A Microarchitectural Analysis of Soft Error Propagation in a Production-Level Embedded Microprocessor

Jason Blome[1], Scott Mahlke[1], Daryl Bradley[2] and Krisztián Flautner[2]

[1]Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{jblome, mahlke}@umich.edu

[2]ARM, Ltd.
Cambridge, United Kingdom
{daryl.bradley, krisztian.flautner}@arm.com

ABSTRACT

Current trends in device scaling continue to cause an increasing risk of transient faults in microprocessors due to high energy strikes from radiated particles. In this work, we present a thorough microarchitectural analysis of the effects of soft errors on a production-level Verilog implementation of an ARM926EJ-S core. We examine the propagation of faults occurring in both sequential state elements and combinatorial logic and note a number of critical distinctions in the error propagation behavior of soft errors occurring at logic gates versus state elements. Further, we exemplify the ways in which the emerging trend of faults in combinatorial logic will affect the scope of the soft error problem, especially in the embedded design space. Also, since this work was conducted on a production-level core, we highlight some of the nuances of soft error effects that arise and are specific to production-level designs.

1. INTRODUCTION

Device scaling trends towards reducing feature size, increasing integration, and lowering voltage levels increase the soft error rate of microprocessors by both lowering the minimum amount of charge necessary to cause a bit flip and increasing the number of susceptible targets for potential particle strikes to cause errors. These trends have made reliability an increasingly important design constraint in a variety of different microprocessor markets.

Though strict reliability constraints have typically been applied almost exclusively in aerospace and high-end server markets, dramatically increasing demand for embedded microprocessors in a variety of emerging areas, such as the automotive and health care industries, have generated interest in highly reliable embedded designs as well.

The standard mechanism for reporting device reliability is the number of failures in time, or the FIT rate, where a rate of 1 FIT means that the mean time before an error occurs is one billion device hours. As an example of the increasing need for reliability in embedded devices, take the case of expanding integration in the automotive industry. According to the Federal Highway Administration, there were 231 million registered vehicles in the United States in 2003. If each automobile were equipped with anti-lock brakes systems (ABS) controlled by an embedded microprocessor with a FIT rate of 114 FITs or about one error per 1,000 device years, approximately 26.34 cars would experience errors within their ABS systems each hour. Since not all registered vehicles are in operation at any given time, and current ABS systems are equipped with mechanical back ups, this problem is not nearly so dramatic, but nevertheless provides an example of increasing need for reliable embedded devices.

One of the key features differentiating the embedded from the high performance design space is longer clock cycle times. This longer cycle time in embedded designs typically leads to larger logic depths between sequential state elements. The effects of these larger logic depths are two-fold. First, large logic depths increase the relative area of the chip consumed by combinatorial logic, making combinatorial logic much more susceptible to particle strikes. For example, combinatorial logic consumed 58% of the total cell area (excluding caches) of ARM926EJ-S core used in this work. Second, larger logic depths typically imply a wider signal fanout, thus increasing the number of potential target latches which may store and possibly propagate an incorrect value caused by a single soft error.

Recent studies have shown that the frequency of soft errors in combinatorial logic gates is approaching that observed in SRAM cells [8]. Previous work has largely focused on the effects of soft errors occurring in state elements [3, 9]. In addition, some studies of sensitivity analysis and error propagation paths between logic blocks have been conducted for soft errors occurring in combinatorial logic [6]. In this work, we focus on a thorough, low-level microarchitectural analysis of the effects and propagation behavior of soft errors occurring in both combinational logic and sequential state elements. In particular, we expose a number of key differences between the soft error propagation behavior of faults occurring in sequential state elements and combinatorial logic at the microarchitectural level. We analyze these effects on a production-level ARM926EJ-S embedded microprocessor. The major contributions of this work are as follows:

A design-independent soft error injection and analysis framework. We present a framework which can be used to study the architectural and microarchitectural effects of soft errors occurring in both sequential state elements and combinatorial logic gates. This framework is responsible for reporting detailed error propagation information as errors propagate through *all* microarchitectural state in the design over an arbitrary number of cycles subsequent to fault injection.

An empirical derivation of the logical and temporal soft error masking rates for a commercial embedded microprocessor. In this work we, conduct a number of experiments in order to derive both an application-independent average logical masking rate as well as the logical and temporal masking

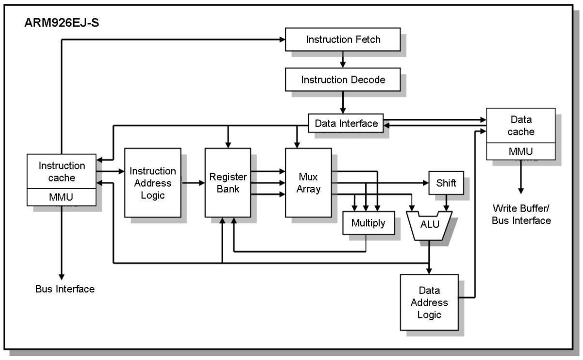

Figure 1: A high-level block diagram of the ARM926EJ-S core

rates observed when executing a standard image processing algorithm.

An analysis of the error fanout and propagation behaviors of soft errors in a commercial embedded microprocessor. We present a detailed analysis of the number of architectural and microarchitectural errors caused by soft errors over time and examine how they propagate throughout the design. Further, we identify a number of critical distinctions between the propagation behavior of soft errors occurring in combinatorial logic and state elements and describe how these distinctions affect the scope of the soft error problem.

The remainder of this work proceeds as follows. Section 2 describes an overview of the embedded microprocessor design that was used in our experiments as well as the fault injection and error propagation analysis framework. Section 3 demonstrates the results of fault injection experiments in both combinatorial logic and in sequential state elements. Section 4 discusses some of the implications of the results as well as some interesting nuances of soft error effects when analyzed on a commercial design. Section 5 presents some of the prior work that has been done is this area and finally, Section 6 concludes this discussion.

2. FAULT ANALYSIS FRAMEWORK

The experiments described in this work were conducted using a Verilog model of an ARM926EJ-S microprocessor [1]. The ARM926EJ-S is a 32-bit embedded architecture and has a standard five stage pipeline consisting of fetch, decode, execute, memory and write-back stages. The core datapath of this ARM core is depicted in Figure 1. The implementation used in this work has 37 architecturally defined registers (31 32-bit general purpose and six status registers), 4 KB of instruction cache and 4 KB of data cache. The Verilog model was synthesized with scan-chain insertion and design-for-test methodologies in an Artisan library characterized for a 130 nm process using the Synopsys Physical Compiler. The synthesized netlist and a hand-designed floorplan were processed in Avanti Astro for clock tree synthesis and physical placement in order to ensure that a five nanosecond clock cycle time was met. Once fully synthesized with all design rule constraints satisfied, timing information was extracted in Standard Delay Format so that it could be annotated on to the netlist and simulated using Synopsys VCS.

The testbench used for simulation of the ARM926EJ-S instantiates a pair of the synthesized netlists: a reference design and the unit under test. Both are annotated with the timing information gathered from the synthesis and layout tools. The testbench also includes a behavioral memory model that is used to load benchmarks at simulation initialization. An overview of the soft error test harness is shown if Figure 2

The soft error injection and analysis framework used in our experiments is composed of a set of Verilog Programming Interface (VPI) libraries which are invoked at the start of simulation. Upon invocation, the framework probes the design in order to derive the set of all sequential state elements and combinatorial logic gates within the unit under test. Depending on the simulation parameters, once the fault injection framework is initialized, it may schedule fault injection experiments at arbitrary points in time for arbitrary durations, selecting a random design element (register or logic gate) as a fault injection target and inverting the value at the node's output.

Experiments in this work are divided into *application-based analysis* and *random-state analysis*. Each experiment is conducted targeting both sequential state elements and combinatorial logic gates. Application-based analysis is carried out by running benchmark code loaded into the behavioral memory model at simulation initialization. In this case, the framework will select a random point in time between 2,500 and 5,000 cycles after the start of simulation to conduct its first fault injection. If the experiment being conducted is intended to include temporal masking analysis, the fault injection time is randomly selected in picoseconds, and the fault duration is randomly selected on the interval $[0.25 * CLK, CLK]$ where CLK is the clock cycle time of 5 ns that this design was synthesized to meet. Otherwise, the fault injection time is scheduled at some future rising edge of the clock signal and will be held for the duration of one clock cycle.

When random-state analysis is conducted, the framework is used to drive the experiments by setting the microprocessor to a randomly generated microarchitectural state, injecting a fault, observing the effects of the fault in the subsequent cycle, and repeating. The random-state based experiments are meant to derive an application-independent measure of logical masking within the microprocessor core. In this case, the observation of soft error effects is limited to only the subsequent cycle because the machine is not guaranteed to have been in a valid state at fault injection time.

At fault injection time, depending on the type of injection experiment being simulated (soft errors in combinatorial logic, soft errors in sequential state, or both), a random design element is selected for fault injection from the unit under test. If the fault is to be injected into a logic element, a random combinatorial gate in the design is selected and the value present on it's output is inverted, simulating an upset caused by a particle strike. Similarly, when faults in registers are being simulated, a random register is selected and its output is inverted. When a fault is injected into the design, the framework logs the fault site, the time of injection, and the pulse duration (if the pulse occurred at a logic element).

After a fault has been injected into the system, at each subsequent rising clock edge *every* microarchitectural register in the unit under test is compared against its dual

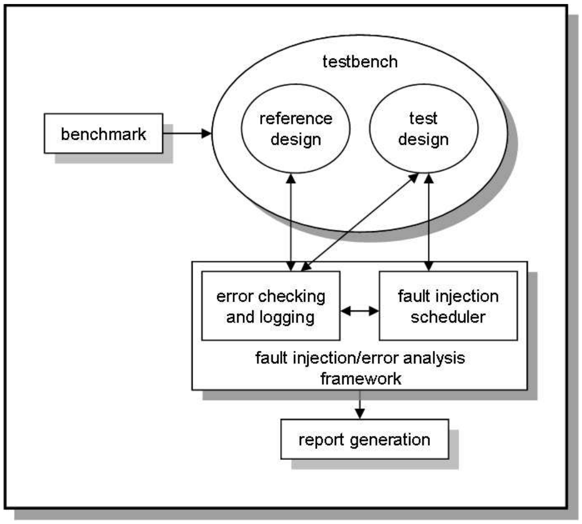
testbench
benchmark
reference design
test design
error checking and logging
fault injection scheduler
fault injection/error analysis framework
report generation

Figure 2: An overview of the soft error injection and analysis framework

in the reference design. Further, all top-level output ports on the design (I/O buses, coprocessor interface, test equipment) and inputs into the caches are checked to ensure that no corrupt values have escaped from the core datapath. If, in the first cycle after fault injection, no register, cache, or top-level port mismatches occur, the injected fault did not affect the system, and so a new random time in the future is selected for another fault injection experiment. If any register, cache, or port mismatches do occur, the fault analysis framework logs the relative cycle and site of the error for later analysis.

The fault analysis framework then continues to track the progress of errors throughout the system for an arbitrary number of cycles after the fault injection. If during this period, no errors are present, and no errors have propagated out to the caches or top-level ports, the system is clean, and the fault was successfully masked, allowing a new random time for fault injection to be scheduled. If top-level port or cache errors did occur, or a latent error still lingers in the design which has not yet affected architectural state, then simulation halts, and error logs are written for post-processing to analyze propagation behavior and architectural state effects. Though latent errors in the design may not have caused errors in software-visible state, they still pose a threat and may potentially cause a data corruption. A moderate amount of microarchitectural state in our microprocessor core consists of mode-specific state elements that are not always exercised by our limited benchmark simulations, but given a more realistic workload could potentially propagate errors throughout the system.

3. EXPERIMENTAL RESULTS

Though a particle strike may cause a transient pulse at any node within a circuit, in order for the pulse to become an error, it must be latched at some point. Transient pulses that are not latched derate the soft error rate (SER) of the microprocessor and must be accounted in order to understand the effects of soft errors and the overall SER. The three types of circuit-level soft error derating factors are as follows:

Logical masking: Logical masking occurs when a transient pulse is effectively gated from all possible destination state elements; for example, a transient pulse at the output of a circuit which is ANDed with 0 will always be logically masked.

Temporal masking: Temporal masking (or latching-window masking) occurs when a transient pulse propagates to a state element, but does not arrive within the capture window of the state element.

Electrical masking: Electrical masking occurs when a transient pulse is attenuated by subsequent logic gates such that the pulse does not affect the output of the circuit.

The experiments presented in this work examine the effects of both logical and temporal masking on the overall soft error rate, but leave electrical masking for future work. Further, we conduct an analysis of the architectural and microarchitectural activity that occurs as a result of transient pulses occurring at both sequential and combinatorial logic elements. The remainder of this section describes the following experiments:

Average Logical Masking Rate: In this experiment, we quantify the average, application-independent amount of logical masking that takes places within the ARM926EJ-S core by setting the microarchitectural state to a random configuration and simulating a transient pulse for the duration of one cycle.

Workload-Specific Logical Masking Rate: Here, we derive the logical masking rate within the microprocessor when running a standard image processing algorithm.

Workload-Specific Logical and Temporal Masking Rate: Here, we analyze the effect that temporal masking has on the overall microarchitectural and architectural masking rates within the design by injecting transient pulses for random durations at random points in simulation time.

Microarchitectural Error Propagation Behavior: In this experiment, we examine the propagation of errors through architectural and microarchitectural state over time and examine some of the key differences between faults occurring in logic and in registers.

3.1 Average Logical Masking Rate

In this experiment, we derive the application-independent average logical masking rate for the ARM926EJ-S. At the start of simulation, the microarchitectural state of the core is set to a random configuration. Next, at the rising edge of the clock, a transient pulse is injected at a random node for the duration one clock cycle. Finally, the fault's effect over the course of the subsequent cycle is observed. In this work, we ran 100,000 fault injection experiments for soft errors simulated in both logic and registers. Since the randomly generated machine state may be undefined or illegal, the effects of fault injection are only tracked for one cycle because subsequent cycles will likely lead to a reset state or potentially undefined behavior.

The average masking rates for microarchitectural state,

Error Location	Masking Rate	Incorrect Bits
Microarchitectural state	30.04%	3.76
Architectural state	92.88%	1.03
Top-level port	97.49%	1.46

Table 1: Average logical masking rates and number of bits corrupted for soft errors occurring in registers for random machine state

Error Location	Masking Rate	Incorrect Bits
Microarchitectural state	77.94%	14.80
Architectural state	96.37%	10.4
Top-level port	97.9%	4.625

Table 2: Average logical masking rates and number of bits corrupted for soft errors occurring in logic for random machine state

architectural state and the top-level output ports of the microprocessor are shown in Tables 1 and 2 for faults injected into registers and logic gates respectively. Along with the average masking rates, we note the average number of incorrect bits for each error class. It is interesting to note here that even though we only observe the effects of a fault for the cycle directly subsequent to fault injection, a significant number of errors are still found in architectural state elements.

As demonstrated in Tables 1 and 2, the logical masking rate for microarchitectural state is much higher for faults injected into logic gates than it is for faults in sequential state elements. This trend holds across all experiments in which we analyze the effects of soft errors on microarchitectural state. This result is intuitive because on average, the size of the logic cone from an arbitrary logic gate to its target register(s) is typically about half the average size of the logic cone from a register to its potential targets, thus increasing the number of opportunities for soft errors that occur in registers to be stored.

Another trend exhibited in Tables 1 and 2 that holds throughout each of our experiments is that the average number of microarchitectural and architectural bits that are incorrect within the design when a soft error manifests is substantially larger for soft errors occurring in logic elements. Soft errors occurring in logic are exhibited as multi-bit errors and typically occur when soft errors corrupt control logic. These types of errors can be particularly disastrous when decoders for large state arrays such as the register file are subject to transient pulses.

3.2 Workload-Specific Logical Masking Rate

Many of the configurations loaded onto the design in the previous experiment from Section 3.1 are either not valid or not reachable under normal operating conditions. In order to better understand the effects of soft errors on our processor core under typical operating conditions, we conducted another series of fault injection experiments while running an image processing application on the design. The application executed is a commonly used image processing kernel rgb2yuv, which converts an image from the RGB to the YUV color-space. The application is run for at least 2,500 cycles to ensure that the microprocessor structures are warmed before any fault injection experiments are conducted. The results demonstrating the logical masking rates for this experiment are shown in Tables 3 and 4 for soft errors occurring in registers and combinatorial logic, respectively. As a further analysis of the average number of bits corrupted, errors in this experiment are tracked throughout the processor core for twenty cycles after fault injection. The results for the average number of bits corrupted per error class are shown in Figure 3

Error Location	Masking Rate	Incorrect Bits
Microarchitectural state	6.47%	1.26
Architectural state	88.35%	1.0
Top-level port	89.32%	1.15

Table 3: Average logical masking rates and number of bits corrupted for soft errors occurring in registers when running the rgb2yuv benchmark

Error Location	Masking Rate	Incorrect Bits
Microarchitectural state	78.44%	20.92
Architectural state	94.74%	5.53
Top-level port	95.12%	5.78

Table 4: Average logical masking rates and number of bits corrupted for soft errors occurring in logic when running the rgb2yuv benchmark

The most significant trend to be observed in Tables 3 and 4 is the sharp decrease in logical masking at the microarchitectural level for faults occurring in registers. This seems to occur simply because valid machine states tend to offer less logical masking potential, that is, the paths between pairs of registers is more often sensitized for valid machine states. Despite the surprisingly low microarchitectural masking rate, the architectural masking rates presented in Tables 3 and 4 tend to be consistent, though slightly greater than those demonstrated in previous work [9, 6]. In general, these high masking rates can be attributed to the fact that the design used in our experiments is a production core which contains a number of test and debug structures which typically remain dormant during normal operation. However, as we demonstrate later, these structures are responsible for increasing the overall soft error masking rate, but they are also a point of severe weakness, where faults at particular nodes can have catastrophic results (for instance a pulse occurring at the scan-enable node at the base of a scan chain).

In Figure 3, we demonstrate the average number of manifested errors per fault per cycle for each error class. This figure further exemplifies the trend noted in Section 3.1, showing that the average number of manifested errors per fault for both microarchitectural and architectural state is significantly larger for soft errors occurring in logic elements rather than sequential state. This trend is examined in more detail in Section 3.4.

3.3 Workload-Specific Logical and Temporal Masking Rate

In order to analyze the effect of temporal masking on soft

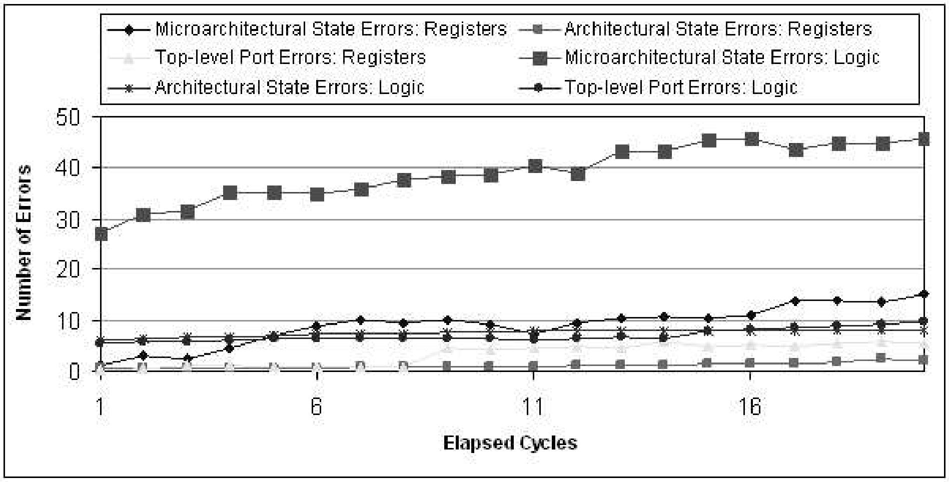

Figure 3: Average number of incorrect bits per error type over the twenty cycles subsequent to fault injection

Error Location	Masking Rate	Incorrect Bits
Microarchitectural state	83.76%	41.49
Architectural state	96.59%	13.29
Top-level port	96.33%	9.74

Table 5: Average logical and temporal masking rates combined and the number of bits corrupted for soft errors occurring in logic when running the rgb2yuv benchmark

errors occurring in combinatorial logic, we repeat the previous experiment from Section 3.2 using the rgb2yuv algorithm, injecting faults randomly in time across clock cycle boundaries and maintaining the faults for random durations on the interval $[0.25 * CLK, CLK]$. The combined logical and temporal masking rates are shown in Table 5.

Table 5 demonstrates that temporal masking has a somewhat marginal effect on the overall soft error masking rate. The masking rate for microarchitectural errors is only increased by about 5% and even less for architectural errors and top-level ports when timing information is taken into account. Also, it is important to note that as clock cycle times continue to decrease, the latching window time will become more significant with each technology generation, thus further reducing the effects of temporal fault masking.

When comparing the masking rates of faults occurring in state elements against those occurring in logic as seen in Tables 3 and 5, the difference in the microarchitectural masking rate is dramatic. While this may appear to imply that faults in logic are not nearly as significant a problem as faults in sequential state, the difference in error rates between faults occurring in logic and faults occurring in sequential state for architectural registers is only about 8%. This means that even though soft errors occurring in logic are much more likely to be masked at the microarchitectural level, at the software interface, the masking rates are not substantially different. Further, as can be seen by the average number of bits of architectural state affected, faults occurring in logic may have a much more dramatic effect on program execution.

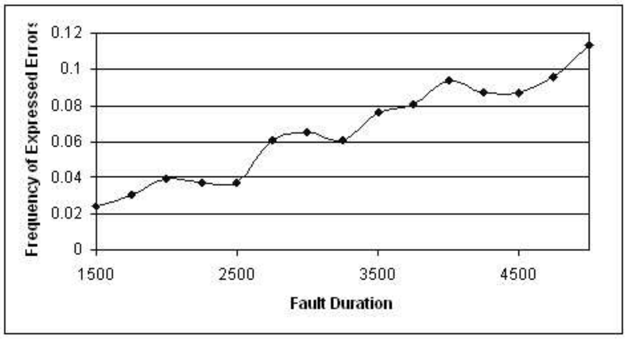

Figure 4: Relative frequency of transient pulse durations leading to error(s), durations shown here are in picoseconds

In order to further understand the effects of pulse duration on the overall soft error masking rate, we analyze faults occurring at worst-case nodes in the design. That is, we restrict fault injection to the output of sequential state elements and vary the pulse duration randomly across the clock cycle time, thus ensuring worse than average delay between fault injection site and potential target latches. The results of this experiment are shown in Figure 4. Figure 4 shows that there is a definite correlation between the fault duration and the frequency with which errors are expressed in the microprocessor. However, even very small pulses were still able to cause a significant number of errors, and given that particle strikes are likely to be random throughout the depth of the circuit, it is clear that even very short pulses may prove problematic in future technology generations.

3.4 Soft Error Propagation Behavior

In this section, we analyze how soft errors tend to propagate through the system over time in both architectural and microarchitectrual state. We define the architectural state as the set of 37 software-visible physical registers defined in the ARM ISA [7] and the microarchitectural state as the set of all state elements within the design, excluding the architectural registers. We again perform fault injection experiments while executing the rgb2yuv application as our

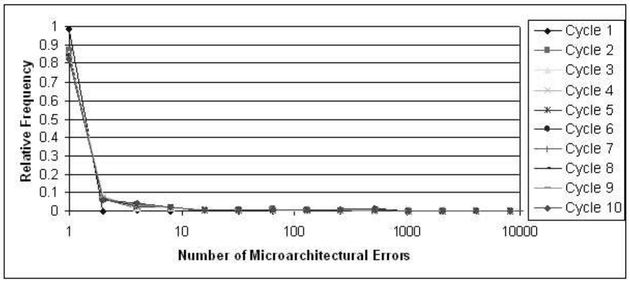

Figure 5: Frequencies for the given number of microarchitectural state errors when faults are injected into registers

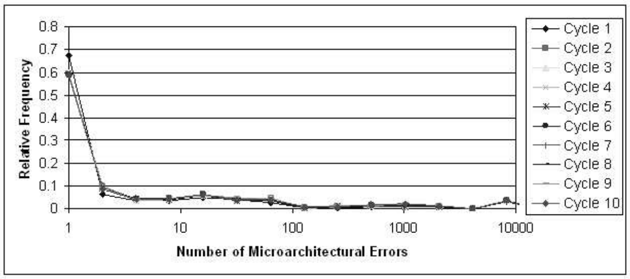

Figure 6: Frequencies for the given number of microarchitectural state errors when faults are injected into logic

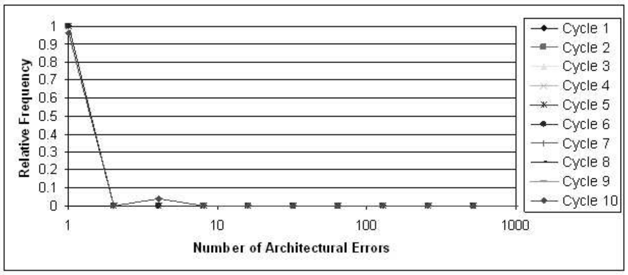

Figure 7: Frequencies for the given number of architectural state errors when faults are injected into registers

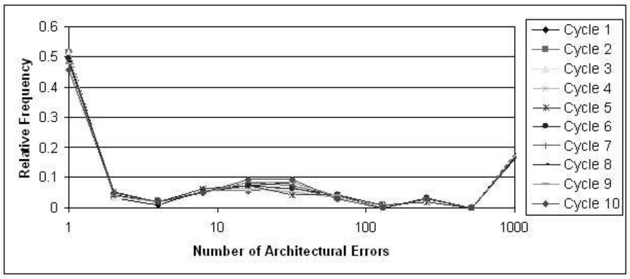

Figure 8: Frequencies for the given number of architectural state errors when faults are injected into logic

workload. Figures 5 and 6 show the relative frequencies for which a given number of microarchitectural state errors occur for the ten cycles subsequent to fault injection.

Figure 5 demonstrates that if a soft error occurring in a register is manifested, it leads to a single bit error at the microarchitectural level more than 80% of the time. Further, it is shown to be extremely likely that this error will remain a single bit error over the course of its lifetime. Alternatively, Figure 6 shows that soft errors occurring in logic may have a dramatic effect on microarchitectural state when they are manifested. For example, multi-bit errors occur in more than 30% of the fault injection experiments when the simulated soft errors are not masked. Further, nearly 5% of the faults which manifest errors corrupt over 6,500 microarchitectural registers, or more than 30% of the microarchitectural state elements in the microprocessor. These types of catastrophic effects are typically the result of faults within test equipment such as JTAG or design for test related logic. These cases are discussed further in Section 4.

In order to better understand the propagation of errors within the system and how they may potentially effect software running on the design, we also demonstrate the propagation behavior of soft errors into architectural state. Figures 7 and 8 demonstrate the relative frequency of architectural state bit errors manifested for the ten cycles after fault injection.

Figure 7 shows that multi-bit architectural state errors tend to occur very rarely when soft errors affecting registers are not masked. However, as shown in Figure 8, when soft errors occurring in logic manifest as errors in architectural state, multi-bit errors of four bits or more account for more than 45% of the occurrences. Further, as shown by the spike at the end of Figure 8, more than 15% of the faults originating in logic that affect architectural state cause more than 90% of the architectural state bits ($\sim$ 1000 state elements) to hold incorrect values. This is typically the result of faults occurring in the decode stage of the pipeline or in decode logic for the register file.

To elucidate the architectural effects presented in Figures 7 and 8, we also study the number of incorrect bits per architectural register and the number of architectural registers within the register file that contain errors. Figures 9 and 10 demonstrate the frequencies of incorrect bits per architectural register when soft errors occur in registers and logic respectively and Figures 11 and 12 show the number of architectural registers containing errors for each fault.

Figure 9 shows that the manifestation of soft errors as multi-bit architectural register errors tends to be very rare for soft errors occurring in state elements. This data corroborates the results presented in [9] and the thesis that a potential low-overhead protection mechanism against soft errors occurring in state elements would be to provide ECC or parity bits on the register file.

Figure 10 shows that multi-bit errors where the entire 32 bits of the architectural register are corrupted, tends to be the norm when soft errors occurring in logic are manifested in the register file. This demonstrates that the ECC protection used to combat soft errors occurring in state elements would tend to fail in protecting against soft errors in logic. Further, Figure 12 shows that not only are multi-bit ar-

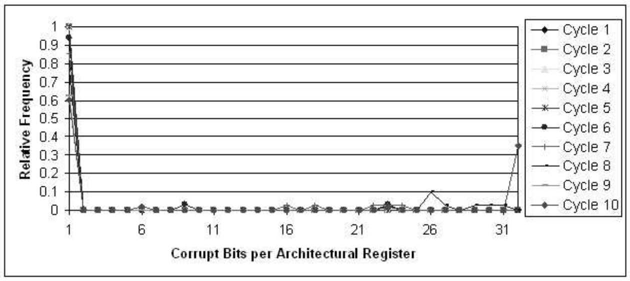

Figure 9: Frequency that a given number of bits per architectural register are corrupted when faults are injected into registers

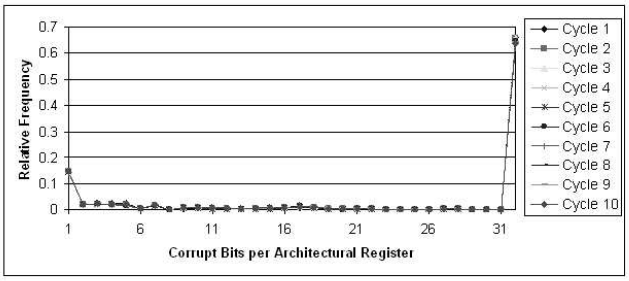

Figure 10: Frequency that a given number of bits per architectural register are corrupted when faults are injected into logic

chitectural register errors likely, but multiple architectural register corruptions also occur with a moderate frequency.

4. DISCUSSION OF RESULTS

The results demonstrated in the previous section corroborate those shown in previous work characterizing the effects of soft errors in state elements [3, 9]. Further, the results provide some insight into why faults in combinatorial logic typically are responsible for a greater percentage of manifested errors as shown in [6]. A key characteristic of faults occurring in combinatorial logic is the likelihood of fault fanout and manifestation as multi-bit errors throughout the system at the microarchitectural and architectural levels. In [8], Shivakumar et al. has shown that soft errors occurring in combinatorial logic are expected to increase in frequency, matching the rates observed in unprotected SRAM cells, and Mitra et al. [4] report that 18% of the overall soft error rate of a state-of-the art design is already consumed by soft errors occurring in combinatorial logic. These results demonstrate an increasing need for understanding of the behavior of soft errors in combinatorial logic and the development of new techniques to protect against them.

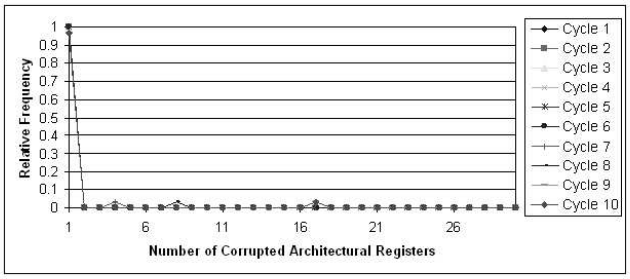

Figure 11: Frequency that a given number of architectural registers contain errors when faults are injected into registers

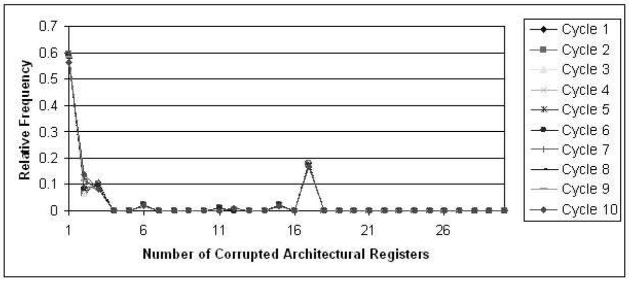

Figure 12: Frequency that a given number of architectural registers contain errors when faults are injected into registers

Figures 9 and 10 from the previous section demonstrate that not only are multi-bit architectural register errors the common consequence of soft errors manifested in logic, but multi-architectural register corruptions also occur with moderate frequency. Occurrences such as these defy commonly accepted state element protection mechanisms such as ECC and parity, and demonstrate a dramatically different landscape of the soft error problem at the microarchitectural level than previously reported.

While multi-bit errors pose problems for some accepted circuit-level soft error detection and correction mechanisms, at the microarchitectural level, the phenomenon of soft error fan-out also opens the door for a new class of low overhead detection mechanisms. By conducting statistical analysis of average fan-out rates within logic blocks, it becomes possible to achieve high coverage transient pulse detection without protecting all state in the design. For example, using statistical analysis to strategically place clock-delayed shadow latches, similar to those used in RAZOR [2] to detect timing errors, could be used to provide high-coverage protection for soft errors in combinatorial logic.

The problems of fault fanout and multi-bit errors may not be unique to the embedded design space, however, they are accentuated by it. High-performance microprocessor designs are typically limited in the logic depth allowed between sequential elements in order to keep the clock frequency as high as possible. Not only does this decrease the level of fanout, but it also decreases the relative area of combinatorial logic as more pipeline registers become necessary. However, with increasing integration and higher complexity within designs, it is expected that high performance designs will tend towards results similar to those as demonstrated here in future generations.

Also important to a thorough understanding of the soft error problem is the implication of the increasing need for components included in the design for test purposes. An interesting occurrence, demonstrated by the tail in Figure 6, is that of massively multi-bit errors, typically caused by soft errors in test equipment. As designs become increasingly complex, the need for higher observability and controllability for testing at manufacture time becomes critical. However, it seems that the mechanisms that are included for test purposes are some of the most sensitive areas within the design. Microprocessors are manufactured and shipped with dormant test equipment, such as JTAG, logically tied to 0. However, faults within this test equipment logic tend to manifest with moderate frequency and when they do, the effects are dramatic. For example, each register within the ARM926EJ-S is a scan register meaning that when presented with a scan enable signal, it captures its input from its predecessor along the scan chain rather than its standard input. Each register in the design includes logic dedicated to propagating scan enable signals and scan data, and faults at these nodes tend to cause substantial state errors within the system.

5. RELATED WORK

Kim and Somani [3] conduct software-simulated fault injection campaigns on an RTL model of the PicoJava-II microprocessor to determine the *soft error sensitivity* of logic blocks within the design. The soft error sensitivity (SES) metric used in this work is defined as the probability that a soft error within a given logic block will cause the processor to enter an incorrect architectural state. The fault model used in this work is similar to our own, though the authors of this paper conduct analysis strictly at the architectural level.

In [5] Mukherjee et al. define the term *architectural vulnerability factor* (AVF) to be the probability that a fault in a microarchitectural structure will cause an error in program output. The authors use a performance simulator of the Itanium II microarchitecture to determine the AVFs for structures within their simulated microarchitecture. Our work presents similar results at the architectural level, but focuses rather on the microarchitectural effects of soft errors.

Wang, et al. [9] characterize the effects of soft errors on an out-of-order, superscalar Alpha-like processor core. The fault model used in this work simulates single bit flips in sequential state elements within the design, and an analysis of the failure modes exhibited in simulation is described. In this work the authors explore the effects of soft errors on a substantially different microarchitectural model and limit their studies to fault occurring in state elements.

Saggese, et al. [6] present a similar analysis of the effects of soft errors occurring in both sequential state elements and combinatorial logic on a DLX microprocessor model. The error manifestation rates demonstrated in their work are corroborated by our own, however, in our work we have chosen to focus on the error propagation behavior exhibited at the microarchitectural level rather than a sensitivity analysis of different blocks within the design.

6. CONCLUSION

In this work, we present a design-independent framework for injecting and analyzing the effects of soft errors at the microarchitectural level. We use this framework to empirically derive the logical and temporal soft error masking rates for a production-level ARM926EJ-S microprocessor core. Further, we provide an analysis of the propagation behavior of soft errors and their microarchitectural effects over time on this design and identify a number of critical distinctions in the propagation behavior of soft errors that occur in sequential state elements as opposed to combinatorial logic.

Our results corroborate previous results studying the architectural error rates of soft errors occurring in state elements [9, 3], but also identify potential problems with using traditionally accepted state protection mechanisms, such as ECC or parity, as soft errors affecting combinatorial logic become more frequent. Lastly, we identify some potential problems that may arise due to the effects of increasing testability on system reliability.

7. REFERENCES

[1] ARM Ltd. *ARM926EJ-S Technical Reference Manual*, Jan. 2004. http://www.arm.com/pdfs/DDI0198D_926_TRM.pdf.

[2] T. Austin, D. Blaauw, T. Mudge, and K. Flautner. Making typical silicon matter with razor. *IEEE Computer*, 37(3):57–65, Mar. 2004.

[3] S. Kim and A. Somani. Soft error sensitivity characterization for microprocessor dependability enhancement strategy. In *International Conference on Dependable Systems and Networks*, pages 416–428, June 2002.

[4] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim. Robust system design with built-in soft-error resilience. *IEEE Computer*, 38(2):43–52, Feb. 2005.

[5] S. S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high performance microprocessor. In *International Symposium on Microarchitecture*, pages 29–42, Dec. 2003.

[6] G. P. Saggese, A. Vetteth, Z. Kalbarczyk, and R. Iyer. Microprocessor sensitivity to failures: Control vs. execution and combinatorial vs. sequential logic. In *International Conference on Dependable Systems and Networks*, pages 760–769, June 2005.

[7] D. Seal. *ARM Architecture Reference Manual*. Addison-Wesley, London, UK, 2000.

[8] 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 *International Conference on Dependable Systems and Networks*, pages 389–398, June 2002.

[9] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *International Conference on Dependable Systems and Networks*, page 61, June 2004.