Low-cost Protection for SER Upsets and Silicon Defects

Mojtaba Mehrara Mona Attariyan Smitha Shyam Kypros Constantinides
Valeria Bertacco Todd Austin

Advanced Computer Architecture Lab
University of Michigan, Ann Arbor, MI 48109
{mehrara,monattar,smithash,kypros,valeria,austin}@umich.edu

ABSTRACT

Extreme transistor scaling trends in silicon technology are soon to reach a point where manufactured systems will suffer from limited device reliability and severely reduced life-time, due to early transistor failures, gate oxide wear-out, manufacturing defects, and radiation-induced soft errors (SER). In this paper we present a low-cost technique to harden a microprocessor pipeline and caches against these reliability threats. Our approach utilizes online built-in self-test (BIST) and microarchitectural checkpointing to detect, diagnose and recover the computation impaired by silicon defects or SER events. The approach works by periodically testing the processor to determine if the system is broken. If so, we reconfigure the processor to avoid using the broken component. A similar mechanism is used to detect SER faults, with the difference that recovery is implemented by re-execution. By utilizing low-cost techniques to address defects and SER, we keep protection costs significantly lower than traditional fault-tolerance approaches while providing high levels of coverage for a wide range of faults. Using detailed gate-level simulation, we find that our approach provides 95% and 99% coverage for silicon defects and SER events, respectively, with only a 14% area overhead.

1. INTRODUCTION

The advent of silicon technologies below 65nm, has created a widely held concern for the reliability of transistors and interconnects. Leading technology experts have voiced predictions that overall system reliability will be severely compromised by a number of burgeoning failure mechanisms, including radiation-induced transient errors, transistor wear-out, early failure, and manufacturing defects [23]. They warn that without the addition of fault-tolerant design techniques, overall system reliability will be significantly impacted, resulting in shorter times to first failure and reduced manufacturing yield. In this paper, we detail a low-cost technology to harden a microprocessor pipeline and cache memory system against these reliability threats.

Below 65nm, a number of silicon failure modes are sharply accentuated, creating a hostile operating environment ripe with a variety of transient and permanent failure mechanisms. Transient faults, or soft errors as they are often called, are the result of energized alpha or neutron particles striking logic components, creating logic glitches which can potentially corrupt latch inputs or memory cells. While there are a number of situations that can mask SER faults (*e.g.*, microarchitectural or logical masking), significant concerns remain that SER effects will grow with decreasing process technologies and shortening clock cycles.

More recently, focus in the computer architecture and VLSI research communities has started to turn toward the problem of silicon defects. These failures are the result of transistors and interconnects that are either defective when manufactured, or fail in the field due to operational stresses. A number of important silicon failure mechanisms have emerged, in particular:

Silicon Wear-out: Extreme device scaling results in extremely small device structures, with only 10's of atoms of thickness in the dimension of critical structures. If exposed to prolonged voltage or temperature stresses, these fragile structures can fail. Many failure modes are possible, including metal electro-migration, hot carrier degradation, and gate breakdown.

Early Transistor Failures: Deep submicron scaling increases the probability of early transistor failures, due to the reduced effectiveness of component burn-in. Small leaky transistors can induce thermal runaway, a side-effect of burn-in which can destroy robust components that would otherwise pass burn-in testing.

Device Defects: A number of fabrication factors, such as optical proximity effects, airborne impurities, and processing material flaws can result in defective transistors and interconnect. Additionally, the gate oxides used in deep submicron have thinned to the point where process variation can lead to currents penetrating the gate, rendering a transistor broken.

Classic fault-tolerant techniques are available which can tolerate many of these faults, including parity or error correction codes (ECC), dual or triple-modular redundancy, time-redundant computation or checkers [22]. However, these techniques suffer from large overheads that start at 100% and quickly rise if the design must accommodate high availability. In contrast to traditional techniques, the design presented here leverages domain-specific fault-tolerance techniques that significantly reduce the performance and area overheads of providing high-coverage support for fault detection, diagnosis, recovery and repair.

1.1 Contributions of This Paper

We present a low-cost technology to protect a processor pipeline and its cache memory system from both transients faults and permanent silicon defects. To keep costs low, we only provide guarantees for the first silicon defect (although it is likely that the design presented could tolerate additional defects). Given this fault model, we exploit a novel combination of on-line distributed checkers and microarchitectural

978-3-9810801-2-4/DATE07 © 2007 EDAA

checkpointing which efficiently identify defects and recover from their impact. The microarchitectural checkpointing mechanism implements a capability to roll back execution up to 1000's of cycles. Using the protection of checkpointing, we periodically exercise the on-line distributed checkers to verify the functional integrity of the hardware. If the online tests succeed, the underlying hardware is known to be free of defects, and the previous checkpoint is no longer needed. If a defect is detected, processor state is restored through the last checkpoint, and we repair the hardware by reconfiguring it to operate without the defective component, possibly with a slight performance degradation. We utilize the natural redundancy of instruction-level parallel processors to reduce repair costs. Finally, we employ a double-sampling latch design to protect the pipeline from transient faults and latch defects.

The design presented in this paper is based on our earlier proposed reliable pipeline solution presented in [21]. We used the same processor design as our baseline system, and we enhanced the previously presented technology with the following novel features:

1. We extend the resilient design methodology to tolerate both transient and hard silicon faults, with lower cost and better coverage than previously published approaches. For a 14% area overhead, we provide 99% and 95% coverage against transient faults and silicon defects, respectively. This is a marked improvement in coverage over our previous design [21] (which did not have protection against transient faults or control logic defects) and an order-of-magnitude improvement over traditional techniques such as triple modular redundancy (TMR).
2. We introduce a novel *reflexive self-test* technique which allows each distributed checker to check itself. This approach obviates the need for expensive logic to check the functional integrity of the checkers, resulting in higher overall fault coverage with no increases in area costs.
3. We develop a general approach to protect arbitrary control logic blocks from defects. Our approach, based on dual-modular redundancy, provides high coverage of control logic block defects with moderate cost. We demonstrate through detailed physical design analysis that protection of control logic is vital to achieve high overall fault coverage.

A key aspect of our work is the observation that online testing techniques provide high levels of resiliency at much lower cost than traditional redundant computation techniques. Approaches such as triple-modular redundancy and N-version hardware exercise redundant hardware on each cycle, with area and power overheads of 100% or more [22]. With the online testing-based solutions used in this work, computation is not checked, rather, the underlying hardware is periodically verified. If a faulty component is detected, the computation is repaired by restoring the last known-good checkpoint and disabling the broken resource.

The remainder of this paper presents our resilient processor design approach, and assesses its impact on fault coverage, design cost, and system performance. Section 2 overviews the proposed design technique, including support for online testing and microarchitectural checkpointing. Our soft-error detection scheme is described in Section 3. In Section 4 we present a detailed simulation-based evaluation of the approach. Finally, Section 5 reviews related works, and Section 6 gives conclusions.

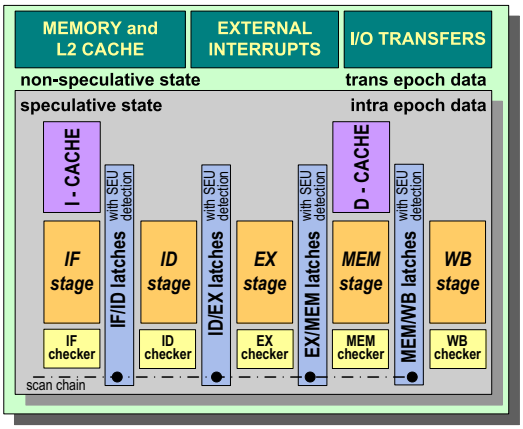


Figure 1: Fault-tolerant pipeline architecture.

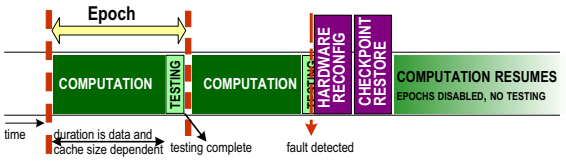


Figure 2: Epoch-based execution timeline.

2. SILICON DEFECT PROTECTION

An overview to our resilient processor design technology, which was first introduced in [21], is illustrated in Figure 1. The technique relies on a register and memory checkpointing mechanism implemented transparently in the microarchitecture, which creates protected *epochs* of computation. As illustrated in Figure 2, an epoch is an interval of computation, usually thousands of cycles long, which can be undone at any time for the purpose of recovering from potentially erroneous results created by a silicon defect or transient SER fault. The amount of computation within an epoch is variable and it is determined by the temporary storage capacity of the processor's data cache and the occurrence of interrupts. As shown in Figure 2, the processor runs unfettered for an entire epoch, after which the built-in self-testing infrastructure fully tests the underlying hardware. This is in contrast with [21] where the testing was concurrent with the processor's execution. The main benefit of employing this approach compared to the work in [21] is that it eliminates the need for storing two checkpoints and two-phase state commits, and in case of an error, the recovery can be done by simply reloading the previous checkpoint. This approach leads to greater area savings, due to smaller checkpointing state, at the expense of a slightly larger performance overhead.

If the built-in self-tests pass, the hardware is known to be free of defects, and the computational epoch can be safely retired to non-speculative storage. In the event that the tests fail, the underlying hardware has been found to be defective due to a silicon defect. In this case, the computation in the previous epoch may have been corrupted by the defect. Consequently, the speculative results must be invalidated and the state of the processor restored to the beginning of the previous epoch, which is the last known correct state as the pipeline had just passed a successful on-

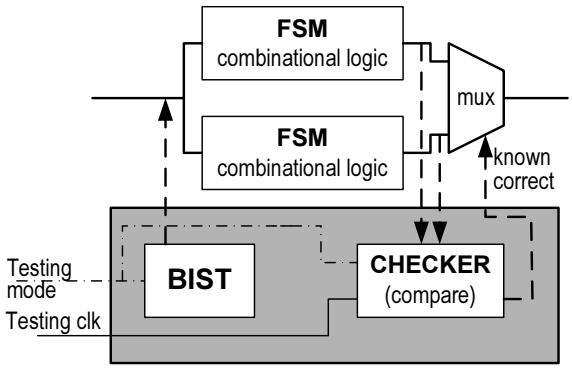


Figure 3: Control logic checker network.

line test battery. Before the mainstream computation can advance, however, the faulty component must be repaired. In a processor with instruction-level parallelism (ILP), we can rely on multiple copies of nearly all resources, hence, we repair the system by disabling the defective component and by reconfiguring the processor so that to avoid utilizing that unit. Once repaired, the processor will continue to run in a performance-degraded mode.

In the remainder of this and the following section we detail the enhancements made to the resilient processor design of [21], focusing on improving fault coverage through better protection of the system's control logic and new support for transient faults. Details of the BIST techniques employed in other parts of the processor can be found in [21].

Protecting Control Logic. To achieve high fault coverage it is critical to protect the control logic, since this logic constitutes a non-trivial fraction of the area in most processor designs. For the pipeline's control logic of our prototype design, we employ a dual-modular redundancy (DMR) based approach, as illustrated in Figure 3. We run two copies of the pipeline control logic in parallel, each with the same set of inputs. Every cycle, the outputs of the control blocks are compared and if any difference occurs, a fault is flagged. To localize the fault, we use BIST to determine which of the two control block copies is defective. Once identified, the defective control logic block is permanently disabled. Note that this approach can only tolerate defects to the extent that they occur in only one of the two control logic blocks. This technique has area-cost advantages over triple-modular redundancy because checker and BIST logic are typically smaller than a third copy of control logic as required to implement TMR. Note also that we protect each control logic block individually, leading to smaller overhead in the interconnect and higher resiliency.

Checker, Check Thyself. The checkers constitute a non-trivial portion of the processor's area – more than 10% of the area of our prototype design. Consequently, if the checkers themselves were not checked, they would severely limit overall design fault coverage. To keep area costs low, checkers are checked using the same component they monitor, a technique we call *reflexive self-test*. In other words, our online test harnesses are designed such that they produce a correct result only when *both* the unit-under-test and the checker are free of silicon defects and other faults. For example, a BIST-vector generator and a 9-bit adder is used to check the processor's adder. At the same time, the processor's adder is used to test the functional integrity of the BIST-vector generator and the 9-bit adder.

In traditional testing the BIST vectors are selected so that they have a high probability to detect defects in the unit under test. In reflexive testing we add the additional constraint that the test vectors must also expose defects in a broken checker (assuming that the unit under test is still working). Consequently, assuming a single-defect fault model, a BIST test will fail if there is a defect either in the unit-under-test or in the checker. If the defect is in the checker, the end result will be the disabling of the working unit and its broken checker, hence the desired result of disabling the defective checker component is achieved as a byproduct.

Checkpointing and Recovery. We employ a microarchitectural checkpointing and rollback mechanism to restore correct program state in the event of a defect or transient fault. This mechanism is similar to the one described in [15]. Within each computational epoch's execution, register and memory updates are buffered in speculative state to prevent any external memory corruption. At the end of each epoch, online tests are run to verify that the underlying hardware is functionally correct. In case the hardware is found fault free, the speculative state is committed and the processor continues with the next epoch; otherwise, the program state is rolled back to the beginning of the epoch, the hardware is repaired, and program execution continues unimpaired. Figure 1 shows the boundary between speculative and committed (non-speculative) state.

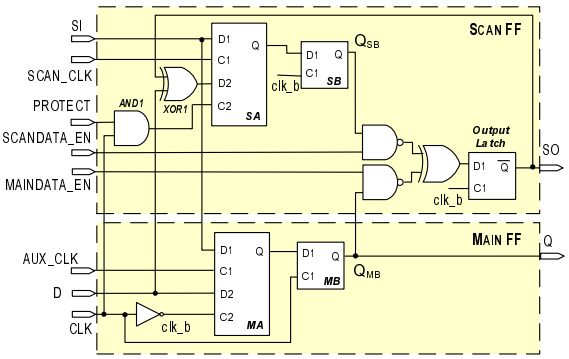
A single-port SRAM is used to store the registers state at the beginning of each epoch. Memory updates are held speculatively by adding a volatile bit to each cache line. These bits are reset at the beginning of an epoch. Subsequently, when a store operation is executed, the value is stored to a cache line and the corresponding volatile bit is set to indicate that the contents are speculative in the current epoch. The extent to which the cache can buffer speculative memory updates determines the length of each epoch. Therefore, a cache miss on a set in which all lines are marked as volatile terminates the computational epoch and forces the processor to either commit the state (if end-of-epoch hardware tests pass) or rollback to the previous checkpoint (in the event of a failure). Committing of state is done by clearing all volatile bits from the cache lines and thereby moving all formerly speculative state to non-speculative. Rollback of the state is implemented by flushing the pipeline and restoring architectural registers from the backup register file. Additionally, all volatile cache lines are invalidated to prevent any corrupted memory update.

Sequential Elements. We achieve coverage for the design's sequential elements as a byproduct of our transient fault protection technique which is presented in the following section.

3. TRANSIENT FAULT PROTECTION

We propose a novel circuit for transient fault detection, based on a double-sampling scan flip-flop. Figure 4 depicts the proposed fault-tolerant scan cell detecting soft errors in both sequential and combinational logic. In addition, it can detect hard failures in sequential elements. The figure also lists different operating modes of the cell and their corresponding input configurations.

Our SER-tolerant flip-flop is composed of a main FF block and a scan FF block where each block includes a master and a slave latch. In addition the scan FF block contains an XOR gate detecting when the two master-slave FFs have latched



INPUT CONFIGURATIONS FOR CELL USE

	PROTECT	SCANDATA_EN	MAINDATA_EN	AUX_CLK
Normal op. – no protection	0	0	0	0
Normal op. – with protection	1	1	1	0
Shift out SI or error signal	0	1	0	0
Main FF data to scan chain	0	0	1	0
Test FF for hard failure	0	1	1	Single pulse

ENERGY-DELAY CHARACTERISTICS (normalized to a regular scan cell)

Nominal power	1.96	Area	1.5
Power @error cycle	2.69	CLK-Q delay	1.12

Figure 4: SER-Tolerant flip-flop.

different values (as it is the case when an SER hits) and an additional latch storing this information permanently. The two blocks are fed with two distinct clocks, the main clock and a skewed clock. In our case the skewed clock is the inverse of the main clock and is indicated in the Figure as *clk_b*. The main FF latches the incoming data signal on the positive edge of the clock, while the scan FF samples the same signal on the skewed clock's positive edge. We made the assumption that transient faults manifest as glitches of less than half clock cycle duration (which is a safe assumption up to designs operating at several GHz) [17, 26]. Hence, if an incorrect value is latched in the main FF due to an SER, the glitch will subdue before the signal is latched again half a clock cycle later by the scan block. When this situation occurs the XOR gate outputs a 1, which is stored in the *Output Latch* right away. In addition, the output signal *SO* is fed back to the *XOR1* gate, which forces the input of the scan FF to always observe the complement of the data signal, continuously forcing an "SER-detected" situation. Figure 5 shows a timing diagram of the situation just described. The *protect*, *scandata_en* and *maindata_en* are enabling signals which are always active during the normal protected operation. Note that for our flip-flop design to work, we must enforce a minimum path delay constraint of 50% of the clock cycle.

At the end of each computation epoch all error signals (SO) are shifted out through the scan chain (using our *shift out* configuration). We partitioned the latches into zones to speed up this process, and we can collect all the error signals within 40 cycles, which is below the time required by the BIST testing process. If an error is detected, each cell within the zone is evaluated to discern between an SER or a permanent latch failure. We do so using the *si*, *scan_clk* and *aux_clk* signals. If the error does not repeat, we assume an SER occurred and we simply trigger the rollback mechanism to restore the previous known correct state before proceeding. Otherwise, we derive that the latch has failed permanently and we detect which of the scan FF or the main FF is faulty and disable it. To contain the amount of interconnect, we designed our system so that we can disable the scan FF block or the main FF block for all flip-flops within the same zone.

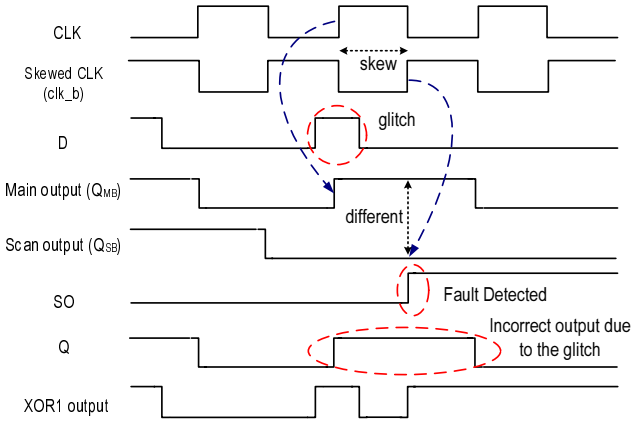


Figure 5: Timing diagram for a transient fault.

The bottom of Figure 4 shows the results of timing and power simulations on the error trapping cell with skewed input clock. The output latch and the extra gates we used for implementing the correct functionality account for the increase in power, area and delay in our design, compared to a simple latch. The overall area overhead of using this cell instead of ordinary scan cells is about 1.64% of the total processor area, which is a nominal amount compared to the amount of coverage improvement. By using this approach, we can cover nearly all soft errors in combinational and sequential logic.

4. EXPERIMENTAL EVALUATION

In this section we present a detailed physical design of a 4-wide VLIW processor including 32KB instruction and data cache, enhanced to include our low-cost protection against SER faults and silicon defects. We examine the performance of the design, using both circuit timing simulation and architectural simulation to assess the impacts of fault protection. We also examine the cost of the defect protection technology by measuring area overhead of the testing logic (*e.g.*, vector generation and checkers). Finally, we evaluate our fault coverage, *i.e.*, what fraction of defects randomly placed are protected, by carefully measuring the portion of silicon area protected.

Evaluation Framework. Circuit-level evaluation was performed on the prototype 4-wide VLIW prototype, specified in Verilog, and synthesized for a 0.18um TSMC process using Synopsys Design Compiler. Architectural evaluation was done using the Trimaran tool set, a re-targetable compiler framework for VLIW/EPIC processors [24], and the Dinero IV cache simulator [8]. We evaluate our designs by running benchmarks from SPECint2000, MediaBench [11] and MiBench [7] benchmark suites.

Self-test BIST vectors were generated using carefully hand-selected vectors, or by randomly cycling through random vector sets until a small group of effective vectors was located. Once the test vector set is identified, it is encoded into an on-chip ROM storage unit, created using Synopsys design tools. Coverage analysis simulation is performed by

Design Block	Total area (mm^2)	Checker area (mm^2)	% of Total Area	Protected area (mm^2)	% of Area (Coverage)
IF	0.127	0.008	6.6	0.114	89.8
ID	0.278	0.023	8.2	0.261	93.6
RF	2.698	0.133	4.9	2.635	97.7
EX	2.993	1.166	39.0	2.896	96.8
WB	0.171	0.007	4.2	0.158	92.7
Latches	0.164	0.122	1.4	0.164	99.9
Overall Core	6.431	1.459	22.7	6.228	96.8
I-cache	2.033	0.009	0.5	1.881	92.5
D-cache	2.044	0.009	0.5	1.892	92.6
Overall System	10.508	1.477	**14.1**	10.001	**95.2**

Table 1: Area costs of individual design components and overall system.

injecting faults into a logic simulation of the detailed VLIW processor gate-level design.

Fault Coverage. In this section we examine the coverage of our fault-tolerant mechanisms, by measuring the fraction of faults covered through fault injection experiments. This fraction represents the overall design defect coverage. Table 1 lists the coverage of the overall design, as well as the coverage of individual processor components. Overall design coverage is 95%, meaning that 95 out of 100 defects randomly placed into the processor are covered in our fault-tolerant design.

The unprotected area of the design mainly consists of resources that do not exhibit inherent redundancy in the design such as glue logic blocks. We are currently developing system-level protection solutions to provide defect tolerance for these miscellaneous resources. Our ultimate future goal is to push overall coverage to 99% while keeping area costs low.

Area Overheads. The addition of test vector ROMs, where test vectors are stored, plus the checkers and checkpointing infrastructure bears a cost on the overall size of the design. Table 1 lists the total area of the defect tolerant component (Total area), the defect protection infrastructure area (Checker area), and the area that is covered by the test harness (Protected area). As shown in Table 1, area overheads for defect protection are quite modest, with most overheads less than 10%. The overheads within the caches are even lower, less than 1% for the prototype. Consequently, the overall overhead for defect protection is quite low. Adding support for defect and transient fault protection increased the total area of the design by only 14%.

Performance Implications. As the system runs, it will periodically pause to run online self-tests. These pauses constitute a down-time and a potential performance loss if they occur with too much frequency. We examined the impact of our defect protection mechanism on the performance of programs running on the defect-tolerant prototype design. Table 2 lists the number of vectors to fully test each component, showing that only few vectors are required to test each unit. The bandwidth requirements of testing are the number of vectors needed to fully test components for stuck-at-0 and stuck-at-1 faults. The caches are not listed in Table 2 because they use parity bits to allow for continuous detection of defects. Clearly, the time required to fully test the hardware is quite small, only 128 cycles, with the register file taking the longest time to complete its test.

Table 3 lists statistics about computational epoch lengths for a variety of programs. Note that the processor is equipped

Component	**# of test vectors**
ALU	20
MUL	55
Decoder	63
Register File	128
Pipeline Control	12
Memory Control	13

Table 2: Number of test vectors to achieve 100% coverage for stuck-at-0 and stuck-at-1 faults.

Benchmark	**Avg. epoch size (cycles)**	**Testing Overhead (%)**
175.vpr	50499	0.51
181.mcf	120936	0.21
197.parser	106380	0.24
256.bzip2	162508	0.16
unepic	33604	0.76
epic	196211	0.13
mpeg2dec	1135142	0.02
pegwitdec	169617	0.15
pegwitenc	304310	0.08
FFT	23145	1.11
patricia	139952	0.18
qsort	1184756	0.02
Average	**302254**	**0.30**

Table 3: Epoch size statistics and testing overheads.

with a 32 KByte 4-way set-associative data cache and an eight entry fully associative volatile victim cache. We report the average epoch size in cycles and the testing overhead in each benchmark. The fact that testing only takes at most 128 cycles explains why the performance impact figures listed in the table are quite small.

5. RELATED WORK

Several approaches for providing SER and defect tolerance have been proposed in the past few years. The concept of using time redundancy methods for mitigating soft errors has been explored in [2], [20] and [16]. In [16], three samples of the input are taken at different clock edges and the final output is determined using a majority voter. In [18], the authors propose reusing scan chain resources for transient fault detection in flip-flops. They introduce two different scan cell designs which are based on blocking and trapping transient faults at the output of each flip-flop. While their approach is efficient in terms of area, power and delay overhead, it does not detect transient faults in combinational logic. The solution in [19] proposes a time-redundancy based scheme with scan-path reuse in which a time-shifted version of the input is given to the scan flip-flop. The C-element which was introduced in [18] is then used to block the error at the flip-flop's output.

The NanoBox project [10] distributes error detection and correction to the low level logic blocks, by proposing a self-correcting logic block consisting of a lookup table with appropriate error detection and correction entries. In [6] the design of a defect-tolerant chip multiprocessor switch is explored, analyzing the resulting tradeoff between defect tolerance and area overhead. In [3], the authors present (SRAS), a Self-Repairing Array Structures hardware mechanism for on-line repairing defected microprocessor array structures, such as a reorder buffer and a branch history table. The proposed mechanism detects faults by employing dedicated check rows. DIVA [25], an on-line checker component inserted into the retirement stage of a microprocessor pipeline, fully validates all computation, communication, and control exercised in a complex microprocessor core. Finally, the

work in this paper is based on the BulletProof pipeline design of [21]. The design presented herein includes a number of significant enhancements, including support for SER fault tolerance, defect tolerant control logic and self-checking checkers.

There have been a number of research efforts focused at providing defect tolerance in finite-state automatons. As a result, numerous techniques are devised to detect and correct faults in the output state bits [4, 9, 12, 13]. Many of these techniques establish specific patterns for the next state bits with a checker constantly monitoring the correctness of the pattern. Therefore, any fault that changes the pattern can be detected [1, 5, 14]. The main problem of these detection techniques is that if the pattern is simple, a defect can propagate to the output while still abiding the pattern's rules. As an example, if the next state bits use one-hot encoding, a bidirectional fault which changes a bit from 1 to 0 and another from 0 to 1 can still abide the one-hot pattern. It is possible to design more complex patterns, but the corresponding overhead is not always tolerable. Another problem with these techniques is that they can only protect next state bits, since the primary outputs of the control logic block do not necessarily follow a specific pattern.

6. CONCLUSIONS

In this paper we presented a low cost technology that protects a microprocessor pipeline and caches against transient faults caused by natural radiation-induced single-event upsets (SER) and hard silicon defects. The approach we take is notably different from traditional approaches to fault tolerance. A microarchitectural checkpointing mechanism creates speculative epochs of computation after which distributed, domain-specific on-line tests verify the integrity of the underlying hardware. Additionally, a double-sampling latch design is used to detect transient fault logic glitches which have corrupted the pipeline state. If, at the end of an epoch, the hardware is fault-free, the epoch computation is allowed to retire to non-speculative state. In the event that a fault is exposed, the program state is rolled back to the last known good program state at the beginning of the last epoch. If the fault is due to a transient fault, the epoch is re-executed, otherwise, the defective component is disabled, thereby allowing the processor to continue correct execution in a degraded-performance mode.

We have implemented a physical model of a prototype 4-wide VLIW processor which employs our low-cost solution for fault tolerance support. Analysis of this design indicates that area overheads are quite modest, providing transient and hard silicon fault protection with only a 14% increase in total area. This is a remarkable improvement over traditional redundancy-based techniques, such as triple-modular redundancy, which incurs overheads starting at 200%. Additionally, we demonstrated through gate-level fault injection studies that fault-detection coverage is very high: 95% of all hard silicon defects and 99% of all transient faults are covered. Additional simulation studies confirmed that periodic online testing has negligible impact on the overall system performance.

7. REFERENCES

[1] D. Anderson and G. Metze. Design of totally self-checking check circuits for m-out-of-n codes. *IEEE Transaction on Computers*, 22:263–269, March 1973.

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

[3] F. A. Bower, P. G. Shealy, S. Ozev, and D. J. Sorin. Tolerating hard faults in microprocessor array structures. In *Proc. of Int'l Conf. on Dependable Systems and Networks (DSN)*, 2004.

[4] D. Bradley and A. Tyrrell. Immunotronics - novel finite-state-machine architectures with built-in self-test using self-nonself differentiation. *IEEE Transactions on Evolutionary Computation*, 6(3):227–238, June 2002.

[5] F. S. C. Bolchini, R. Montandon and D. Sciuto. A state encoding for self-checking finite state machines. *Proceedings of the ASP-DAC*, pages 711–716, August 1995.

[6] K. Constantinides, J. Blome, S. Plaza, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. Bulletproof: A defect-tolerant CMP switch architecture. In *Proc. of the Int'l Symp. on High-Performance Computer Arch.*, Feb. 2006.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE Annual Workshop on Workload Characteristics*, pages 3–14, 2001.

[8] M. D. Hill and A. J. Smith. Evaluating associativity in CPU caches. *IEEE Trans. on Computers*, 38(12):1612–1630, 1989.

[9] N. Jha. Separable codes for detecting unidirectional errors. *IEEE Transaction on Computer-Aided Design*, 8, May 1990.

[10] A. J. KleinOsowski and D. J. Lilja. The NanoBox project: Exploring fabrics of self-correcting logic blocks for high defect rate molecular device technologies. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 19–24, 2004.

[11] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *Int'l Symposium on Computer Architecture*, pages 330–335, 1997.

[12] J. Lo. A hyper optimal encoding scheme for self-checking circuits. *IEEE Trans. Computers*, 45(9):1022–1030, Sept. 1996.

[13] M. Marouf and D. Friedman. Design of self-checking checkers for Berger codes. *Proc. Eighth Symp. Fault-Tolerant Computing*, pages 179–184, June 1978.

[14] M. Marouf and D. Friedman. Efficient design of self-checking checkers for m-out-of-n codes. *IEEE Trans. Computers*, 27:482–490, June 1978.

[15] J. F. Martinez, J. Renau, M. C. Huang, M. Prvulovic, , and J. Torrellas. Cherry: Checkpointed early resource recycling in out-of-order microprocessors. In *Proc. Int'l Symposium on Microarchitecture (MICRO)*, pages 3–14, 2002.

[16] D. G. Mavis and P. H. Eaton. Soft error rate mitigation techniques for modern microcircuits. In *Proceedings of 40th Annual Reliability Physics Symposium*, pages 216–225, 2002.

[17] N. Miskov-Zivanov and D. Marculescu. MARS-C: modeling and reduction of soft errors in combinational circuits. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 767–772, 2006.

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

[19] S. Mitra, M. Zhang, N. Seifert, B. Gill, S. Waqas, and K. S. Kim. Combinational logic soft error correction. In *International Test Conference*, November 2006.

[20] M. Nicolaidis. Time redundancy based soft-error tolerance to rescue nanometer technologies. In *Proc. VLSI Test Symposium*, pages 86–94, 1999.

[21] S. Shyam, K. Constantinides, S. Phadke, , V. Bertacco, and T. Austin. Ultra low-cost defect protection for microprocessor pipelines. In *Proc. of the Symp. on Architectural Support for Prog. Languages and Operating Systems (ASPLOS)*, Oct. 2006.

[22] D. P. Siewiorek and R. S. Swarz. Reliable computer systems: Design and evaluation, 3rd edition. *AK Peters, Ltd*, 1998.

[23] J. H. Stathis. Reliability limits for the gate insulator in CMOS technology. *IBM Journal of Research and Development*, 46(2/3):265–286, 2002.

[24] Trimaran. An infrastructure for research in ILP. http://www.trimaran.org, 2000.

[25] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design. In *Proc. of Int'l Conf. on Dependable Systems and Networks (DSN)*, pages 411–420, 2001.

[26] B. Zhang, W.-S. Wang, and M. Orshansky. FASER: Fast analysis of soft error susceptibility for cell-based designs. In *ISQED '06: Proceedings of the 7th International Symposium on Quality Electronic Design*, pages 755–760, 2006.