Understanding the Impact of Gate-Level Physical Reliability Effects on Whole Program Execution

Raghuraman Balasubramanian
University of Wisconsin-Madison
ragh@cs.wisc.edu

Karthikeyan Sankaralingam
University of Wisconsin-Madison
karu@cs.wisc.edu

Abstract

This paper introduces a novel end-to-end platform called PERSim that allows FPGA accelerated full-system simulation of complete programs on prototype hardware with detailed fault injection that can capture gate delays and digital logic behavior of arbitrary circuits and provides full coverage. We use PERSim and report on five case studies spanning a diverse spectrum of reliability techniques including wearout prediction/detection (FIRST, Wearmon, TRIX), transient faults, and permanent faults (Sampling-DMR). PERSim provides unprecedented capability to study these techniques quantitatively when applied to a full processor and when running complete programs. These case studies demonstrate PERSim's robustness and flexibility — such a diverse set of techniques can be studied uniformly with common metrics like area overhead, power overhead, and detection latency. PERSim provides many new insights, of which two important ones are: i) We discover an important modeling "hole" — when considering the true logic delay behavior, non-critical paths can directly transition into logic faults, rendering insufficient delay-based detection/prediction mechanisms targeted at critical paths alone. ii) When Sampling-DMR was evaluated in a real system running full applications, detection latency is orders of magnitude lower than previously reported model-based worst-case latency - 107 seconds vs. 0.84 seconds, thus dramatically strengthening Sampling-DMR's effectiveness. The framework is released open source and runs on the Zync platform.

1. Introduction

Hardware reliability is growing in importance and many argue it could become a first-order constraint. To address this, a plethora of work on hardware fault detection [17, 18, 26, 36, 20, 4, 24, 38, 11], prediction [42, 44, 29, 3, 7, 13, 40, 45, 16, 6], recovery [32, 41], and tolerance [12, 23, 34] has been proposed. Understandably, these works adopt various frameworks for quantitative evaluation that fall under two broad categories, namely high-level performance simulators or device-level models. As we elaborate below, both suffer from inadequacies and modeling errors that affect understanding of reliability.

In the high-level performance simulator approach, a fault model is used in a performance simulator. Many of the reliability problems arise from static process variation, dynamic temperature and voltage variation, and dynamic wearout. These phenomena fundamentally change the *delay* of logic gates and consequently the digital logic behavior. Building an approximate coarser-granularity fault-model that captures these effects in a performance simulator introduces large modeling and coverage problems. Typically the notion of gate-level delays is completely absent, while the notion of gate-level logic effects is highly abstracted. Further, even performance simulators have been found ineffective to run entire programs — often-limiting evaluation to a few 100 million instructions. FPGA accelerated simulation inherently can model only logic behavior, and delay simulation and delay modeling is currently not feasible. Overall, in this approach, the workload's effect is captured, but device and microarchitecture accuracy is heavily compromised.

In the device-level approach, detailed device level simulation using SPICE or gate-delay aware simulation is employed. Since, it is time consuming, it is impossible and impractical to conduct an end-to-end evaluation when running full programs. Typically a small and regular sub-circuit like an adder, or a subset of critical paths is considered and studied for a short duration with random inputs. In this approach, while device-level behavior is captured, the workload and architecture/microarchitecture are not modeled.

The important interaction of workload behavior with underlying gate and circuit-level impact of reliability has remained largely under-studied, and our community lacks good tools for such studies. For "conventional" architecture, microarchitecture, and hardware/software co-designed approaches for reliability, this end-to-end capability can complement the aforementioned techniques. For cross-layer approaches that expose reliability concerns up to applications, this end-to-end capability is necessary for meaningful evaluation. *Our paper seeks to address this important facet of quantitative studies by developing a framework that allows the end-to-end investigation of the impact of physical effects of individual gates when running full programs.*

The key idea behind our framework is to monitor with FPGA acceleration any arbitrary circuit of the processor being emulated, while running an actual workload of interest. Four co-designed mechanisms form our framework called PERSim - Physical Effects Reliability Simulator. A mechanism called *input sequence extraction*, extracts a log of all inputs for a gate-level circuit during the execution of a program. These inputs are used to analyze the circuit using *delay aware simulation* and *fault modeling* to cover reliability effects at the gate level. Using a mechanism for *fault injection and deterministic re-execution*, the faults are introduced back in to the system, and their effects studied by running the full programs again using FPGA acceleration. Using an approach with efficient fine-grained communication with an FPGA, delay-aware simulation (not just gate-level logic simulation) and accurate fault modeling, we present a novel co-simulation environment that captures the effects of reliability phenomena on application behavior. The toolchain, along with documentation and tutorials, can be found at `http://research.cs.wisc.edu/vertical/PERSim`.

We make the following contributions.

- A framework with unprecedented fidelity in studying the end-to-end physical effect of reliability at the gate-level running entire programs. Our framework runs the OpenRISC processor and is capable of running full SPEC benchmarks and similar large code bases.
- An FPGA acceleration platform with fine-grained signal observability — enabling high simulation speeds (25 million cycles per second per board) without any limits on signal observability or fault site coverage. Figure 1 shows our evaluation cluster of FPGA boards.
- Demonstrating four fault prediction and detection techniques from the literature using PERSim. These include FIRST [40], Wearmon [45], Online wearout prediction [6], Sampling+DMR [27]. Compared to what has been studied before for these mechanisms, PERSim enables a significant leap in capability: it allows detailed logic and delay simulation and measurement of latencies for arbitrary gates in a full processor running entire benchmarks.
- Evaluations running full applications on arbitrary circuits helped uncover new design insights within each technique (Table 2) and one important general insight: when considering the true logic delay behavior, non-critical paths can directly transition into logic faults, rendering insufficient delay-based wearout detection/prediction mechanisms targeted at critical paths alone.
- The first end-to-end study of transient fault effects on application behavior with device-level charge accumulation-based modeling of particle strikes.

This paper is organized as follows. Section 2 describes the design, Section 3 describes our implementation using the Zynq platform, Section 4 describes the case studies, Section 5 discusses related work and Section 6 concludes.

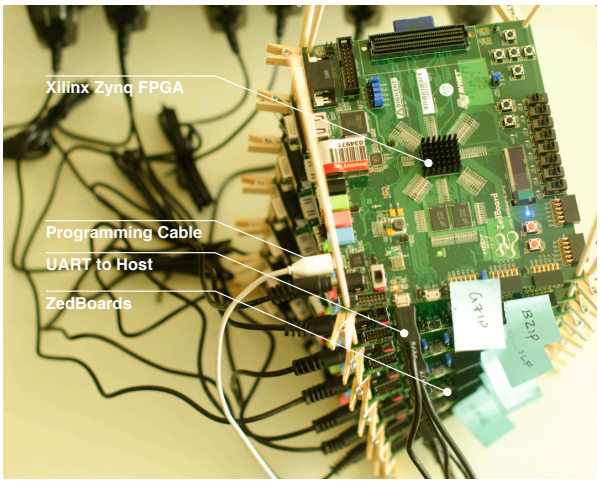


Figure 1. PERSim FPGA cluster

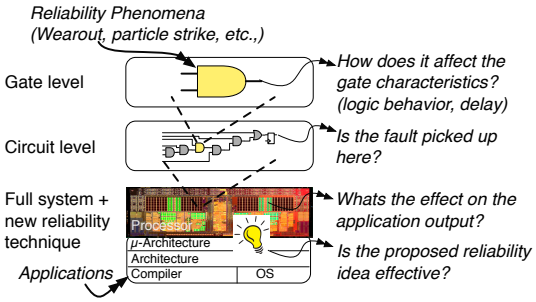


Figure 2. Design goals and rationale

2. Design

In this section we first define our objective and break it down into achievable design goals. Next, we describe how our solution is organized. We then illustrate how this organization satisfies our design goals.

2.1. Objective

This work seeks to answer the question, *"How does reliability physics impact program behavior?"* Figure 2 shows the need for modeling reliability effects at the gate level and analyzing the consequence at the architecture level. Reliability phenomena change the characteristics of individual gates. A faulty gate may adversely effect the logic behavior at the circuit level. This fault may then cause an anomaly in the application. This leads us to our three goals.

2.2. Design Goals

Capture reliability effects at the gate level The physics of reliability is well understood at the gate level and matches the physical phenomenon.

Capture fault propagation to circuit level At the circuit level, gates may be abstracted by defining their logic behavior and their propagation delay. The impact of a particular

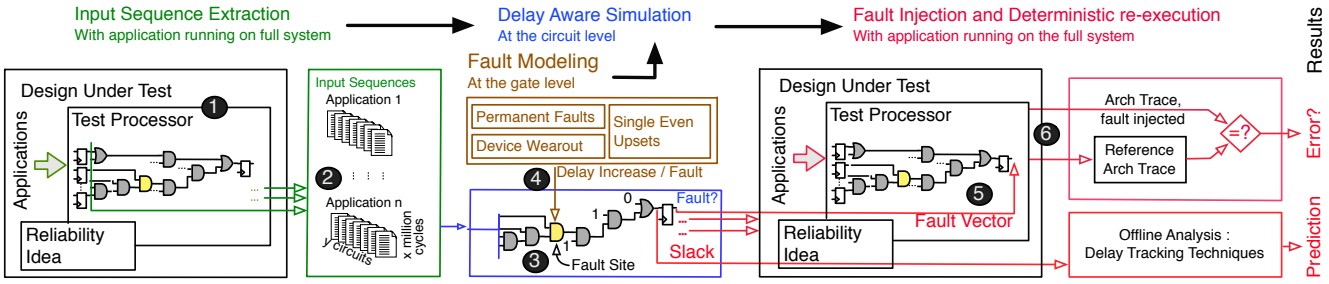


Figure 3. Design overview

gate failure is dependent on the operation of the entire circuit. Hence, it is key to model the state of the circuit around a faulty gate to capture its effect on the microarchitecture.

Check for errors at the full system running entire applications At any time, the utilization of a module in a microprocessor is dependent on the application that is currently executing. Consequently, certain applications or application phases may be unaffected by faults occurring in certain areas in the microprocessor. It is hence critical to evaluate the impact of faults in a full system setting.

In addition to these goals, we seek to create an easily extensible platform to facilitate usage by researchers.

2.3. PERSim Organization

Our goal to check for errors at the full system can be achieved by FPGA acceleration. However, our other goals require gate level modeling and circuit level analysis, and hence require the capability to observe and modify select signals in fine detail when running on the FPGA. Figure 3 shows our overall design organized into the four mechanism. The processor being studied we refer to as the test processor ❶. We start by selecting fault sites in the test processor. Next we isolate the circuit around the sites ❸ and subject them — these are much smaller compared to the full processor — to *delay aware simulation*. *Fault models* capture the effect of reliability phenomena on the logical and delay behavior ❹ of the fault sites. The delay aware simulation is augmented to incorporate these effects. FPGA acceleration is augmented with hooks for *input sequence extraction* to extract sequences from the running benchmarks ❷ to then drive the circuit under delay aware simulation. The output of the delay aware simulation is then used to create a cycle accurate fault vector ❺ that is then injected in to the test processor. In this *fault injection and deterministic re-execution step*, we again use FPGA acceleration to check for the fault's impact on the architecture state and program execution ❻. The next four paragraphs describe the individual mechanisms.

Input Sequence Extraction Detailed delay aware simulation on a full processor is time consuming, so, we simulate only a fraction of it. A fault in a particular gate manifests at the microarchitectural level (on a flip-flop that it drives) only in certain conditions. In particular, characteristics of other gates in its fan-out and fan-in and the logical values driven in a particular cycle affect fault propagation. Hence, we simulate all gates in the fan-in and fan-out of a fault site in delay aware mode. While running an application in an accelerated platform, the Input Sequence Extraction mechanism creates a cycle-by-cycle reference of signal states at the fan-in that will serve as inputs to drive these circuits.

Delay Aware Simulation Using data collected from the input sequence extraction, we run the circuit in detailed delay aware simulation. Reliability phenomena modify the logic behavior and electrical characteristics like the propagation delay of a gate. The properties of the gate identified as the fault site is augmented to include these effects. At every clock cycle, this simulation checks for anomalies that would cause a fault at latches driven by the circuit – like a flipped logic value or an unstable signal at the end of the clock period. The simulation creates a fault vector marking the cycles in which faults appear at the circuit level.

Fault Modeling We utilize available fault models to obtain the effect of phenomena such as *device wearout*, *single event upsets* and *permanent faults* on the logic and delay behavior of individual gates. These models include (i) MOSRA[1] which captures aging effects leading to device wearout and breakdown [43], (ii) a charge accumulation model to capture the effect of a particle strike leading to single event upsets and (iii) probabilistic models that simulate device breakdown seen as permanent faults. Using these gate level models implies that we isolate each gate and obtain its physical properties independent of gates that surround it.

Fault injection and Deterministic re-execution Using the fault vector, we check the effect of the faults on the architectural state and the program. We do this in two steps. First we run the program to completion while saving the architectural state transitions. Next we run the program with a fault injected in the exact execution cycle identified by the delay aware simulation. Using the state transitions obtained earlier as the reference, we mark mismatches as faults.

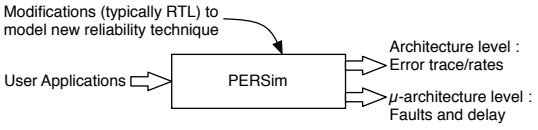


Figure 4. Evaluating a technique using PERSim

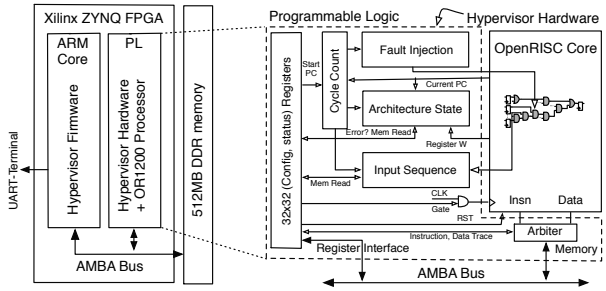


Figure 5. The Zynq FPGA and hardware

2.4. Life of a PERSim Experiment

Figure 4 illustrates a researcher's role in using PERSim and its four mechanisms to implement and evaluate a new reliability technique. The researcher primarily supplies applications and makes minor modifications to the test processor to model the proposed technique. They may choose to augment the fault models if necessary and perform further offline analysis on the faults that cause errors that are the end output of PERSim's fault-injected re-execution.

3. Implementation

This section describes our implementation of PERSim. We introduce the FPGA platform and elaborate on the architecture we implement to accelerate the experiments. We then briefly elaborate on the key implementation features of each mechanism.

3.1. FPGA accelerated system simulation

PERSim is built on the Xilinx Zynq FPGA platfrom. The Zynq FPGA integrates an ARM processor with user programmable logic on a single chip and enables fine grained communication between them. On the FPGA, we integrate an OpenRISC OR1200 processor as our test processor and a firmware-hardware layer we call the PERSim hypervisor.

Xilinx Zynq FPGA The Xilinx Zynq FPGA [1] integrates a fully featured dual core ARM Cortex-A9 processor with a user programmable logic (FPGA) as an SOC. The ARM core and user hardware realized are connected to a shared bus, enabling fine-grained firmware-hardware interaction. The shared bus also serves as the bridge to an external memory. Development boards like the ZedBoard [2] integrate the Zynq FPGA with on-board memory, code download circuitry and a number of peripherals such as USB ports, video and audio ports and GPIO. Both the ARM core and user hardware can access a large external memory through the shared bus.

[1]MOS Reliability Analysis is a plugin to Synopsys HSPICE to model wearout at the transistor level.

Figure 5 shows the modules of our PERSim hypervisor and the OpenRISC processor on a Zynq FPGA. Firmware running on the ARM core communicates with registers in hardware that wraps around the test processor. These registers may be programmed to control the operation of the OpenRISC processor and also to orchestrate collecting signal and architectural state samples and fault injection studies. In the next few paragraphs, we describe the processor under test and the PERSim hypervisor.

The OpenRISC processor The OR1200 core [28] is a 32-bit, in-order, 4-stage implementation of the OpenRISC architecture. The core is written in Verilog RTL and is available open-source along with a firmware toolchain, linux kernel port and a functional simulator. We chose this as the test processor as it is simple to understand and modify and is capable of running full applications — other processors like OpenSPARC and FabScalar [10] may be used.

We made minor modifications to the RTL to expose processor state (such as current PC) that the PERSim hypervisor uses to monitor application progress. Further modifications to extract input sequences and introduce faults are discussed in detail in Section 3.2 and Section 3.5.

PERSim Hypervisor The hypervisor is a firmware-hardware design created to enable fine grained control of the test processor, monitoring application progress, input sequence extraction and fault injection.

The hypervisor hardware is written in Verilog RTL and mapped onto the programmable logic part of the FPGA and is clocked at the same frequency as the processor under test. The hypervisor firmware is written in C and runs on bare metal on the ARM processor. The hardware hooks on to the on-chip shared bus through two interfaces – to connect the registers to the ARM core and to access the memory directly to serve the processor under test. The hypervisor performs a simple memory translation to forward memory requests to an address space reserved for the test processor. The hardware registers are visible to the firmware running on the ARM core as memory mapped registers.

The firmware handles initialization and controls hardware by programming registers. The hypervisor functions are handled by hardware modules that enable the following:

- *Fine grained processor control:* Allows the test processor to be brought back to its reset state, or paused and played by controlling the reset and clock gating.
- *Monitoring processor progress:* By tracking the current PC and the memory requests, the hypervisor ensures that the processor is making active progress. Counters may be configured to count the number of instructions and cycles after execution passes a programmed start PC.
- *Input Sequence Extraction:* Taps at inputs to circuits driving fault sites feed memories in the hypervisor. The values are saved on the fly based on programmed

cycle start and end values. When the memories are full, the hardware clock gates the processor and waits for firmware to read them. These memories can be read through access registers.

- *Fault injection:* The firmware programs the type, location (fault site), and the exact clock cycle in which a fault is to be introduced. The hypervisor provides the following options to emulate a fault – (i) holding the previous value (ii) stuck at 0 (iii) stuck at 1 (iv) bit-flip at the output. The hypervisor handles the fault injection in two steps. First, the program is executed without faults injected, and the hardware saves the architectural state – namely the PC, register writes and stores – to an on-chip memory. Next, in a fault-injected re-execution, the hardware compares the architectural state to the previously stored state and records a mismatch as an error.

The OpenRISC processor enables running full applications while the hypervisor offers cycle-by-cycle visibility into specific circuit regions, and flexible fault injection and error checking capability. This enables the input sequence extraction, and fault injection and deterministic re-execution mechanisms. We next summarize how this platform can be used in conjunction with delay aware simulation and fault modeling to accurately model the effect of a reliability phenomenon that is exposed by a new reliability technique. More details about the implementation and usage may be found at our website `http://www.cs.wisc.edu/vertical/PERSim`.

3.2. Input sequence Extraction

Figure 6(a) illustrates the input sequence extraction setup. At the end of each experiment, for each application, this mechanism creates a file containing the state of the latches that drive each subcircuit for a desired number of clock cycles: Ⓐ in Figure 6(a).

3.3. Delay aware simulation

Figure 6(b) illustrates the delay aware simulation setup. Gate characteristics from a synthesis library are used to simulate circuits in delay aware mode. We augment these characteristics, in particular the delay and logic behavior using the fault models. Driving the inputs sequences, we record a cycle-by-cycle trace of faults and output delays Ⓑ, used by fault modeling to drive fault injection.

3.4. Fault modeling

In PERSim fault models convey the effect of a reliability phenomena by capturing the corresponding change in gate behavior. As these effects vary, PERSim allows for creating and integrating various models. We provide out-of-the-box support for modeling the following:

- **Device Wearout** using Synopsys HSPICE with the MOS Reliability Analysis plugin [43]. This model captures aging due to NBTI and HCI [35, 25].
- **Transient Faults** using a charge accumulation model. The model captures the effect of a particle strike as a temporary logic glitch. The time at which a particle strikes and the charge transferred (consequently the duration of an Single Event Upset) are modeled as uniform and Gaussian random variables respectively.
- **Permanent Faults** using probabilistic models to pick fault sites and the cycle in which a gate breaks down as a stuck-at-0 or stuck-at-1 fault.

3.5. Fault injection and deterministic re-execution

Figure 6(c) illustrates the mechanism used to observe a fault's effect at the architecture level. The hypervisor first creates a clean trace of the architecture state. Next, choosing one fault site and using the information on when it manifests Ⓑ, the program is re-executed. Comparing the traces, we obtain a cycle accurate architectural error trace Ⓒ.

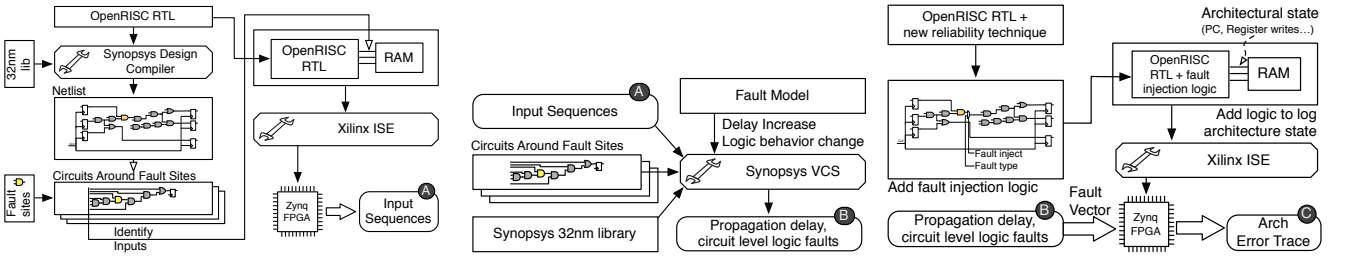
3.6. Implementation Considerations

The **number of fault sites** that can be simulated with PERSim is unlimited. Naive storage of input sequences to Block-RAMs limits the number of flip-flops monitored. In contrast, our design enables saving any number of flip-flops over arbitrary time durations. When the BRAMs are full, the hypervisor hardware automatically clock-gates the processor, freezing it. The firmware then reads the values and removes the clock-gating, allowing input sequence extraction to continue till the BRAMs are full again. This repeats for the desired number of cycles. The tradeoff is emulation speed - while the processor operates at 50MHz, BRAM reads are limited by UART baud rates.

Replacing the test processor in PERSim is straightforward. The hypervisor treats the processor as a gray box. The only modifications to the processor are taps for input sequence extraction, architecture state extraction and fault injection. All complexity arises from mapping the original design to the FPGA. Specific operations like mapping register files to FPGA BRAMs are necessary. In our opinion, the Zynq FPGA (with on-chip memory controllers and the ARM core) reduces this complexity significantly. For example, FabScalar could use the ARM core for system call emulation (typically done by the host).

A **multicore processor** may be used as the test processor. OpenSPARC is one open-source cache-coherent multicore design available and porting it to a new FPGA platform is non-trivial. Further considerations include the limited logic area on the FPGA and avoiding sources of indeterminate re-execution on multicore processors.

Modeling **operating conditions** such as temperature and supply voltage are done during delay aware simulation and SPICE simulations. A unique fault vector is generated for each operating condition. As the Fault Injection mechanism is oblivious to the operating condition, considering many operations conditions translates to more runs (fault vectors), with no additional mechanisms needed.



(a) Input Sequence Extraction (b) Delay Aware Simulation (c)Fault Injection & Deterministic Re-execution

Figure 6. PERSim mechanisms

Technique / *Phenomenon*	**Description**
FIRST [40]	• Periodic hardware signature check
	• Use BIST and test vectors
Wearout	• Reduce clock period, voltage to mimic aging
WearMon [45]	• Periodic DFT check
	• Use existing scan hardware and test vectors
Wearout	• Test at different clock periods
	• Track and analyze failures offline
Online Wearout	• Continuous signal delay monitoring
Prediction [6]	• Delays digitized and sampled
Wearout	• Offline analysis to calculate moving averages
Transient Fault	• Particle strikes cause single event upsets
Analysis	• Delay increase may (not) cause fault to be latched
	• Gate's output may (not) be masked by other gates
Single Event Upsets	• Fault may (not) cause architectural error
Sampling+DMR [27]	• Permanent faults typically cause frequent errors
Permanent Faults	• DMR in short periods (sampling) sufficient to capture these

Table 1. Case studies evaluated on PERSim

Our current fault modeling and fault injection infrastructure does not capture the **interaction between faults**, across several fault sites. Faults that occur in a group of gates and the change in behavior of gates surrounding a faulty gate are not modeled. This is a limitation worthy of consideration and improving in future work.

4. Using PERSim - Case-studies

In this section we pick five case studies and evaluate them using PERSim. Table 1 summarizes the ideas and the reliability phenomenon they handle. The five techniques we picked have never been studied at this level of detail. They are FIRST[40], WearMon[45], TRIX[6], transient fault modeling, and Sampling-DMR[27]. These cover prediction and detection techniques that tackle a range of reliability phenomena including wearout, transient faults and permanent faults. Before elaborating on the case studies, we summarize our key findings. First, PERSim is robust and extensible as it was easily adaptable for these five diverse studies. In each of these cases, we were able to configure PERSim with simple modifications to create a full system evaluation platform. The configuration and usage of PERSim is described in Table 3. We also uncovered some previously unknown limitations in these techniques and obtained key insights into reliability issues as listed in Table 2.

This section is structured as follows. For each case study, we start with a description of the technique, and summarize the experiments done by the authors. We describe how PERSim is adapted to evaluate the idea. We then describe the results and analyze them — *confirming*, *enhancing*, or *contradicting* the results with original work. In each evaluation, we use a mix of the following seven benchmarks from the SPEC2000 integer and floating-point suite: parser, bzip, mcf, mesa, quake, and twolf. Their compiled binaries and details on compilation, input sets etc. is on our website.

4.1. FIRST

Description FIRST is a wearout detection technique that detects marginal failures while introducing marginal operation. Using existing scan and built-in self test (BIST) circuitry, test vectors are pushed and tested at different clock frequencies. In a new chip, as the test frequency is increased, more failures will be seen as the critical paths will show up as timing failures. As the chip ages and the gates slow down, timing faults become more prominent in lower frequencies. The onset of failures is predicted by analyzing the fault rate.

Prior experimental setup and results In their evaluation, Smolens et al.[40] build a proof of concept using the instruction fetch module in the OpenRISC processor. Using a BIST module they inject test vectors and evaluate signatures. By varying the clock period, they show that signature mismatches occur at lower clock frequencies using 1000 BIST vectors. They do not model a fault or project these results as the chip wears-out.

Implementation using PERSim Using SPEC benchmarks, we derive large sequences that we use as test vectors. We model degradation using Synopsys HSPICE with the MOS Reliability Analysis (MOSRA) plugin. Next, we use our delay aware simulation mechanism to check the delay behavior of circuits around ten fault sites under multiple clock periods to simulate marginal operation and for multiple months to capture wearout. We do not use the fault injection mechanism, as this is a fault prediction technique. Instead, we tabulate the number of errors in the output of the delay aware simulation to pick the optimal threshold value — number of errors above which — to make a prediction.

Technique	Original Evaluation	PERSim Evaluation	Key advancement
FIRST	• OpenRISC Instruction fetch module • No wearout modeling	• Full Processor • NBTI + HCI	*Modeling Hole Covered* Gates in non-critical paths not covered. PERSim enables full processor coverage
Wearmon	• FPU module from OpenSPARC	• Full OpenRISC Processor	
Online Wearout Prediction	• ALU from OpenRISC • Random inputs	• Full OpenRISC Processor • Full SPEC2000 benchmarks	
Transient Fault Analysis	• Gate level fault modeling and microarchitecture impact analysis • Microarchitecture level fault modeling application level impact analysis	• Gate level modeling of particle strike and application level impact analysis	*Cross-layer transient fault analysis* Accurate modeling of particle strikes on individual gates ⇒ impact on full programs
Sampling+DMR	• OpenSPARC FPU error traces used to train models	• Full processor error traces used to train models	*Fine-grained signal visibility* Cycle-by-cycle error traces running full programs

Table 2. Key differences in evaluation methodology: State-of-the-art vs PERSim

Technique	Input Sequence Extraction	Delay Aware simulation	Fault Injection and Deterministic re-execution	Fault Modeling
FIRST	• No modifications	• Several clock periods • Several months of wearout	• N/A - Offline analysis. • Errors at different clock periods	• Delay degradation due to wearout • Synopsys HSPICE + MOS Reliability Analysis
WearMon	• No modifications	• Several clock periods • Several months of wearout	• N/A - Offline analysis • Track errors using specific clock period for each signal	• Delay degradation due to wearout • Synopsys HSPICE + MOS Reliability Analysis
Online Wearout Prediction	• No modifications	• Added TDC to convert slack to 5-bit precision number • Several months of wearout	• N/A - Offline analysis. • Exponential Moving Average calculated per signal over time	• Delay degradation due to wearout • Synopsys HSPICE + MOS Reliability Analysis
Transient Fault Analysis	• No modifications	• SEUs modeled as increase in delay • Also check for logic masking	• No modifications	• Uniform probability of when a strike occurs in a clock cycle • Gaussian distributed increase in delay
Sampling+DMR	• No modifications	• N/A - delay aware mode disabled, reports logic masking	• No modifications	• Permanent faults triggered at random fault sites picked at random times

Table 3. Using PERSim to evaluate reliability techniques

Description of PERSim results In our evaluation, we start by recreating Smolens et al., setup — the result is presented in column 1 of Table 4(*confirms prior results*). The operational clock period of our processor is 250ps. Under the wearout we model, the faults manifest after 42 months. In our experiments, we inject five million test vectors and sum the errors from fault sites spread across the full processor. This is repeated at several clock periods — shown in each row in the table. We repeat the experiment at monthly increments of device wearout for four years.

Analysis of PERSim results Table 4 shows that as the processor wears-out, the number of errors increases at all clock periods. Using a period that is 2% less than the operational period and an error threshold of 50000, we may predict an onset of errors four months in advance. This is highlighted in gray in Table 4. This is an empirical result when using OpenRISC implemented using the particular 32 nm library we used. PERSim can be used to evaluate this threshold for a different processor and technology.

We extend prior work by doing a full system analysis while modeling device wearout. From the table we see that lower clock periods may be used to increase the prediction horizon. These serve to detect an emerging wearout fault that would cause a soft fault — or a timing fault at the gates they drive. Delay degradation is also a symptom before hard faults manifest on non-critical paths. As the BIST mechanism exposes delay faults only in critical paths, gates that may directly transition into hard failures are left uncovered.

New Insights Our evaluation of FIRST shows that:

- Using marginal operation introduced by clock period control and signatures at the circuit level helps predict onset of wearout *only* in the critical paths effectively.

4.2. WearMon

Description WearMon works by analyzing errors in scan tests to predict device wearout. WearMon uses periodic testing similar to FIRST, using test vectors and comparing outputs against expected outputs. The tests are repeated with decreasing clock frequencies, and the failures are saved in a multilevel memory and analyzed offline to predict failure.

Prior experimental setup and results As WearMon test may be run while functional units are unutilized, Zandian et al. [45] show that the performance overheads are zero. Unlike FIRST, WearMon works at the path level or the device level, tracking errors specific to each path monitored. Using the OpenSPARC FPU, the authors show that regions of execution exist during which the FPU is unused. These occur frequently and last for a duration when the test vectors can analyze the module without any impact on performance.

Implementation using PERSim In our experiment, we use the input sequence generated by running SPEC bench-

Months	0	1	2	3	4	5	6	7 ...	15	16	17	18 ...	36	37	38	39	40	41	42	43	44
250ps	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2623	5246	7869
245ps	0	0	0	0	0	0	0	0	0	0	0	2623	47214	49837	52460	55083	57706	60329	62952	65575	68198
240ps	0	0	0	0	0	2623	5246	7869	28853	31476	34099	36722	71235	83936	97575	111214	124853	138492	152131	165770	179409
235ps	0	2623	5246	7869	10492	13115	15738	18361	61377	75016	88655	102294	334157	347796	361435	375074	388713	402352	415991	429630	444364
230ps	2623	5246	7869	21508	35147	48786	62425	76064	185176	198815	212454	227188	477666	492400	507134	527244	541978	556712	576822	596932	617042

Table 4. Evaluation of FIRST: Error rates measured as chip wears-out.

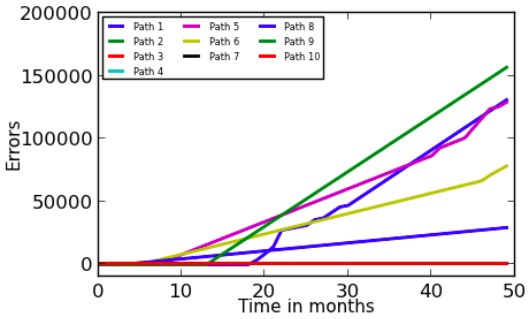


Figure 7. Evaluation of WearMon. Paths 2, 3, 4, 7 and 10 are non-critical and show no errors.

marks as the test vectors. We were unable to generate concise ATPG sequences; however, we believe the test vectors we use are sufficiently large to cover most transitions and activate all paths. We also observe this in our experiment. Using the test sequence to drive circuits, we save the error rates as a function of multiple clock frequencies (test frequencies). This is done by controlling the clock periods in the delay aware simulation and is repeated at various months of chip aging. One key feature of WearMon is to save only the failures and sufficient metadata (such as which frequency errors appear in and what test vector caused it). We model this by saving the error rates along with test frequency and the sequence number that caused the error. For a given gate, it is sufficient to look at one test frequency to predict errors. The experiment is repeated by applying the delay degradation derived from MOSRA to track errors over five years of chip operation.

Description of PERSim results Figure 7 shows the results from our experiments. We pick fault sites that are spread across the processor and have different initial path delays. Checking for faults at the path level enables us to track faults using specific periods for each fault site we track. The figure shows the errors as a function of device wearout on ten fault sites, each tracked using a clock period best suited for that fault site. These error values are continuously analyzed offline to determine an impending failure.

Analysis of PERSim results The clock periods we used to track ranged from the operational clock period to 30% lower. We see that five of the fault sites do not show up as errors before transitioning into hard breakdowns — these sites have path delays significantly lower than the clock period. Clocks with much lower periods must be used to capture the increase in their propagation delays due to wearout.

New Insights We evaluate WearMon on a full processor system. This leads to the following new insights.

- Marginal operation using clock period control and tracking timing faults at specific clock periods enables predicting onset of failures in near-critical paths.
- Using a full system reveals that modules like the decoder do not have periods of under utilization.
- WearMon test may not be applied to modules that hold state in latches without taking the processor offline — a design concept occluded by previous evaluation.

4.3. Online Wearout Prediction

Description Continuously monitoring the degradation in delay is another way to predicting failure. In the technique proposed by Blome et al.[6], the circuit paths in a processor are sampled, and their remaining slack is digitized. This is then averaged using an exponential moving average algorithm (TRIX) over multiple cycles, to annul the effect of the application on delay. By tracking this average over multiple months, we can learn the trend in the degradation, and infer an impending failure.

Summary of prior experimental setup and results Blome et al.[6] pick the ALU from OpenRISC for their evaluation. They model Oxide Breakdown as the reliability phenomenon causing wearout. Using Monte-Carlo simulations on the ALU, they show that using exponential moving averages to track the path delays helps predict an onset of failure. They use this setup to extensively evaluate the sensitivity of weights in calculating the averages. The authors quantify the overheads of using their technique. They observe the correlation between the number of samples considered in calculating the average and the prediction accuracy. They find that the prediction horizon varies across different modules.

Implementation using PERSim Using SPEC benchmarks as applications on our input sequence extraction, we create a 50 million input sequence array. We model gradually manifesting failures such as NBTI and HCI. Degradation in delay is derived with a setup similar to the one used in FIRST. We augment the delay aware simulation mechanism with the sampling and capture circuitry to digitize the slack in the paths. We save this value at the output of specific fault gates over the 50 million cycle execution. As this is a prediction technique, we do not use the fault injection and deterministic re-execution mechanism. Instead, we feed the digitized slacks to a TRIX evaluation block and

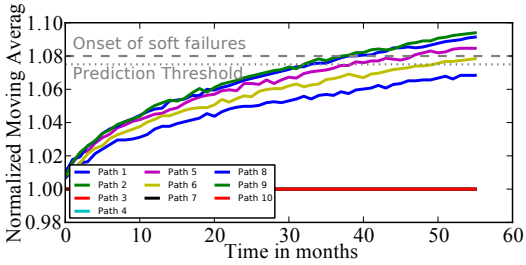


Figure 8. Online wearout prediction results

save the averages over 50 million cycles of execution representing one continuous sample. This experiment is then repeated by increasing the age of the chip by a month per experiment over 5 years.

Description of PERSim results Figure 8 shows the per-path averages plotted as a function of device wearout. We use an 8% increase in the average as the threshold value, above which we predict a device failure. As Blome et al., conclude, the time when a prediction is made is not independent on the circuit tracked. With additional resources, the degradation rate may be combined with the delay averages to arrive a more timely prediction.

Analysis of PERSim results In our experiments, we observe that the program behavior has a significant effect on the averages. For example, a code region that utilizes FPU for a short period of time shows up as an increase in the average delay momentarily. In order to differentiate this effect from actual wearout, it is essential to consider a moving average that includes samplings from a large period of time — 50 million cycles in our experiments.

In their evaluation Blome et al. use Oxide Breakdown — which causes a sharp increase in delay — as the fault model and simulate the ALU of a processor. In our evaluation, we use a model that tracks the degradation due to NBTI and HCI — phenomena that manifest as a gradual increase in delay. As our fault sites were spread across the processor, the inputs and utilization vary widely depending on the program behavior. This helped us observe the influence of the application on the moving average — and the need for a large sampling period.

New Insights

- In addition to wearout, delay averages are influenced by the program behavior. It is important to consider the average over millions of application cycles.
- Continuous, high frequency sampling is necessary. This limits the number of paths that can be monitored. PERSim may be used to pick the paths that are most vulnerable and to pick the coefficients of the averaging algorithm.

The above microarchitecture level techniques rely on observing delay degradation to predict the on-set of failures. Our evaluations reveal a common issue with partial system evaluations — many gates that do not lie on near critical paths do not show up as faults at the microarchitecture level. PERSim covers this *modeling hole* — paths 2, 3, 4, 7 and 10 in Figure 8 are such non critical paths.

4.4. Transient Fault Analysis

Description In this section, we demonstrate how PERSim can be used to check the behavior of an application executing when transient faults occur in the substrate. When a particle strike occurs, the logical behavior of a gate may become unpredictable for a short period of time. This may result in one of the following outcomes.

1. *Delay masking* The period of the fault may be shorter than the clock period and allow safe recovery of the gate before its output is latched on.
2. *Logic masking* A gate may produce a wrong value but be masked by other logic. For example C $\Leftarrow$ A OR B; A transient fault in A is logically masked when B=1.
3. *Architectural masking* A transient fault may cause a wrong value to be latched by a flip flop. In the following cycles, if this sequential cell does not contribute to an architectural state change, it causes no error visible to the application.
4. *Program corruption* A transient fault may result in crashing the application or cause erroneous results.

Summary of prior experimental setup The state-of-the-art in transient fault analysis has been limited to

- Modeling particle strikes at the device level and analysis at the circuit level. An example is work done by Shivakumar et al. [37]. They model a particle strike at the device level as a transient increase in charge accumulated and evaluate the effect at the circuit level. Using a simple circuit spanning a pipeline stage, they classify the faults as logically masked, electrically masked and latch-window masked.
- Using an abstract model in a performance simulator or RTL model [33, 39, 8]. A specific example is research by Saggese et al. [33] that uses FPGAs for acceleration while modeling transient faults as a bit flip in a latch or a gate and reports the impact on the program.

Implementation on PERSim We use a charge accumulation model to capture the effect of a particle strike on a gate. The behavior of the gate is perturbed while charge left behind by a particle strike dissipates. In a given clock cycle, we model the time at which the particle strikes using a uniform probability distribution. The increase in delay before which the gate settles is a function of the charge accumulated and is modeled using a Gaussian distribution centered around the guard band period. We obtain the rate of transient faults that are not delay, logically or architecturally masked. We simulate a stress test with 5 million particle strikes targeting the 10 paths (a total of 72 gates) during the representative region of the application. A particle strike's outcome being non-masked (i.e. causing program corruption or error) is dependent on the individual gates and time

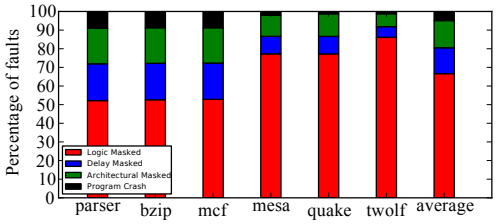


Figure 9. Transient Fault Analysis Results

at which a particle strikes. To estimate a stable or representative error-rate, many particles must be observed. We noticed the error-rate becomes stable when at least 3.5 million strikes are considered.

Description of PERSim results Figure 9 shows the percentage of single event upsets that cause the program corruption and those that are delay, logic, architecture masked.

Analysis of PERSim results From our evaluation, we see that only 5.1% of the single event upsets lead to a change in program behavior. Previous techniques to mitigate transient faults [31] use selective hardening and partial replication. To identify the most vulnerable gates, they rely on path delays, choosing gates that are not delay masked. Our results help identify the fault sites that cause program corruption.

New Insights

- To the best of our knowledge, this is the first end-to-end study of the impact of transient faults on application behavior while modeling the effect of particle strikes as charge accumulations at the device level.
- On average, considering *particle strikes on logic alone*, 94.9% are masked, while 5.1% cause program corruption. Thus reliability techniques for transient faults must consider logic also, and considering only flip-flop and SRAM susceptibility alone is insufficient.

4.5. Sampling+DMR

Description Sampling+DMR is a low overhead permanent fault detection technique. With the application program executing, a checked processor is coupled to a redundant checker processor for a short period of time. Frequently occurring permanent faults are caught immediately; less frequently occurring faults are caught as architectural errors eventually.

Summary of prior experimental setup and results The key metrics to evaluate are the detection latency — number of cycles after which a permanent fault is detected and the number of undetected errors — that escape before an error falls in a sampling window. These variables are dependent on the burstiness of the error occurrences. Nomura et al. [27] use a two-step approach to evaluate these metrics. First they train a 3 stage Hidden Markov Model with error patterns from an OpenSPARC FPU running SPEC benchmark traces. Using this model, they derive the relationship between the detection latency and number of undetected errors as a function of sampling frequency and duration. They

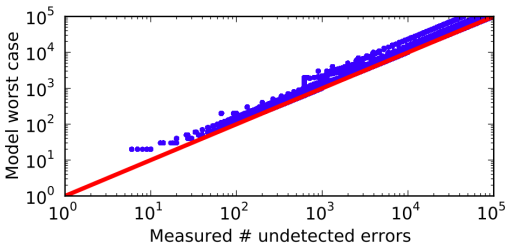


Figure 10. Sampling+DMR : Empirical results vs. Model predicted

Defect Rate	Detection Latency (seconds) @ % DMR			
	Author's Results [27]		Our Results	
	1%	5%	1%	5%
10^{-1}	0.09	0.03	0.002	5×10^{-6}
10^{-2}	2.2	1	0.47	0.22
10^{-4}	72	10	0.80	0.64
10^{-6}	107	21	0.84	0.71
3-stage HMM Coefficients	s0 = 0.042, s1 = 0.478, s2 = 0.48			

Table 5. Sampling+DMR : Detection Latency vs. Defect Rate

validate the models by injecting faults while smaller benchmarks are run on the OpenRISC processor. The models are used to derive the worst case latency and number of undetected errors accurately.

Implementation using PERSim Using input sequence extraction running SPEC benchmarks, we derive circuit inputs for five million cycles. In our experiment, we model permanent-stuck-at-faults only. The faults are introduced in random gates at a random cycle past the initialization run of the benchmark (in hot code). The delay aware simulation mechanism is run with delay information ignored. This helps us capture the logic masking effect, permanent faults show up as bit flips in the latches that they drive only when their outputs are not masked by the other gates. This fault vector marking the exact cycles in which faults manifest is used as the input to the fault injection and deterministic re-execution mechanism. The architectural state is saved for 100000 cycles. Comparing the architecture state with a reference, we measure the error rate and the number of faults detected in the first sample.

Description of PERSim results We introduced stuck-at-faults into 71 gates picked across the processor. In our evaluation, we generate error patterns when permanent faults are introduced in a full processor running applications. Using this pattern, we train the 3-stage HMM. We then use this model to generate a realistic estimate of detection latency.

Analysis of PERSim results Nomura et al. use errors seen on the OpenSPARC FPU to train the models. We enhance model accuracy by training it using real error patterns observed as random gates fail across the processor while executing applications. The error occurrence pattern could itself serve as data to evaluate ideas that currently assume a random occurrence pattern. We repeat the model validation experiment performed by the authors. The results

are shown in Figure 10. The blue dots represent measured errors and the red line shows the bound predicted by the model. As we use a more realistic processor, our data trains the Hidden Markov Model to be more realistic and hence close to observed data. This is also seen in our detection latency measurements — Table 5 shows that in a real system, Sampling+DMR captures errors in significantly fewer cycles than the worst case prediction made by Nomura et al.

New Insights

- We obtained the exact architectural level error occurrence pattern caused by a device level permanent fault and coefficients for the 3-state HMM models.
- Strengthens the case for Sampling-DMR's effectiveness — detection latencies considering full applications is orders of magnitude lower than previously reported worst-case latencies.

5 Related Work

Full system simulation while modeling faults at the device level poses several challenges. FPGA-based emulations like CrashTest [30] provide the much-needed acceleration to run full applications, but fall short in modeling device level impact of reliability phenomena. Low level simulators [39, 8] have incorporated accurate fault models, but are limited by simulation speed. Cross layered simulation platforms like [19] create a fault dictionary using device level modeling that is then introduced in higher level simulations. This fails to capture the influence of program behavior on the fault manifestations.

Of the cross layer simulation platforms, SWAT-Sim [21] is closely related to PERSim. PERSim provides better signal observability, coverage, and simulation speed. SWAT-Sim targets evaluating the impact of permanent faults modeled at the gate level on application behavior. Their co-simulation framework creates a one-to-one mapping from the nets of a netlist version of a processor to variables in a performance simulator. This limits visibility into arbitrary signals. Although a functional simulator can simulate a full system, it is significantly slower than an FPGA accelerated version – limiting the analysis to a few 100 million cycles.

6 Conclusions

Our goal was to create a platform that allows end-to-end investigation of reliability physics on individual gates while running full programs. We developed four mechanisms namely, input sequence extraction, delay aware simulation, fault injection and deterministic re-execution and fault modeling, that work together to achieve this goal. We implemented the input sequence extraction and fault injection and deterministic re-execution using a Xilinx Zynq FPGA. These mechanisms enable running full programs. PERSim's fault modeling mechanism has out-of-the-box support for modeling device wearout, permanent and transient faults. Delay aware simulation enables translating the faults at the device level to the microarchitectural level.

Using PERSim we implemented and evaluated state-of-the-art reliability techniques spanning wearout detection/prediction, permanent fault detection, and transient fault effects. PERSim's unprecedented capability to model physical effect of reliability at the gate-level and propagate its effect up through the application and run an entire application yields new insights on these techniques. A key general finding is that gates in non-critical paths age and may breakdown before critical-path gates and hence wearout detection/prediction targeted at critical-path alone is insufficient. Overall, PERSim appears to be an effective evaluation tool — it helps uncover fundamental design issues that are occluded by performance simulators and non end-to-end simulation approaches. For example, using PERSim in fault injection studies can provide much deeper understanding compared to conventional studies which are restricted to logic faults [9, 39] or utilize performance simulators [15].

Looking forward, reliability techniques are likely to require architecture solutions and there is evidence high-level modeling is insufficient to gain meaningful intuition. Considering a somewhat well understood and narrowly scoped area like transient faults, Cho et al. recently [9] showed that "high-level error injection techniques can be highly inaccurate." On the other hand architecture solutions and even disruptive ones like cross-layer approaches [5, 14, 12, 22] may well become necessary to address reliability. However, insufficient evaluation and modeling errors are likely to severely curtail their adoption and understanding of effectiveness. Going forward, tools and mechanisms like in PERSim are likely essential for meaningfully understanding and evaluating reliability concerns and could become a foundational simulation framework.

7. Acknowledgments

We thank the anonymous reviewers and the Vertical group for providing valuable comments. Support for this research was provided by NSF under the following grant: CNS-1117782.

References

[1] Xilinx zynq fpga. Website. `www.xilinx.com/products/silicon-devices/soc/zynq-7000`.

[2] Zedboard. Website. `www.zedboard.org`.

[3] M. Agarwal, B. Paul, M. Zhang, and S. Mitra. Circuit failure prediction and its application to transistor aging. In *VTS '07*.

[4] T. M. Austin. Diva: A reliable substrate for deep submicron microarchitecture design. In *MICRO '99*.

[5] R. Balasubramanian and K. Sankaralingam. Virtually aged sampling dmr: Unifying circuit failure detection and circuit failure prediction. In *MICRO '13*.

[6] J. Blome, S. Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *MICRO '07*.

[7] K. Bowman, J. Tschanz, C. Wilkerson, S. Lu, T. Karnik, V. De, and S. Borkar. Circuit techniques for dynamic variation tolerance. In *DAC '09*.

[8] H. Cha, E. M. Rudnick, J. H. Patel, R. K. Iyer, and G. S. Choi. A gate-level simulation environment for alpha-particle-induced transient faults. *Computers, IEEE Transactions on*, 1996.

[9] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra. Quantitative evaluation of soft error injection techniques for robust system design. In *DAC '13*.

[10] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg. Fabscalar: Composing synthesizable rtl designs of arbitrary cores within a canonical superscalar template. In *ISCA '11*.

[11] K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco. Software-based online detection of hardware defects mechanisms, architectural support, and evaluation. In *MICRO '07*.

[12] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *ISCA '10*.

[13] Ernst et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO '03*.

[14] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *ISCA '12*.

[15] N. Foutris, D. Gizopoulos, J. Kalamatianos, and V. Sridharan. Assessing the impact of hard faults in performance components of modern microprocessors. In *ICCD*, 2013.

[16] V. Gherman, J. Massas, S. Evain, S. Chevobbe, and Y. Bonhomme. Error prediction based on concurrent self-test and reduced slack time. In *DATE '11*.

[17] D. Gizopoulos, M. Psarakis, S. V. Adve, P. Ramachandran, S. K. S. Hari, D. Sorin, A. Meixner, A. Biswas, and X. Vera. Architectures for online error detection and recovery in multicore processors. In *DATE '11*.

[18] M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz. Transient-fault recovery for chip multiprocessors. In *ISCA '03*.

[19] Z. Kalbarczyk, R. K. Iyer, G. L. Ries, J. U. Patel, M. S. Lee, and Y. Xiao. Hierarchical simulation approach to accurate fault modeling for system dependability evaluation. *Software Engineering, IEEE Transactions on*, 1999.

[20] C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In *DSN '07*.

[21] M.-L. Li, P. Ramachandran, U. R. Karpuzcu, S. K. S. Hari, and S. V. Adve. Accurate microarchitecture-level fault modeling for studying hardware faults. In *HPCA '09*.

[22] X. Li and D. Yeung. Application-level correctness and its impact on fault tolerance. In *HPCA '07*.

[23] X. Li and D. Yeung. Exploiting Soft Computing for Increased Fault Tolerance. In *Workshop on Architectural Support for Gigascale Integration*, June 2006.

[24] A. Meixner, M. E. Bauer, and D. J. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. In *MICRO '07*.

[25] K. Mistry and B. Doyle. How do hot carriers degrade n-channel MOSFETs? *IEEE Circuits and Devices Magazine*, 1995.

[26] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. Detailed design and evaluation of redundant multithreading alternatives. In *ISCA '02*.

[27] S. Nomura, M. D. Sinclair, C.-H. Ho, V. Govindaraju, M. de Kruijf, and K. Sankaralingam. Sampling+dmr: practical and low-overhead permanent fault detection. In *ISCA '11*.

[28] OpenRISC, http://opencores.org/or1k/.

[29] J. Park and J. Abraham. An aging-aware flip-flop design based on accurate, run-time failure prediction. In *VTS '12*.

[30] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, and T. Austin. Crashtest: A fast high-fidelity fpga-based resiliency analysis framework. In *ICCD '08*.

[31] I. Polian, J. P. Hayes, S. M. Reddy, and B. Becker. Modeling and mitigating transient errors in logic circuits. *IEEE Transactions on Dependable and Secure Computing*.

[32] M. Prvulovic, Z. Zhang, and J. Torrellas. Revive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors. In *ISCA '02*.

[33] G. P. Saggese, N. J. Wang, Z. Kalbarczyk, S. J. Patel, and R. K. Iyer. An Experimental Study of Soft Errors in Microprocessors. *IEEE Micro*, 25(6):30–39, 2005.

[34] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *PLDI '11*.

[35] D. Schroder. Negative bias temperature instability: What do we understand? *Microelectronics Reliability*.

[36] E. Schuchman and T. N. Vijaykumar. BlackJack: Hard Error Detection with Redundant Threads on SMT. In *DSN '07*.

[37] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *DSN '02*.

[38] S. Shyam, K. Constantinides, S. Phadke, V. Bertacco, and T. Austin. Ultra low-cost defect protection for microprocessor pipelines. In *ASPLOS '06*.

[39] V. Sieh, O. Tschache, and F. Balbach. Verify: evaluation of reliability using vhdl-models with embedded fault descriptions. In *FTCS '97*.

[40] J. C. Smolens, B. T. Gold, J. C. Hoe, B. Falsafi, and K. Mai. Detecting emerging wearout faults. In *SELSE '07*.

[41] D. J. Sorin, M. M. K. Martin, M. D. Hill, and D. A. Wood. Safetynet: improving the availability of shared memory multiprocessors with global checkpoint/recovery. In *ISCA '02*.

[42] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De. Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In *Symposium on VLSI Circuits*, pages 112–113. IEEE, 2009.

[43] B. Tudor, J. Wang, C. Sun, Z. Chen, Z. Liao, R. Tan, W. Liu, and F. Lee. Mosra: An efficient and versatile mos aging modeling and reliability analysis solution for 45nm and below. In *ICSICT '10*.

[44] X. Wang, D. Tran, S. George, L. Winemberg, N. Ahmed, S. Palosh, A. Dobin, and M. Tehranipoor. Radic: A standard-cell-based sensor for on-chip aging and flip-flop metastability measurements. In *ITC '12*.

[45] B. Zandian, W. Dweik, S. H. Kang, T. Punihaole, and M. Annavaram. Wearmon: Reliability monitoring using adaptive critical path testing. In *DSN '12*.