# An Experimental Study of Soft Errors in Microprocessors

THE ISSUE OF SOFT ERRORS IS AN IMPORTANT EMERGING CONCERN IN THE DESIGN AND IMPLEMENTATION OF FUTURE MICROPROCESSORS. THE AUTHORS EXAMINE THE IMPACT OF SOFT ERRORS ON TWO DIFFERENT MICROARCHITECTURES: A DLX PROCESSOR FOR EMBEDDED APPLICATIONS AND A HIGH-PERFORMANCE ALPHA PROCESSOR. THE RESULTS CONTRAST IMPACT OF SOFT ERRORS ON COMBINATIONAL AND SEQUENTIAL LOGIC, IDENTIFY THE MOST VULNERABLE UNITS, AND ASSESS SOFT ERROR IMPACT ON THE APPLICATION.

Giacinto P. Saggese
Nicholas J. Wang
Zbigniew T. Kalbarczyk
Sanjay J. Patel
Ravishankar K. Iyer
University of Illinois, Urbana-Champaign

The scaling of devices, operating voltages, and design margins for increasing performance and functionality raises concerns about the susceptibility of future-generation systems to soft errors. Many sources contribute to soft errors, including energetic radiation particles, capacitive coupling, electromagnetic interference, and other sources of electrical noise.[1] Historically, such soft errors were mostly of concern when designing high-availability systems typically used in mission- or life-critical applications. However, because of the confluence of device and voltage scaling, and increasing system complexity, experts forecast that transient errors will be a problem in future highly integrated hardware designs. From the incidence of energetic radiation particles on sequential logic alone, it is expected that failures in time for a chip will increase with the number of devices (that is, with Moore's law).[1] Furthermore, recent studies indicate that the soft-error rate attributed to the data paths and combinational logic of processors is increasing.[2]

This article presents an experimental study of the effects of transient errors on two microprocessors: a DLX processor, representative of a microprocessor for an embedded system, and an Alpha processor, representative of a high-performance microprocessor.

The fault model we used is independent of transient-error causes (for example, energetic radiation particles or electrical noise). We conducted fault injection campaigns on gate-level implementations of processors implemented with different design technologies (namely, FPGA for the DLX processor and ASIC for the Alpha processor). This is in contrast with other studies, which introduce a methodology to estimate soft-error rates in an early



Published by the IEEE Computer Society

design stage of high-performance processors,[3] use mathematical models to predict the impact of soft errors in next-generation manufacturing technologies,[2] or estimate soft-error impact via hierarchical simulation starting at a transistor level and going up to the application level.[4]

Here, we use the terms soft and transient error interchangeably. The fault models we used, a bit flip in sequential elements or a transient voltage pulse in combinational elements, are independent of the underlying physical cause.

This article makes several contributions to quantifying the soft-error impact on microprocessors:

- *Fault masking at the processor level.* A significant percentage of injected faults (more than 95 percent for DLX and 85 percent for Alpha) are masked at the processor logic or architectural level; that is, faults do not manifest at the processor pins.
- *Varying fault sensitivity among processor functional blocks.* In the DLX processor, speculation blocks are five times more sensitive to faults (more likely to propagate faults to the processor pins) than processor execution blocks. In the Alpha processor, the flip-flops holding information about queue states and the validity of data in the pipeline are eight times more sensitive to faults than flip-flops representing address fields for memory operations.
- *Combinational and sequential logic (flip-flop) errors.* About 4 percent of faults in combinational logic (for transient voltage pulses of one cycle) and about 10 percent of faults in sequential logic manifest as errors at the processor pins.
- *Application-level impact of faults.* Processor-level faults can cause application crashing, incomplete execution, or fail-silent data violation, or the application can mask them. Only about 53 percent of errors that manifest at the processor pins impact the application for both processors. Nearly 40 percent of fail-silent data violations in the DLX processor are due to errors in the processor execution block. In the Alpha processor, nearly 90 percent of faults injected into the register alias table resulted in fail-silent data violations. In the DLX processor, the register file, the instruction fetch and the dispatcher unit, account for 70 percent of application crashes. In the Alpha processor, 54 percent of fail-silent data violations and application crashes come from the register alias table, the physical register file, and the data paths.
- *Likelihood of multiple bit errors.* About 17 percent of faults in the combinational logic (Alpha processor) lead to double or multiple bit errors in sequential elements.

Due to intrinsic differences in the implementation of the two processors (FPGA versus ASIC), not all results are directly comparable. In this article, we provide a comparison whenever it is meaningful; otherwise we keep the results separated.

## Experimental methodology

We focused our investigations on two processor designs: an FPGA implementation of a DLX-like processor and an ASIC implementation of an Alpha-like processor.[5] The DLX processor occupies 9,526 slices of a Xilinx Virtex E2000-8. The minimum clock period of the synthesized system is about 60 ns. We synthesized the Alpha processor with an IBM 0.18-μm standard-cell library.

### Fault model

This study employs two fault models:

- *Bit flips in sequential elements.* Transient faults (from a variety of causes, including radiation, noise, and power supply glitches) affecting sequential elements—pipeline flip-flops and SRAM cells—can result in an inversion of the state held by the element, through the regenerative feedback of a static flip-flop, an alteration of the charge stored in a dynamic flip-flop, or propagation from a combinational logic node. We model such transients as bit flips by inverting a randomly chosen state element's logical state at a randomly chosen clock cycle. We introduce the state inversion at the clock edge, and it remains until overwritten.
- *Transient voltage pulses* in *combinational elements.* A combinational element in FPGA technology is either a look-up table or a fixed, nonprogrammable gate:

multiplexer, selector logic, XOR, and buffers. A combinational element in an ASIC implementation is an elementary component of the standard library (such as a NAND or NOR gate). To model the effect of a transient error on a combinational gate, we force the netlist node driven by the gate to the inverted value for the entire fault duration. Fault durations span from one-fifth to one clock cycle.

We make the following assumptions:

- The probability of concurrent particle strikes is negligible; that is, we adopt a single-fault model.
- Faults are equally distributed spatially over the circuit components (the sequential and combinational elements).
- Faults are equally distributed in time; that is, uniformly distributed during the simulation time.
- Faults in combinational elements are transients in the output signal of the gates.
- Faults in sequential and combinational elements are transients; that is, the perturbation is temporary, even if its effect can last indefinitely.

### Fault injection framework and application-level outcomes

We obtain the effect of faults injected into the target systems via simulation. For the DLX processor, we used a gate-level description annotated with gate delays representing the system after place-and-route. Conducting extensive fault simulation campaigns of the Alpha processor at gate-level turned out to be practically infeasible. To obtain accurate results for injections into combinational logic in a reasonable amount of time, we selected and synthesized three blocks for fault injection: the decode unit, the architectural-register alias table, and the branch unit. We then simulated their accurate gate-level description with the remaining components described at the register transfer level (RTL). When injecting sequential elements, we used a *latch-accurate* model of the Alpha processor.

In all our fault injection campaigns, we injected faults one at a time and measured their impact on flip-flop state, pins, architectural-state and application level, by comparing the injected system behavior with a golden run. At the architectural level, we considered a fault as an error if its impact was visible for at least one clock cycle. We defined application-level failures as follows:

- *Crash.* A memory location loaded or stored by the processor is out of the boundaries of the application image, or the program execution generates an abnormal exception.
- *Fail-silent data violation* (or *silent data corruption*). The application terminates without crashing, but the memory image or the processor's architectural state contains corrupt data.
- *Incomplete execution.* The program does not complete in the expected time (normal execution time plus a 10 percent margin).
- *No effect.* There is a mismatch at the pin level of the processor (for DLX) or at the architectural level (for the Alpha), but the application program finishes correctly.

## Transient faults in a DLX processor

We examine the effects of transients in the DLX processor in sequential and combinational elements propagating to processor pins, and observing how often this results in a incorrect user visible malfunctioning.

### Logic- and architectural-level masking

We examined the effect of transient faults in sequential and combinational elements, focusing on the propagation to the pins of the DLX processor. Our results are for two applications: bubble sort and a prime number generator. Table 1 summarizes fault injection results for different fault durations. For bubble sort, the error manifestation rate increases from 2.4 to 4.4 percent by varying the fault duration from one-half to one clock cycle. From this data, we conclude that about 96 percent of injected faults never propagated beyond architectural state. We discuss a similar result for the Alpha processor later. Table 1 also provides the average error latency (in terms of clock cycles) defined as the difference between the time of the fault injection and the time when the error manifests at the processor pins.

**Table 1. Manifestation rate at the processor pins as a function of fault duration.**

| Application | Fault duration (cycles) | Manifestation rate (percentage) | Average error latency (cycles) |
|---|---|---|---|
| Bubble sort | 1/2 | 2.4 ± 0.3 | 7.1 |
| | 1 | 4.4 ± 0.2 | 8.4 |
| Prime number generator | 1/2 | 2.1 ± 0.3 | 16.6 |
| | 1 | 3.7 ± 0.3 | 16.7 |

Results for the prime number application show a somewhat lower manifestation rate than for bubble sort (3.7 percent versus 4.4 percent) and higher average fault latency (about 17 versus 7 to 8 clock cycles). The longer fault latency results from the prime number application's use of processor registers to store temporary values and, hence, accessing memory less frequently than the bubble sort. As a result, an error in the prime number application can be latent in the processor for a longer time before manifesting at the processor pins. This observation emphasizes the importance of application characteristics in determining fault sensitivity.

Table 2 indicates that the percentage of faults in combinational logic resulting in errors at processor pins increases with the duration of faults (0.7 percent and 4.2 percent for fault duration of 1/8 and 1 clock cycle, respectively). In comparison, between 3.7 percent and 10.4 percent of faults in sequential logic manifest as errors at the processor pins.[5]

**Table 2. Manifestation rate at the processor pins of faults in combinational logic as a function of fault duration (bubble sort).**

| Fault duration (cycles) | Sensitivity of combinational logic (percentage) | Sensitivity of flip-flop (percentage) |
|---|---|---|
| 1/8 | 0.7 | 3.7 |
| 1/4 | 1.2 | 6.3 |
| 1/2 | 2.2 | 7.3 |
| 1 | 4.2 | 10.4 |

**Table 3. Classification of fault injection experiment outcomes for the bubble sort.**

| Outcome | Error (percentage) |
|---|---|
| Crash | 23 |
| Fail-silent data violations | 13 |
| Incomplete execution | 12 |
| No effect | 53 |

## Application-level masking

Tables 3 and 4 summarize the distribution of manifested errors (observed at the processor pins) among the application failure categories for bubble sort. About 53 percent of errors that propagate outside the processor boundaries do not impact the application's correct behavior. The reason for this is the intrinsic error masking of applications. An error might cause the write of an incorrect value into a memory location. If the subsequent operation overwrites the corrupted location before the application uses the corrupted data, the error is inconsequential for correct behavior. This is consistent with the experimental results on other systems.[4]

Table 4 shows that for all functional units apart from the bus interface (BI), the likelihood of an application masking an error varies between 0.5 percent for the reorder buffer (RB) and 6.1 percent for the load-store (LS) unit. The much higher figure observed for the BI (34 percent) is because of application specifics: The BI controls the usage of the data bus, and consequently errors in the BI often result in accessing data at an incorrect memory location. As long as the application does not attempt to access an illegal location (which would result in a crash), the error can impact the outcome of a single iteration, causing an intermediate data sort to be incorrect. However, due to the nature of bubble sort algorithm, the data is correctly sorted in the next iteration.

Table 4 shows that three units, register file (RF), instruction fetch (IF), and dispatcher (DP), account for 70 percent of the crashes. As expected, faults in the IF and BI are most likely to cause crashes (5 percent sensitivity).

**Table 4. Outcome of fault injection as a function of the injected component (in percentage).**

| | Crash | | Fail-silent data violation | | Incomplete execution | | No error (fault masking) | |
|---|---|---|---|---|---|---|---|---|
| Functional block | Contribution from block (percentage) | Block sensitivity (percentage) | Contribution from block (percentage) | Block sensitivity (percentage) | Contribution from block (percentage) | Block sensitivity (percentage) | Contribution from block (percentage) | Likelihood of fault masking (percentage) |
| Arithmetic logic unit | 4 | 0.4 | 11 | 0.5 | 9 | 0.4 | 8 | 1.7 |
| Branch resolve unit | 1 | 0.8 | 5 | 1.6 | 0 | 0 | 3 | 3.2 |
| Bus interface unit | 9 | 5.1 | 5 | 1.7 | 3 | 0.9 | 25 | 34.2 |
| Commit unit | 0 | 0 | 0 | 0 | 11 | 13.3 | 0 | 0 |
| Dispatcher | 13 | 1.4 | 8 | 0.5 | 9 | 0.5 | 11 | 2.8 |
| Instruction fetch | 16 | 5.4 | 5 | 1 | 23 | 4 | 7 | 5.4 |
| Load-store fetch | 9 | 1.4 | 18 | 1.6 | 11 | 0.9 | 16 | 6.1 |
| Multiply-divide unit | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Register file | 41 | 2.2 | 29 | 0.8 | 9 | 0.2 | 21 | 2.5 |
| Register buffer | 4 | 0.4 | 16 | 0.8 | 26 | 1.2 | 3 | 0.5 |
| Write buffer | 3 | 0.5 | 3 | 0.2 | 0 | 0 | 8 | 2.9 |

**Table 5. Outcome of fault injection as a function of the injected component.**

| Functional block | Crash (percentage) | Fail-silent data violation (percentage) | Incomplete execution (percentage) |
|---|---|---|---|
| Execution | 45 | 40 | 17 |
| Control | 17 | 24 | 35 |
| Speculation | 17 | 10 | 34 |
| Memory interface | 21 | 26 | 14 |

This is because a fault in IF can corrupt the address field of an instruction, and a fault in the BI can force an access to an illegal memory location. Both cases are likely to result in an application crash. About 63 percent of the fail-silent data violations are from three units: RF, LS, and RB. RF and RB hold the application state and, consequently, a fault can lead to fail-silent data corruption. Similarly, a fault in the LS can corrupt a value read from (or written to) memory. Incomplete execution mainly comes from faults in two functional units, RB and IF (which collectively contribute to almost 50 percent of the cases).[6]

Table 5 shows the contribution of the various functional blocks to the different outcome categories. The largest percentage of crashes (45 percent) and fail-silent data violations (40 percent) originate from errors in the execution block. Although the contribution of speculation and control blocks is smaller (the two blocks collectively cause about 34 percent crashes and 34 percent of fail-silent data violations), the percentage is high enough to justify the need for mechanisms to contain those errors. The contributions of these blocks are significant for incomplete executions: 69 percent of these cases are from errors in speculation (34 percent) and control (35 percent) blocks. Since incomplete execution can come from, for example, a stalled IF, a stalled commit unit (CU) or a corrupted RB, detecting such faults in the processor boundaries might require dedicated, processor-level techniques.

## Transient faults in an Alpha processor

In examining the Alpha processor, we took a different approach to logic-level masking, looking at the processor from the perspective of functional units. We did this in the interest of reducing simulation time, selecting functional units that are composed of circuits and structures commonly found in modern microprocessors.

### Logic-level masking

In this section, we describe the results of fault injection into synthesized components of the Alpha processor. We isolated and synthesized three modules of the Alpha processor: the decode stage of the pipeline (decoder), the branch execution unit (branch), and the architectural-register alias table (RAT). The decoder is predominately random logic. Branch consists of an adder and some logic for evaluating branch conditions. RAT is a

table of architectural-to-physical-register mappings plus a small state machine to facilitate mapping recovery. We chose these three modules because they contain circuitry commonly used as building blocks in processors.

We used the bzip2 application from the SPEC2000 benchmark suite as the workload. Figure 1 plots a summary of the experimental results. The figure shows the manifestation rates at the interface of each injected block, for the various combinations of blocks and fault injection durations. The manifestation rates increase approximately linearly for all three blocks with increases in fault duration. This linear increase is what you would intuitively expect from latching window masking.

The fault injection data were also used to analyze the likelihood of multiple bit errors from a single injected fault. Figure 2 presents this data for the branch block. The decoder exhibits similar behavior, while for the RAT block a large portion of faults result in multiple bit errors.[5]

Single bit-flip errors in architectural state dominate in the decoder and branch blocks. On average, they represent 80 percent of all error manifestations. The larger proportion of multiple bit-flips in the RAT block is attributable to fault injections that affect write-enable signals, typically resulting in the corruption of numerous bits in parallel.

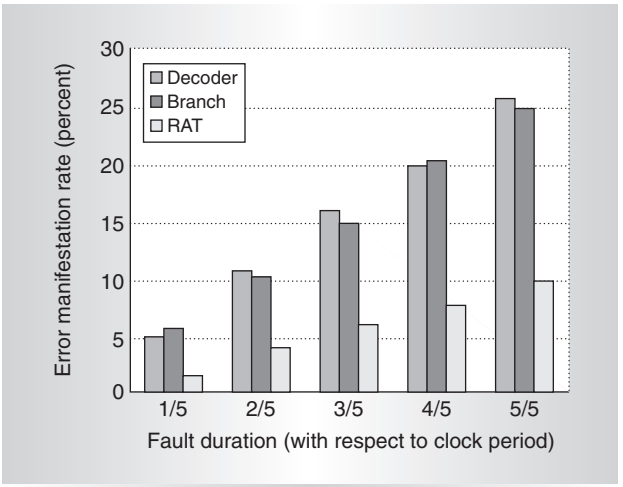


Figure 1. Manifestation rate at the interface of each block for different fault durations.

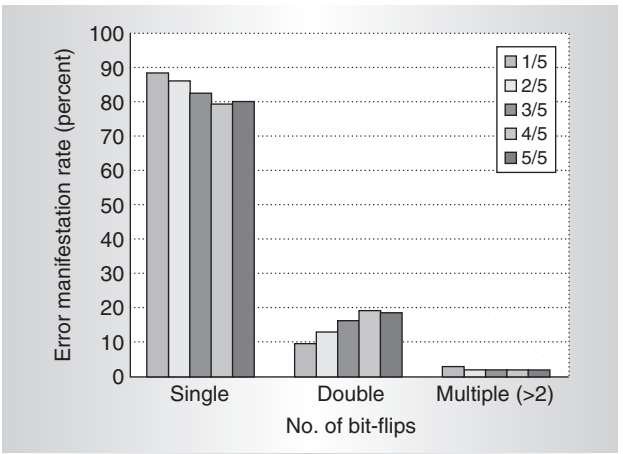


Figure 2. Faults resulting in multiple bit-flops.

## Architectural-level masking

In this section, we discuss the results of fault injection into the processor's microarchitectural state to determine the fault's likelihood of propagating into architectural state or causing pipeline deadlock. The faults injected are bit flips at the latch level of an RTL description. We categorized each flip-flop and RAM cell in the processor based on the general function provided by that bit. For example, we placed flip-flops and RAM cells that hold instruction input and output operands into a data category.[7]

We categorized the results of the fault injection campaign, separating them by the logic block of the injected state bit and the injection's resulting outcome. Figure 3 presents these results. One can observe that the RAT block and the physical register file (regfile) are especially vulnerable to soft errors, since nearly 90 and 40 percent of the faults result in an application failure. This is not surprising since these structures contain the software-visible register file.

We observe that 85 percent of fault injections are masked; this is a significant result, particularly considering that we are only injecting approximately 50 to 55 percent of the surface area of a modern processor die (as estimated from die photos of the Alpha 21264 and the Pentium 4). The uninjected portions include the cache RAM arrays and predictor structures, which we can harden through redundant coding or never contribute to failures. We also observed that the masking levels for flip-flops are higher than that of RAM arrays, indicating that data held in flip-flops are generally less utilized. Recall that about 95

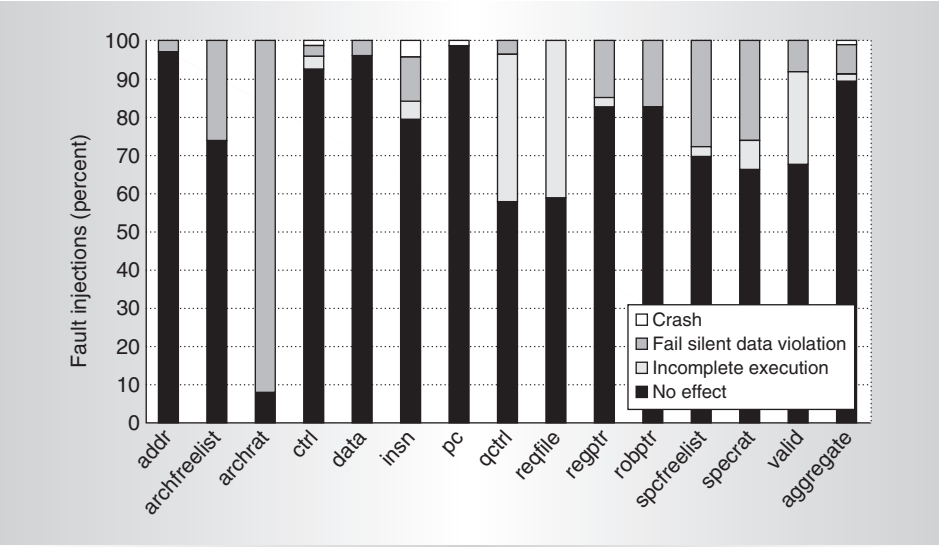


Figure 3. Results of fault injection into flip-flops by type.

percent of faults injected in the DLX were masked at the architectural level.

### Application-level masking

Soft errors that are not masked in the microarchitectural level propagate to the architectural level and become visible to the application. We analyzed the application-level impact of faults that have propagated to the architectural level. To inject a fault, we selected an instruction from the dynamic instruction stream at random and forced it to execute incorrectly (that is, directly flipping a bit from this instruction's result, or flipping a bit in the opcode). We then allowed the program to proceed and monitored the simulation to capture the error impact at the application level in one of four outcomes: crash, fail-silent data violation, incomplete execution, and no effect. Across all the injection campaigns, approximately half of the experiments had no effect.[5] This masking effect arises largely from dead and transitively dead values in the instruction stream.[7]

## Trends

At the architectural level, we observed how often soft errors in combinational logic propagate and cause application failure. A single-cycle soft error in the combinational logic can originate in the instruction decode stage, propagate, and cause application failure more often in the DLX processor (4.2 percent of the time on DLX and 1.5 percent on Alpha). In the instruction execution unit, the corresponding failure rates were 3 percent on DLX and less than 1 percent on Alpha. Much of this difference is likely due to an increased amount of incorrect speculation on the Alpha processor. The Alpha's wider and deeper pipeline leads to a larger portion of unutilized processor resources, for example, due to temporally limited instruction-level parallelism.

We also compared the rates at which soft errors that had already affected architectural state induce application-level failure. Both the DLX and Alpha processors exhibited similar application-level masking rates: approximately half of all injected faults have no effect, a quarter result in program crash, and the other quarter result in fail-silent data violation.

## Solutions: Low-hanging fruit

In order to guide the design and the placement of mechanisms for protecting against soft errors, we estimated the minimum percentage of processor state that we must harden against soft errors to reach a given error coverage goal. In essence, this is identifying the low-hanging fruit of the soft-error problem. This estimate is important in making design decisions if design budgets only allow soft-error protection for a fraction of a design, perhaps via replication, error correction code, radiation-hardened standard cells, or error-trapping latches.[8]

Figure 4 presents this data. The *x*-axis represents the percentage of protected processor state, while the *y*-axis represents the obtained error coverage. The three lines each represent either the processor's entire state (Alpha-ffs+RAMs) or only the flip-flops (Alpha-ffs or DLX-ffs). The prefix of the data label denotes the processor associated with the data.

Before discussing the data, we briefly introduce the methodology we used to generate the data. Ideally, we would evaluate each state element of each processor individually to identify how often each element would lead to processor failure if upset. Unfortunately, evaluating each flip-flop or SRAM cell in a microprocessor design is computationally prohibitive. To address this issue, we divided the processor's state into discrete sets of sequential elements and then evaluated the soft-error sensitivity of each set. We divided the DLX processor into 11 functional units (such as bus interface unit, arithmetic logic unit) and the Alpha processor into 15 sets of state with similar logical function (for example flip-flops devoted to register file pointers or those assigned to data paths).[5] The data points in Figure 4 represent the coverage improvement that comes from incrementally protecting additional sets of sequential elements from soft errors.

In doing so, we average together the soft-error vulnerabilities of each flip-flop or SRAM cell in each set. Consequently, the curves in Figure 4 represent a lower bound on the error coverage obtainable for a given percentage of protected processor state (that is, the true, ideal curve begins steeper before eventually flattening out).

We see that protecting a relatively small fraction of the Alpha processor's state can cover a large fraction of failures due to soft-error. At the knee of the Alpha-ffs+RAMs curve, we note that protecting only 30 percent of processor state covers nearly 80 percent of failures, corresponding to a 5× improvement in mean time between failures.

The DLX processor exhibits a less promising curve. Protecting 30 percent of the processor state (aggregated in terms of functional units) yields a 50 percent reduction in soft-error-induced failures. We can attribute part of this difference to using slightly fewer sets of sequential elements on the DLX processor. Another contributing factor is the categorization of the DLX processor on a higher level of abstraction when compared to Alpha (sets based on functional unit in the DLX versus logical functions in the Alpha).

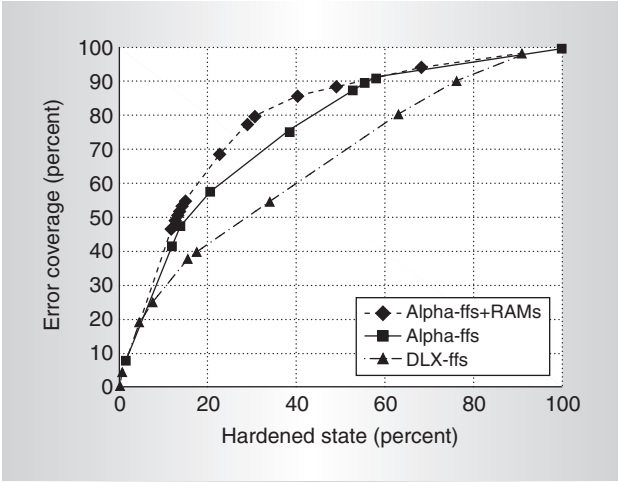


Figure 4. Error coverage versus percentage of protected state.

The Alpha-ffs and DLX-ffs curves are particularly interesting if the processor's SRAM state is easily hardened against soft errors via error correction code. Then the key contributor to soft-error-related failures would likely stem from soft errors in flip-flops. In summary, we see that a relatively small amount of error protection can go a long way in mitigating the effects of soft errors on processors.

This work presents an experimental study of the effect of soft errors on microprocessors. We tracked the rates and ways in which the effects of soft errors propagate through the various abstraction layers, eventually leading to application failure. In particular, we performed case studies on two processors, examined different fault models, and analyzed the effects of soft errors at the logic, architecture, and application levels. From the studies on the two processors, we extracted two high-level trends. First, we observed that across both processors, once a soft-error event propagates into software, the software itself masks about half of the faults from propagating into failures. Second, the rate at which soft errors propagate through the processor architecture and into a software-level failure differs significantly. On the DLX processor, this rate was approximately 4.2 per-

cent while the corresponding figure on our Alpha processor was less than 2 percent. The difference is attributable to the increased speculation and relative inefficiency of a deeply pipelined superscalar processor. Finally, we evaluated the amount of error coverage obtainable by protecting a fraction of a processor design from soft errors. We found that a small, strategically placed amount of error protection can provide very good fault tolerance. MICRO

## Acknowledgments

We thank Fran Baker for her careful reading of an earlier draft of this paper and Anoop Vetteth for his contribution in carrying out the experiments. This work was supported in part by NSF grant ACI 0121658 ITR/AP, MURI grant N00014-01-1-0576, Gigascale Systems Research Center (GSRC/Marco), the Center for Circuits and Systems Solutions (C2S2/Marco), and the team of students who helped construct the Alpha RTL model.

References

1. T. Karnik, P. Hazucha, and J. Patel, "Characterization of Soft Errors Caused by Single Event Upsets in (CMOS) Processes," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 2, Apr. 2004, pp. 128-143.
2. P. Shivakumar et al., "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," *Proc. Int'l Conf. Dependable Systems and Networks* (DSN 2002), IEEE CS Press, 2002, pp. 389-398.
3. S. Mukherjee et al., "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," *Proc. 36th Int'l Symp. Microarchitecture* (Micro-36), IEEE CS Press, 2003, pp. 29-40.
4. Kalbarczyk et al., "Hierarchical Simulation Approach to Accurate Fault Modeling for System Dependability Evaluation," *IEEE Trans. Software Eng.*, IEEE CS Press, vol. 24, no. 5, Sep.-Oct. 1999, pp. 619-632.
5. G.P. Saggese et al., *An Experimental Study of Transient Faults in Microprocessors*, tech. report, Center for Reliable and High-Performance Computing, Univ. of Illinois, Urbana-Champaign, 2005.
6. G.P. Saggese et al., "Microprocessor Sensitivity to Failures: Control vs. Execution and Combinational vs. Sequential Logic," *Proc. Int'l Conf. Dependable Systems and Networks* (DSN 2005), IEEE CS Press, 2005, pp. 760-769.
7. N. Wang et al., "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," *Proc. Int'l Conf. Dependable Systems and Networks* (DSN 2004), IEEE CS Press, 2004, pp. 61-72.
8. S. Mitra et al., "Robust System Design with Built-In Soft-Error Resilience," *Computer*, vol. 38, no. 2, Feb. 2005, pp. 43-52.

**Giacinto P. Saggese** is a postdoctoral research associate at the Center for Reliable and High-Performance Computing, University of Illinois, Urbana-Champaign. His research interests include system dependability and security, computer architecture, and the design and evaluation of high-performance systems targeting FPGAs and ASICs. Saggese has a PhD in electrical and computer engineering from the University of Naples, Italy. He is a member of the IEEE.

**Nicholas J. Wang** is a PhD candidate in the electrical and computer engineering department at the University of Illinois, Urbana-Champaign. His research interests include fault-tolerant computer architectures. Wang has an MS and a BS in electrical and computer engineering from the University of Illinois, Urbana-Champaign. He is a student member of the IEEE.

**Zbigniew T. Kalbarczyk** is a research professor in the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. His research interests include automated design, implementation, and evaluation of dependable and secure computing systems. Kalbarczyk has a PhD in computer science from the Bulgarian Academy of Sciences. He is a member of the IEEE and the IEEE Computer Society.

**Sanjay J. Patel** is an associate professor of electrical and computer engineering, and the Willett Faculty Scholar at the University of Illinois, Urbana-Champaign. He is also chief architect at Ageia Technologies. His research interests include processor microarchitecture, computer architecture, and high-performance and reliable computer systems. Patel has a PhD, an

MS, and a BS in computer science and engineering, all from the University of Michigan, Ann Arbor. He is a member of the IEEE.

**Ravishankar K. Iyer** is the director of the Coordinated Science Laboratory and the George and Ann Fisher Distinguished Professor of Engineering at the University of Illinois, Urbana-Champaign. His research interests include reliable and secure computing. Iyer has a PhD in electrical engineering from University of Queensland, Australia. He is an IEEE and ACM Fellow, and an Associate Fellow of the American Institute for Aeronautics and Astronautics.

Direct questions and comments about this article to Giacinto Saggese, 260 CSL, 1308 West Main, Urbana, IL 61801; saggese@uiuc.edu.