness of the various design configurations in protecting the switch design, we take into account the area and time overheads of the design along with the mean number of defects that the design can tolerate. We also 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 with the area overhead of the protection techniques. In other words, the higher the SPF factor, the more resilient each transistor is to defects. Since the number of defects in a design is proportional to the area of the design, the use of this metric for assessing the effectiveness of the silicon protection techniques is more appropriate.

In Table 2, we list the design configurations that we simulated. The naming convention followed for representing each configuration is described in Table 1. For each simulated

Table 1: Mnemonic table for design configurations. For each portion of the naming convention, we show the possible mnemonics with the related description. The last portion provides some example design configurations.

Mnemonic Group	Mnemonic	Description
Level of applying defect tolerance technique	S	System level
	C	Component level
	G	Gate level
	S+CL	System level clusters
Defect tolerance techniques (can be applied in combinations)	TMR	Triple Modular Redundancy
	#SP #SH(X) ECC	# dedicated spares for each partition # shared spares for partition of type X Error Correction Codes applied at state
System diagnosis technique	IR	Iterative replay
	BIST	Built-In-Self-Test
Example configurations	S+CL_1SP_IR C_2SH(IC)+1SP_BIST	System level clusters with 1 spare for each partition and iterative replay. Component level with 2 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.

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, recovery and system diagnosis (different configurations have different requirements for the extra logic added). We notice 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 and logic needed for the scan chains. Even though the area overhead for the test vectors can be shared over the total number of switches per chip, the area overhead of the BIST technique is still rather large. Another design configuration with high area overhead is the one where TMR is applied at the gate level due to 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 is dependent 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 automatic cluster decomposition 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 in a logic dominated design undesirable.

The SPF values for each design are presented in the fifth column of Table 2. 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's interesting that two design configurations have SPFs lower than 1. The first

Table 2: Results of the evaluated designs. 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.

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	G_TMR	4.00	4.00	1.00	10540	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

Key	Design Configuration	Area O.head	Defects	SPF	#Part.	%Dly
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_R	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

one is TMR applied at the system level, which can tolerate 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. This parameter is very important for the configurations employing ACD. The SPF of a given design configurations, is greatly dependent on the number of partitions in the decomposed design.

Figure 10 shows the dependency of the SPF over the number of decomposed partitions for the design configuration $S+CL_1SP_IR$. 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, and 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. For each design configuration employing automatic cluster decomposition, we ran several simulations for varying numbers of partitions to achieve an optimal SPF.

The final column in Table 2, %Delay, gives the percentage increase of the critical path delay in the switch. We produce a coarse approximation of the delay overheads that is technology independent by making the delay increase proportional to the number of interconnection gates added to the critical path. Thus, TMR-based designs will achieve the same delay increase as spare-based designs as multiplexers and majority gates are treated the same in the analysis. 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

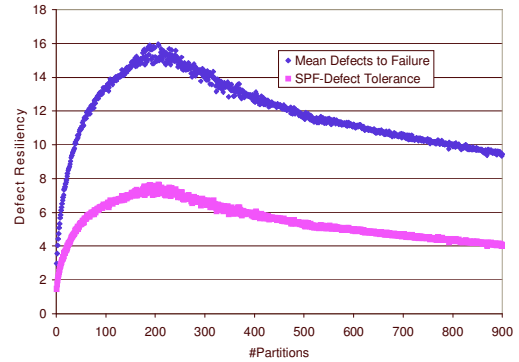


Figure 10: Defect resiliency as a function of the number of partitions. As an example we plot the SPF defect tolerance of configuration $S+CL_1SP_IR$ for a varying number of partitions generated by the ACD algorithm.

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 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 Figure 11 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 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 right-most design configuration, $S+CL_2SP_IR$, provides a defect tolerance of 11.11 SPF, by employing automatic cluster decomposition at the system level with 200 partitions and two extra spares for each partition, along with iterative replay for system diagnosis. The area overhead for implementing this design configuration is 3.42X, and provides the better trade-off between area required and offered defect

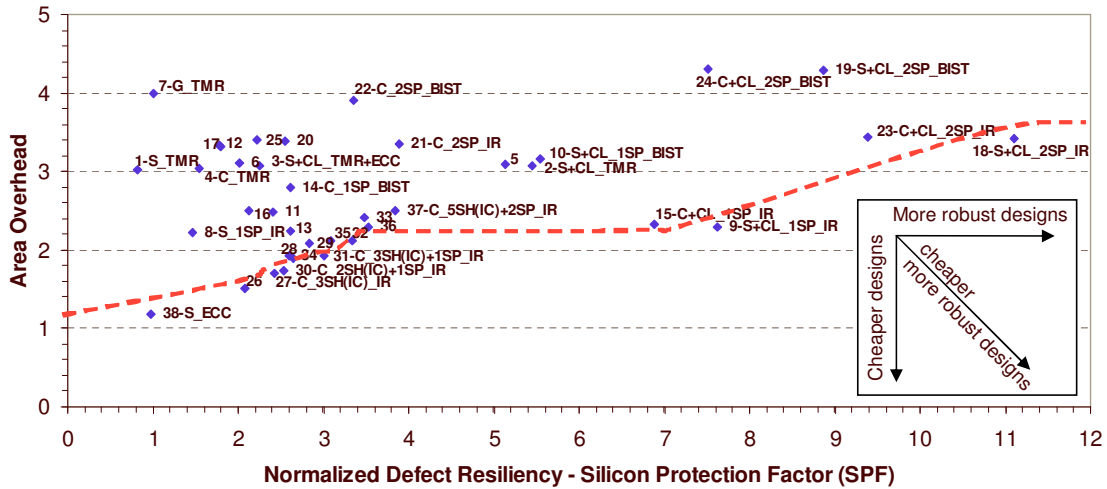


Figure 11: Pareto chart of the explored solutions. The design evaluated are plotted on an area vs. SPF chart. The line across the chart connects the set of optimal solutions. See Table 1 for explanations of design points.

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 overhead less than 2X, but offering SPFs of 2.5-3 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 Figure 12, we present how some of the design configurations affect the lifetime of the switch design for a future post 65nm technology where the mean time between failures to be manifested on a switch is 2 years (a failure rate of 55000 FITs). The graph's horizontal axis represents the years that the switch design is operating. The vertical axis represent 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 Figure 12(a), 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 1, 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 Figure 12(a), we can observe that when applying TMR at the component level, 25% of the shipped parts will be defected by the first year and 75% after the first

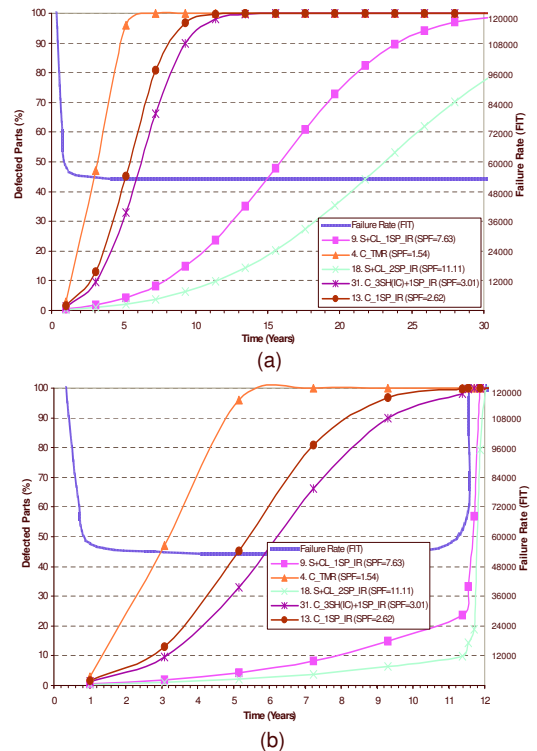


Figure 12: Fault tolerance of some interesting design configurations. Part (a) superimposes the FIT rate of the bathtub model with the percentage of defective parts over time. In addition part (b) takes into account the breakdown period.

three years. On the other hand, when in a design configuration where automated cluster decomposition was applied at the system level with 2 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 Figure 12(a), is presented in Figure 12(b), 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 very critical high dependable applications.

6. 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 wearout, and manufacturing defects. As a result, it will likely be no longer possible to create fault-free designs. In this paper, 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 tradeoff 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 towards the over-reaching goal of designing a defect-tolerant CMP system.

Acknowledgments: This work is supported by grants from NSF and Gigascale Systems Research Center. We would also like to acknowledge Li-Shiuan Peh for providing us access to CMP Switch models, and the anonymous reviewers for providing useful comments on this paper.

7. References

- [1] H. Al-Asaad and J. P. Hayes. Logic design validation via simulation and automatic test pattern generation. *J. Electron. Test.*, 16(6):575–589, 2000.
- [2] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: a survey. *Integr. VLSI J.*, 19(1-2):1–81, 1995.
- [3] T. S. Barnett and A. D. Singh. Relating yield models to burn-in fall-out in time. In *Proc. of International Test Conference (ITC)*, pages 77–84, 2003.
- [4] E. Bohl, et al. The fail-stop controller AE11. In *Proc. of International Test Conference (ITC)*, pages 567–577, 1997.
- [5] S. Borkar, et al. Design and reliability challenges in nanometer technologies. In *Proc. of the Design Automation Conf.*, 2004.
- [6] F. A. Bower, et al. Tolerating hard faults in microprocessor array structures. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*, 2004.
- [7] K. Constantinides, et al. Assessing SEU Vulnerability via Circuit-Level Timing Analysis In *Proc. of 1st Workshop on Architectural Reliability (WAR)*, 2005.
- [8] J. E. D. E. Council. Failure mechanisms and models for semiconductor devices. *JEDEC Publication JEP122-A*, 2002.
- [9] W. J. Dally, et al. The reliable router: A reliable and high-performance communication substrate for parallel computers. In *Proc. International Workshop on Parallel Computer Routing and Communication (PCRCW)*, 1994.
- [10] E. Wu, et al. Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate dioxides. *Solid-state Electronics Journal*, 2002.
- [11] P. Gupta and A. B. Kahng. Manufacturing-aware physical design. In *Proc. of International Conference on Computer-Aided Design (ICCAD)*, 2003.
- [12] hMETIS. <http://www.cs.umn.edu/~karypis>.
- [13] C. K. Hu, et al. Scaling effect on electromigration in on-chip Cu wiring. *International Electron Devices Meeting*, 1999.
- [14] A. M. Ionescu, M. J. Declercq, S. Mahapatra, K. Banerjee, and J. Gautier. Few electron devices: towards hybrid CMOS-SET integrated circuits. In *Proc. of the Design Automation Conference*, pages 88–93, 2002.
- [15] G. Karypis, et al. Multilevel hypergraph partitioning: Applications in VLSI domain. In *Proc. of the Design Automation Conference*, pages 526–529, 1997.
- [16] S. Mukherjee, et al. The soft error problem: An architectural perspective. In *Proc. of the International Symposium on High-Performance Computer Architecture*, 2005.
- [17] S. Mukherjee, et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 29–42, 2003.
- [18] B. T. Murray and J. P. Hayes. Testing ICs: Getting to the core of the problem. *IEEE Computer*, 29(11):32–38, 1996.
- [19] L.-S. Peh. *Flow Control and Micro-Architectural Mechanisms for Extending the Performance of Interconnection Networks*. PhD thesis, Stanford University, 2001.
- [20] R. Rao, et al. Statistical estimation of leakage current considering inter- and intra-die process variation. In *Proc. of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 84–89, 2003.
- [21] N.R. Saxena, and E.J. McCluskey Dependable Adaptive Computing Systems. *IEEE Systems, Man, and Cybernetics Conf.*, 1998.
- [22] P. Shivakumar, et al. Exploiting microarchitectural redundancy for defect tolerance. In *Proc. of International Conference on Computer Design (ICCD)*, 2003.
- [23] D. P. Siewiorek, et al. *Reliable computer systems: Design and evaluation*, 3rd edition. *AK Peters, Ltd Publisher*, 1998.
- [24] J. Smolens, et al. Fingerprinting: Bounding the soft-error detection latency and bandwidth. In *Proc. of the Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004.
- [25] L. Spainhower and T. A. Gregg. G4: A fault-tolerant CMOS mainframe. In *Proc. of International Symposium on Fault-Tolerant Computing (FTCS)*, 1998.
- [26] J. Srinivasan, et al. The impact of technology scaling on lifetime reliability. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*, 2004.
- [27] J. H. Stathis. Reliability limits for the gate insulator in CMOS technology. *IBM Journal of Research and Development*, 2002.
- [28] S. B. K. Vrudhula, D. Blaauw, and S. Sirichotiyakul. Estimation of the likelihood of capacitive coupling noise. In *Proc. of the Design Automation Conference*, 2002.
- [29] N. J. Wang, et al. Characterizing the effects of transient faults on a high-performance processor pipeline. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*, pages 61–70, 2004.
- [30] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*, 2001.
- [31] C. Weaver, et al. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Annual International Symposium on Computer Architecture*, 2004.
- [32] J. F. Ziegler. Terrestrial cosmic rays. *IBM Journal of Research and Development*, 40(1), 1996.