CLEAR: Cross-Layer Exploration for Architecting Resilience
Combining Hardware and Software Techniques to Tolerate Soft Errors in Processor Cores

Eric Cheng[1], Shahzrad Mirkhani[1], Lukasz G. Szafaryn[2], Chen-Yong Cher[3], Hyungmin Cho[1], Kevin Skadron[2], Mircea R. Stan[2], Klas Lilja[4], Jacob A. Abraham[5], Pradip Bose[3], Subhasish Mitra[1]

[1]Stanford University, [2]University of Virginia, [3]IBM Research, [4]Robust Chip, Inc., [5]University of Texas at Austin

Abstract
We present a first of its kind framework which overcomes a major challenge in the design of digital systems that are resilient to reliability failures: achieve desired resilience targets at minimal costs (energy, power, execution time, area) by combining resilience techniques across various layers of the system stack (circuit, logic, architecture, software, algorithm). This is also referred to as *cross-layer resilience*. In this paper, we focus on radiation-induced soft errors in processor cores. We address both single-event upsets (SEUs) and single-event multiple upsets (SEMUs) in terrestrial environments. Our framework automatically and systematically explores the large space of comprehensive resilience techniques and their combinations across various layers of the system stack (798 cross-layer combinations in this paper), derives cost-effective solutions that achieve resilience targets at minimal costs, and provides guidelines for the design of new resilience techniques. We demonstrate the practicality and effectiveness of our framework using two diverse designs: a simple, in-order processor core and a complex, out-of-order processor core. Our results demonstrate that a carefully optimized combination of circuit-level hardening, logic-level parity checking, and micro-architectural recovery provides a highly cost-effective soft error resilience solution for general-purpose processor cores. For example, a 50× improvement in silent data corruption rate is achieved at only 2.1% energy cost for an out-of-order core (6.1% for an in-order core) with no speed impact. However, selective circuit-level hardening alone, guided by a thorough analysis of the effects of soft errors on application benchmarks, provides a cost-effective soft error resilience solution as well (with ~1% additional energy cost for a 50× improvement in silent data corruption rate).

CCS Concepts: **• General and reference → Reliability**; **• Hardware → Fault tolerance**; Transient errors and upsets; **• Computer systems organization → Reliability**

Keywords: Cross-layer resilience; soft errors

1. Introduction
This paper addresses the *cross-layer resilience challenge* for designing robust digital systems: given a set of resilience techniques at various abstraction layers (circuit, logic, architecture, software, algorithm), how does one protect a given design from radiation-induced soft errors using (perhaps) a **combination** of these techniques, **across multiple abstraction layers**, such that overall soft error resilience targets are met at minimal costs (energy, power, execution time, area)? Specific soft error resilience targets addressed in this paper are: *Silent Data Corruption* (*SDC*), where an error causes the system to output an incorrect result without error indication; and, *Detected but Uncorrected Error* (*DUE*), where an error is detected (e.g., by a resilience technique or a system crash or hang) but is not recovered automatically without user intervention.

The need for *cross-layer resilience*, where multiple error resilience techniques from different layers of the system stack cooperate to achieve cost-effective error resilience, is articulated in several publications (e.g., [Borkar 05, Cappello 14, Carter 10, DeHon 10, Gupta 14, Henkel 14, Pedram 12]).

There are numerous publications on error resilience techniques, many of which span multiple abstraction layers. These publications mostly describe specific implementations. Examples include structural integrity checking [Lu 82] and its derivatives (mostly spanning architecture and software layers) or the combined use of circuit hardening, error detection (e.g., using logic parity checking and residue codes) and instruction-level retry [Ando 03, Meaney 05, Sinharoy 11] (spanning circuit, logic, and architecture layers). Cross-layer resilience implementations in commercial systems are often based on "designer experience" or "historical practice." There exists no comprehensive framework to systematically address the cross-layer resilience challenge. Creating such a framework is difficult. It must encompass the entire design flow end-to-end, from comprehensive and thorough analysis of various combinations of error resilience techniques all the way to layout-level implementations, such that one can (automatically) determine which resilience technique or combination of techniques (either at the same abstraction layer or across different abstraction layers) should be chosen. However, such a framework is essential in order to answer important cross-layer resilience questions such as:

1. Is cross-layer resilience the best approach for achieving a given resilience target at low cost?
2. Are all cross-layer solutions equally cost-effective? If not, which cross-layer solutions are the best?
3. How do cross-layer choices change depending on application-level energy, latency, and area constraints?
4. How can one create a cross-layer resilience solution that is cost-effective across a wide variety of application workloads?
5. Are there general guidelines for new error resilience techniques to be cost-effective?

We present CLEAR (Cross-Layer Exploration for Architecting Resilience), a first of its kind framework, which addresses the cross-layer resilience challenge. In this paper, we focus on the use of CLEAR for radiation-induced soft errors[1] in terrestrial environments.

Although the soft error rate of an SRAM cell or a flip-flop stays roughly constant or even decreases over technology generations, the system-level soft error rate increases with increased integration [Mitra 14, Seifert 10, 12]. Moreover, soft error rates can increase when lower supply voltages are used to improve energy efficiency [Mahatme 13, Pawlowski 14]. We focus on *flip-flop soft errors* because design techniques to protect them are generally expensive. Coding techniques are routinely used for protecting on-chip memories. Combinational logic circuits are significantly less susceptible to soft errors and do not pose a concern [Gill 09, Seifert 12]. We address both single-event upsets (*SEUs*) and single-event multiple upsets (*SEMUs*) [Lee 10, Pawlowski 14]. While CLEAR can address soft errors in various digital components of a complex System-on-a-Chip (including uncore components [Cho 15] and hardware accelerators), a detailed analysis of soft errors in all these components is beyond the scope of this paper. Hence, we focus on soft errors in processor cores.

To demonstrate the effectiveness and practicality of CLEAR, we explore 798 cross-layer combinations using ten representative error detection/correction techniques and four hardware error recovery techniques. These techniques span various layers of the system stack: circuit, logic, architecture, software, and algorithm (Fig. 1). Our extensive cross-layer exploration encompasses over 9 million flip-flop soft error injections into two diverse processor core architectures (Table 1): a simple

DAC '16, June 05-09, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06 $15.00
DOI: http://dx.doi.org/10.1145/2897937.2897996

[1] Other error sources (voltage noise, circuit-aging) may be incorporated into CLEAR, but are not the focus of this paper.

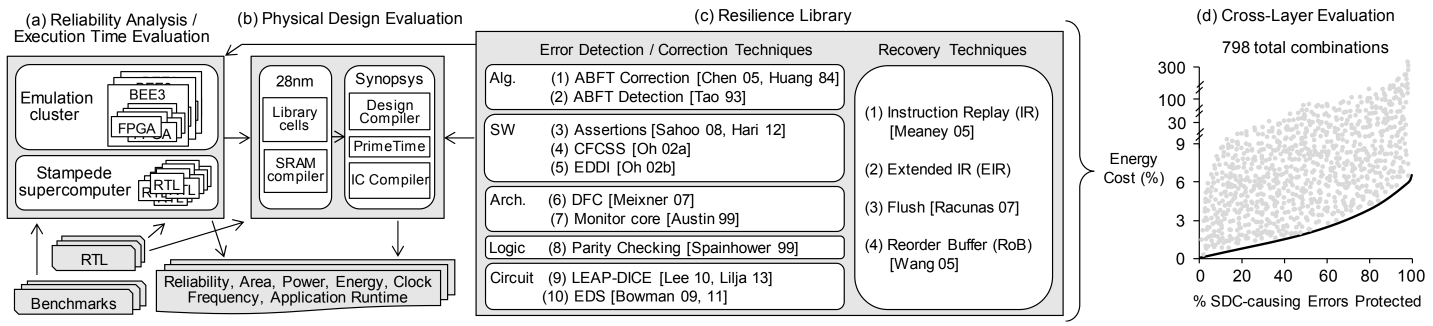

(a) Reliability Analysis / Execution Time Evaluation
Emulation cluster
BEE3
FPGA
Stampede supercomputer
RTL
RTL
Benchmarks
(b) Physical Design Evaluation
28nm
Library cells
SRAM compiler
Synopsys
Design Compiler
PrimeTime
IC Compiler
Reliability, Area, Power, Energy, Clock Frequency, Application Runtime
(c) Resilience Library
Error Detection / Correction Techniques
Alg. (1) ABFT Correction [Chen 05, Huang 84]
(2) ABFT Detection [Tao 93]
SW (3) Assertions [Sahoo 08, Hari 12]
(4) CFCSS [Oh 02a]
(5) EDDI [Oh 02b]
Arch. (6) DFC [Meixner 07]
(7) Monitor core [Austin 99]
Logic (8) Parity Checking [Spainhower 99]
Circuit (9) LEAP-DICE [Lee 10, Lilja 13]
(10) EDS [Bowman 09, 11]
Recovery Techniques
(1) Instruction Replay (IR) [Meaney 05]
(2) Extended IR (EIR)
(3) Flush [Racunas 07]
(4) Reorder Buffer (RoB) [Wang 05]
(d) Cross-Layer Evaluation
798 total combinations
Energy Cost (%)
300
100
30
9
6
3
0
0
20
40
60
80
100
% SDC-causing Errors Protected

Figure 1. CLEAR Framework: (a) BEE3 emulation cluster / Stampede supercomputer injects over 9 million errors into two diverse processor architectures running 18 full-length application benchmarks. (b) Accurate physical design evaluation accounts for resilience overheads. (c) Comprehensive resilience library consisting of ten error detection / correction techniques + four hardware error recovery techniques. (d) Example illustrating thorough exploration of 798 cross-layer combinations with varying energy costs vs. percentage of SDC-causing errors protected.

in-order SPARC Leon3 core (*InO-core*) and a complex super-scalar out-of-order Alpha IVM core (*OoO-core*), across 18 benchmarks: SPECINT2000 [Henning 00] and DARPA PERFECT [DARPA]. Such extensive exploration enables us to conclusively answer the above cross-layer resilience questions:

1. For a wide range of error resilience targets, optimized cross-layer combinations can provide low cost solutions for soft errors.
2. Not all cross-layer solutions are cost-effective.
   a. For general-purpose processor cores, a carefully optimized combination of selective circuit-level hardening, logic-level parity checking, and micro-architectural recovery provides a highly effective cross-layer resilience solution. For example, a 50× SDC improvement (defined in Sec. 2.1) is achieved at 2.1% and 6.1% energy costs for the OoO- and InO-cores, respectively. The use of selective circuit-level hardening and logic-level parity checking is guided by a thorough analysis of the effects of soft errors on application benchmarks.
   b. When the application space can be restricted to matrix operations, a cross-layer combination of Algorithm Based Fault Tolerance (ABFT) correction, selective circuit-level hardening, logic-level parity checking, and micro-architectural recovery can be highly effective. For example, a 50× SDC improvement is achieved at 1.9% and 3.1% energy costs for the OoO- and InO-cores, respectively. But, this approach may not be practical for general-purpose processor cores targeting general applications.
   c. Selective circuit-level hardening, guided by a thorough analysis of the effects of soft errors on application benchmarks, provides a highly effective soft error resilience approach. For example, a 50× SDC improvement is achieved at 3.1% and 7.3% energy costs for the OoO- and InO-cores, respectively.
3. The above conclusions about cost-effective soft error resilience techniques largely hold across various application characteristics (e.g., latency constraints despite errors in soft real-time applications).
4. Selective circuit-level hardening (and logic-level parity checking) techniques are guided by the analysis of the effects of soft errors on application benchmarks. Hence, one must address the challenge of potential mismatch between application benchmarks vs. applications in the field, especially when targeting high degrees of resilience (e.g., 10× or more SDC improvement). We overcome this challenge using various flavors of circuit-level hardening techniques (details in Sec. 4).
5. Cost-effective resilience approaches discussed above provide bounds that new soft error resilience techniques must achieve to be competitive. It is however crucial that the benefits and costs of new techniques are evaluated thoroughly and correctly before publication.

Table 1. Processor designs studied.

Core	Design	Description	Clk. Freq.	Error Injections	Instructions Per Cycle
InO	Leon3 [Leon]	Simple, in-order (1,250 flip-flops)	2.0 GHz	5.9 million	0.4
OoO	IVM [Wang 04]	Complex, super-scalar, out-of-order (13,819 flip-flops)	600 MHz	3.5 million	1.3

[2] 11 SPEC / 7 PERFECT benchmarks for InO-cores and 8 SPEC / 3 PERFECT for OoO-cores (missing benchmarks contain floating-point instructions not executable by the OoO-core RTL model).

2. CLEAR Framework

Figure 1 gives an overview of the CLEAR framework. Individual components of the framework are discussed below.

2.1 Reliability Analysis

CLEAR is not merely an error rate projection tool; rather, reliability analysis is a component of the overall CLEAR framework.

We use flip-flop soft error injections for reliability analysis with respect to radiation-induced soft errors. This is because radiation test results confirm that injection of single bit-flips into flip-flops closely models soft error behaviors in actual systems [Bottoni 14, Sanda 08]. Furthermore, flip-flop-level error injection is crucial since naïve high-level error injections can be highly inaccurate [Cho 13]. For individual flip-flops, both SEUs and SEMUs manifest as single-bit errors. Our SEMU-tolerant circuit hardening and our layout implementations ensure this assumption holds for both the baseline and resilient designs.

We injected over 9 million flip-flop soft errors into the RTL of the processor designs using three BEE3 FPGA emulation systems and also using mixed-mode simulations on the Stampede supercomputer (University of Texas at Austin) (similar to [Cho 13, Davis 09, Ramachandran 08, Wang 04]). This ensures that error injection results have less than a 0.1% margin of error with a 95% confidence interval per benchmark. Errors are injected uniformly into all flip-flops and application regions, to mimic real world scenarios.

The SPECINT2000 [Henning 00] and DARPA PERFECT [DARPA] benchmark suites are used for evaluation[2]. The PERFECT suite complements SPEC by adding applications targeting signal and image processing domains. We chose the SPEC workloads since the original publications corresponding to the resilience techniques used them for evaluation. We ran benchmarks in their entirety.

Flip-flop soft errors can result in the following outcomes [Cho 13, Michalak 12, Sanda 08, Wang 04, 07]: **Vanished -** normal termination and output files match error-free runs, **Output Mismatch (OMM) -** normal termination, but output files are different from error-free runs, **Unexpected Termination (UT) -** program terminates abnormally, **Hang -** no termination or output within 2× the nominal execution time, **Error Detection (ED) -** an employed resilience technique flags an error, but the error is not recovered using a hardware recovery mechanism.

Using the above outcomes, any error that results in OMM causes SDC. Any error that results in UT, Hang, or ED causes DUE. Note that, there are no ED outcomes if no error detection technique is employed. The resiliency of a protected (new) design compared to an unprotected (original, baseline) design can be defined in terms of *SDC improvement* (Eq. 1a) or *DUE improvement* (Eq. 1b). The susceptibility of flip-flops to soft errors is assumed to be uniform across all flip-flops in the design (but this parameter is adjustable in our framework). Techniques that increase runtime or add flip-flops increase the susceptibility of the design to soft-errors. To accurately account for this situation, we calculate, based on [Schirmeier 15], a correction factor γ (where $\gamma \geq 1$), which is applied to

Table 2. Individual resilience techniques: costs and improvements when implemented as a standalone solution.

Layer	Technique		Area Cost	Power Cost	Energy Cost	Exec. Time Impact	Avg. SDC Improvement	Avg. DUE Improvement	False Positive
Circuit[4]	LEAP-DICE (no additional recovery needed)	InO	0-9.3%	0-22.4%	0-22.4%	0%	1× - 5,000×	1× - 5,000×	0%
		OoO	0-6.5%	0-9.4%	0-9.4%				
	EDS (without recovery - unconstrained)	InO	0-10.7%	0-22.9%	0-22.9%	0%	1× - 100,000×	0.1× - 1×	0%
		OoO	0-12.2%	0-11.5%	0-11.5%				
	EDS (with IR recovery)	InO	0-16.7%	0-43.9%	0-43.9%	0%	1× - 100,000×	1× - 100,000×	0%
		OoO	0-12.3%	0-11.6%	0-11.6%				
Logic[4]	Parity (without recovery - unconstrained)	InO	0-10.9%	0-23.1%	0-23.1%	0%	1× - 100,000×	0.1× - 1×	0%
		OoO	0-14.1%	0-13.6%	0-13.6%				
	Parity (with IR recovery)	InO	0-26.9%	0-44%	0-44%	0%	1× - 100,000×	1× - 100,000×	0%
		OoO	0-14.2%	0-13.7%	0-13.7%				
Arch.	DFC (without recovery - unconstrained)	InO	3%	1%	7.3%	6.2%	1.2×	0.5×	0%
		OoO	0.2%	0.1%	7.2%	7.1%			
	DFC (with EIR recovery)	InO	37%	33%	41.2%	6.2%	1.2×	1.4×	0%
		OoO	0.4%	0.2%	7.3%	7.1%			
	Monitor Core (with RoB recovery)	OoO	9%	16.3%	16.3%	0%	19×	15×	0%
Soft-ware[5]	Software Assertions for general-purpose processors (without recovery - unconstrained)	InO	0%	0%	15.6%	15.6%[6]	1.5×[7]	0.6×	0.003%
	CFCSS (without recovery - unconstrained)	InO	0%	0%	40.6%	40.6%	1.5×	0.5×	0%
	EDDI (without recovery - unconstrained)	InO	0%	0%	110%	110%	37.8×[8]	0.3×	0%
Alg.	ABFT correction (no additional recovery needed)	InO OoO	0%	0%	1.4%	1.4%	4.3×	1.2×	0%
	ABFT detection (without recovery - unconstrained)	InO OoO	0%	0%	24%	1-56.9%[9]	3.5×	0.5×	0%

ensure a fair and accurate comparison for all techniques[3] (additional details in [Cheng 16]).

$$SDC\ improvement = \frac{(original\ OMM\ count)}{(new\ OMM\ count)} \times \gamma^{-1} \quad \text{(Eq. 1a)}$$

$$DUE\ improvement = \frac{(original\ (UT{+}Hang)\ count)}{(new\ (UT{+}Hang{+}ED)\ count)} \times \gamma^{-1} \quad \text{(Eq. 1b)}$$

Reporting SDC and DUE improvements allows our results to be agnostic to absolute error rates. Although we have described the use of error injection-driven reliability analysis, the modular nature of CLEAR allows us to swap in other approaches as appropriate (e.g., our error-injection analysis could be substituted with techniques like [Mirkhani 15b], once they are properly validated).

2.2 Execution Time Evaluation

Execution time is estimated using FPGA emulation and RTL simulation. Applications are run to completion to accurately capture the execution time of an unprotected design. We also report the error-free execution time impact associated with resilience techniques at the architecture, software, and algorithm levels. For resilience techniques at the circuit and logic levels, our design methodology maintains the same clock speed as the unprotected design.

2.3 Physical Design Evaluation

We used Synopsys design tools (Design Compiler, IC compiler, and Primetime) [Synopsys] with a commercial 28nm technology library (with corresponding SRAM compiler) to perform synthesis, place-and-route, and power analysis. *Synthesis and place-and-route* (*SP&R*) was run for all configurations of the design (before and after adding resilience techniques) to ensure all constraints of the original design (e.g., timing, DRC) were met for the resilient designs. To account for tool artifacts (e.g., due to variations in RTL or optimization heuristics), separate resilient designs were generated based on error injection results for each individual application benchmark. SP&R was performed for each of these designs and then averaged to minimize these artifacts. Layouts were also carefully generated to mitigate the impact of SEMUs (Sec. 2.4).

2.4 Resilience Library

We carefully chose ten error detection and correction techniques together with four hardware error recovery techniques. These techniques largely cover the space of existing soft error resilience techniques (explained in [Cheng 16]). The costs incurred by each resilience technique (and corresponding resilience improvements) when used as a *standalone solution* (e.g., an error detection / correction technique by itself or, optionally, in conjunction with a recovery technique) are presented in Table 2.

Circuit: The *hardened flip-flops* (LEAP-DICE, Light Hardened LEAP) in Table 3 are designed to tolerate SEUs and SEMUs [Lee 10, Lilja 13] (detailed explanation in [Cheng 16]). *Error Detection Sequential* (*EDS*) [Bowman 09, 11] can be used to detect flip-flop soft errors (in addition to timing errors).

Table 3. Resilient flip-flops.

Type	Soft Error Rate	Area	Power	Delay	Energy
Baseline	1	1	1	1	1
Light Hardened LEAP (LHL)	2.5×10^{-1}	1.2	1.1	1.2	1.3
LEAP-DICE	2.0×10^{-4}	2.0	1.8	1	1.8
EDS[10]	~100% detect	1.5	1.4	1	1.4

Logic: *Parity checking* provides error detection by checking flip-flop inputs and outputs [Spainhower 99]. Our design heuristics (detailed in [Cheng 16]) reduce the cost of parity while also ensuring that clock frequency is maintained as in the original design (by varying the number of flip-flops checked together, grouping flip-flops by timing slack, pipelining parity checker logic, etc.) We minimize SEMUs through layouts that ensure a minimum spacing (the size of one flip-flop) between flip-flops checked by the same parity checker [Amusan 09].

Architecture: Our implementation of *Data Flow Checking* (*DFC*) includes *Control Flow Checking (CFC)*, and resembles [Meixner 07] (details in [Cheng 16]). *Monitor cores* are specialized checker cores that

[3] Research literature commonly considers γ=1. We report results using true γ values, but our conclusions hold for γ=1 as well (latter is optimistic).

[4] Circuit and logic techniques have tunable costs/resilience (e.g., for InO-cores, 5× SDC improvement using LEAP-DICE is achieved at 4.3% energy cost while 50× SDC improvement is achieved at 7.3% energy cost). This is achievable through selective insertion guided by error injection using application benchmarks.

[5] Software techniques are generated for InO-cores only since the LLVM compiler no longer supports the Alpha architecture.

[6] Some software assertions for general-purpose processors (e.g., [Sahoo 08]) suffer from false positives (i.e., an error is reported during an error-free run). The execution time impact reported discounts the impact of false positives.

[7] Improvements differ from previous publications that injected errors into architectural registers. [Cho 13] demonstrated such injections can be highly inaccurate; we used highly accurate flip-flop-level error injections.

[8] We report results for EDDI with store-readback [Lin 14]. Without this enhancement, EDDI provides 3.3× SDC / 0.4× DUE improvement.

[9] Execution time impact for ABFT detection can be high since error detection checks may require computationally-expensive calculations.

[10] For EDS, the costs are listed for the flip-flop only. Error signal routing and delay buffers (included in Table 2) increase overall cost [Bowman 11].

validate instructions executed by the main core (e.g., [Austin 99, Lu 82]). We analyze monitors similar to [Austin 99]. For InO-cores, the size of the monitor core is of the same order as the main core, and hence, excluded from our study. For OoO-cores, the simpler monitor core can have lower throughput compared to the main core and thus stall the main core. We confirm (via IPC estimation) that our monitor core implementation does not stall the main core.

Software: *Software Assertions for General-Purpose Processors* check program variables to ensure that their values are valid. We combine assertions from [Hari 12, Sahoo 08]. *Control Flow Checking by Software Signatures* (*CFCSS*) [Oh 02a] and *Error Detection by Duplicated Instructions* (*EDDI*) [Oh 2b] are implemented via compiler modification. We utilize EDDI with store-readback [Lin 14] to maximize coverage by ensuring that values are written correctly.

Algorithm: *Algorithm Based Fault Tolerance (ABFT)* can detect (*ABFT detection*) or detect and correct errors (*ABFT correction*) through algorithm modifications [Bosilca 09, Chen 05, Huang 84, Nair 90]. (additional details in [Cheng 16]).

Recovery: We consider two recovery scenarios (details in [Cheng 16]): *bounded latency*, i.e., an error must be recovered within a fixed period of time after its occurrence, and *unconstrained*, i.e., where no latency constraints exist and errors are recovered externally once detected (i.e., no hardware recovery is required). Bounded latency recovery is achieved using one of the following hardware recovery techniques (Table 4): *flush* or *Reorder Buffer* (*RoB*) recovery (both of which rely on flushing non-committed instructions followed by re-execution) [Racunas 07, Wang 05]; *instruction replay* (*IR*) or *extended instruction replay* (*EIR*) recovery (both of which rely on instruction checkpointing to rollback and replay instructions) [Meaney 05]. EIR is an extension of IR with additional buffers required by DFC for recovery. Flush and RoB are unable to recover from errors detected after the memory write stage of InO-cores or after the reorder buffer of OoO-cores, respectively (these errors will have propagated to architecture visible states). Hence, LEAP-DICE is used to protect flip-flops in these pipeline stages when using flush/RoB recovery.

Table 4. Hardware error recovery costs.

Core	Type	Area	Power	Energy	Recovery Latency
InO	Instruction Replay (IR) Recovery	16%	21%	21%	47 cycles
	EIR Recovery	34%	32%	32%	47 cycles
	Flush Recovery	0.6%	0.9%	1.8%	7 cycles
OoO	Instruction Replay (IR) Recovery	0.1%	0.1%	0.1%	104 cycles
	EIR Recovery	0.2%	0.1%	0.1%	104 cycles
	Reorder Buffer (ROB) Recovery	0.01%	0.01%	0.01%	64 cycles

3. Cross-Layer Combinations

CLEAR uses a top-down approach to explore the cost-effectiveness of various cross-layer combinations. For example, resilience techniques at the upper layers of the system stack (e.g., ABFT correction) are applied before incrementally moving down the stack to apply techniques from lower layers (e.g., an optimized combination of logic parity checking, circuit-level LEAP-DICE, and micro-architectural recovery). This approach (example shown in Fig. 2) ensures that resilience techniques from various layers of the stack effectively interact with one another. Resilience techniques from the algorithm, software, and architecture layers of the stack generally protect multiple flip-flops (determined using error injection); however, a designer typically has little control over the specific subset protected. Using multiple resilience techniques from these layers can lead to situations where a given flip-flop may be protected (sometimes unnecessarily) by multiple techniques. At the logic and circuit layers, fine-grained protection is available since these techniques can be applied selectively to individual flip-flops (those not sufficiently protected by higher-level techniques).

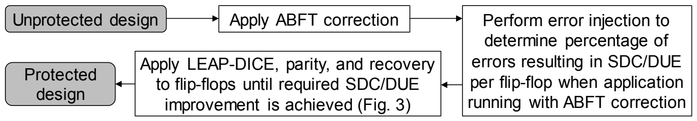

Unprotected design
Apply ABFT correction
Perform error injection to determine percentage of errors resulting in SDC/DUE per flip-flop when application running with ABFT correction
Apply LEAP-DICE, parity, and recovery to flip-flops until required SDC/DUE improvement is achieved (Fig. 3)
Protected design

Figure 2. Cross-layer methodology example for combining ABFT correction, LEAP-DICE, logic parity, and micro-architectural recovery.

Among the 798 cross-layer combinations explored using CLEAR, a highly promising approach combines selective circuit-level hardening using LEAP-DICE, logic parity, and micro-architectural recovery (flush recovery for InO-cores, RoB recovery for OoO-cores). Thorough error injection using application benchmarks plays a critical role in selecting the flip-flops protected using these techniques. Figure 3 and Heuristic 1 detail the methodology for creating this combination. If recovery is not needed (e.g., for unconstrained recovery), the "Harden" procedure in Heuristic 1 can be modified to always return false.

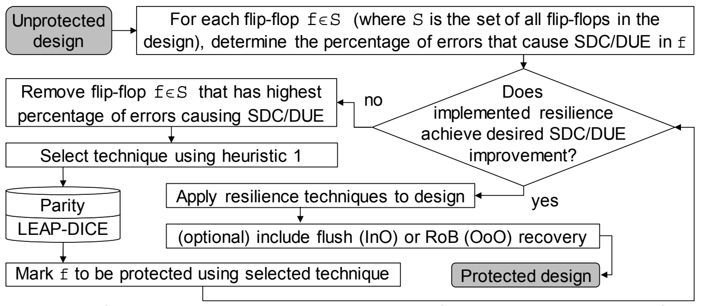
For example, to achieve a 50× SDC improvement, the combination of LEAP-DICE, logic parity, and micro-architectural recovery provides a 1.5× and 1.2× energy savings for the OoO- and InO-cores, respectively, compared to selective circuit hardening using LEAP-DICE (Table 5). The relative benefits are consistent across benchmarks and over the range of SDC/DUE improvements. The overheads in Table 5 are small because we reported the most energy-efficient resilience solutions. Most of the 798 combinations are far costlier. More detailed results (e.g., inclusion of EDS) appear in [Cheng 16].

Table 5[11]. Costs vs. SDC and DUE improvements for efficient resilience techniques.
A (area cost %), P (power cost %), E (energy cost %)

			Bounded Latency Recovery										Unconstrained Recovery[12]										Exec. Time Impact
			SDC Improvement					DUE Improvement					SDC Improvement					DUE Improvement					
			2	5	50	500	max	2	5	50	500	max	2	5	50	500	max	2	5	50	500	max	
InO	Selective hardening using LEAP-DICE	A	0.8	1.8	2.9	3.3	9.3	0.7	1.7	3.8	5.1	9.3	0.8	1.8	2.9	3.3	9.3	0.7	1.7	3.8	5.1	9.3	0%
		P	2	4.3	7.3	8.2	22.4	1.5	3.8	9.5	12.5	22.4	2	4.3	7.3	8.2	22.4	1.5	3.8	9.5	12.5	22.4	
		E	2	4.3	7.3	8.2	22.4	1.5	3.8	9.5	12.5	22.4	2	4.3	7.3	8.2	22.4	1.5	3.8	9.5	12.5	22.4	
	LEAP-DICE + logic parity (+ flush recovery)	A	0.7	1.7	2.5	3	8	0.6	1.5	3.6	4.4	8	0.7	1.6	2.4	2.8	7.6	-	-	-	-	-	0%
		P	1.9	3.9	6.1	6.7	17.9	1.5	3.4	8.4	10.4	17.9	1.9	3.8	5.9	6.5	17.2						
		E	1.9	3.9	6.1	6.7	17.9	1.5	3.4	8.4	10.4	17.9	1.9	3.8	5.9	6.5	17.2						
	ABFT correction + LEAP-DICE + logic parity (+ flush recovery)	A	0	0.4	1.0	1.2	8	0.3	0.4	1.5	2.7	8	0	0.4	0.9	1.1	7.6	-	-	-	-	-	1.4%
		P	0	0.7	1.7	1.8	17.9	1	1	3.3	5.7	17.9	0	0.7	1.6	1.8	17.2						
		E	1.4	2.2	3.1	3.2	19.6	2.4	2.4	4.8	7.2	19.6	1.4	2.2	3	3.2	18.8						
OoO	Selective hardening using LEAP-DICE	A	1.1	1.3	2.2	2.4	6.5	1.3	1.6	3.1	3.6	6.5	1.1	1.3	2.2	2.4	6.5	1.3	1.6	3.1	3.6	6.5	0%
		P	1.5	1.7	3.1	3.5	9.4	2	2.3	4.2	5.1	9.4	1.5	1.7	3.1	3.5	9.4	2	2.3	4.2	5.1	9.4	
		E	1.5	1.7	3.1	3.5	9.4	2	2.3	4.2	5.1	9.4	1.5	1.7	3.1	3.5	9.4	2	2.3	4.2	5.1	9.4	
	LEAP-DICE + logic parity (+ ROB recovery)	A	0.06	0.1	1.4	2.2	4.9	0.5	0.7	2.6	3	4.9	0.06	0.1	1.4	2.2	4.9	-	-	-	-	-	0%
		P	0.1	0.2	2.1	2.4	7	0.1	0.1	2	1.8	7	0.1	0.2	2.1	2.4	7						
		E	0.1	0.2	2.1	2.4	7	0.1	0.1	2	1.8	7	0.1	0.2	2.1	2.4	7						
	ABFT correction + LEAP-DICE + logic parity (+ ROB recovery)	A	0	0.01	0.3	0.5	4.9	0.4	0.6	2.1	3	4.9	0	0.01	0.3	0.5	4.8	-	-	-	-	-	1.4%
		P	0	0.01	0.5	0.8	7	0.1	0.1	3	1.6	7	0	0.01	0.5	0.8	6.9						
		E	1.4	1.5	1.9	2.2	8.5	1.5	1.5	4.2	3	8.5	1.4	1.5	1.9	2.2	8.4						

[11] Costs are generated per benchmark. We report the average cost over all benchmarks. Relative standard deviation is 0.6-3.1%.

[12] DUE improvements are not possible when detection-only techniques are used in an unconstrained recovery scenario.



Unprotected design
For each flip-flop f∈S (where S is the set of all flip-flops in the design), determine the percentage of errors that cause SDC/DUE in f
Remove flip-flop f∈S that has highest percentage of errors causing SDC/DUE
no
Does implemented resilience achieve desired SDC/DUE improvement?
Select technique using heuristic 1
Parity
LEAP-DICE
Apply resilience techniques to design
yes
(optional) include flush (InO) or RoB (OoO) recovery
Mark f to be protected using selected technique
Protected design

Figure 3. Cross-layer resilience methodology for combining LEAP-DICE, parity, and micro-architectural recovery.

Heuristic 1: Choose LEAP-DICE or parity technique
Input: f: flip-flop to be protected
Output: Technique to apply to f (LEAP-DICE, parity)

```
 1: if HARDEN(f) then return LEAP-DICE
 2: if PARITY(f) then return parity
 3: return LEAP-DICE

 4: procedure HARDEN(f)
 5:     if an error in f cannot be flushed (i.e., f is in the memory,
            exception, writeback stages of InO or after the RoB of OoO)
 6:     then return TRUE; else return FALSE
 7: end procedure

 8: procedure PARITY(f)
 9:     if f has timing path slack greater than delay imposed by 32-bit
            XOR-tree (this implements low cost parity checking as
            explained in [Cheng 16])
10:     then return TRUE, else return FALSE
11: end procedure
```

When the application space targets specific algorithms (e.g., matrix operations), a cross-layer combination of LEAP-DICE, parity, ABFT correction, and micro-architectural error recovery (flush/RoB) provides additional energy savings (Table 5). Since ABFT correction performs in-place error correction, no separate recovery mechanism is required for ABFT correction. For our study, we could apply ABFT correction to three of our PERFECT benchmarks: `2d_convolution`, `debayer_filter`, and `inner_product`. When targeting DUE improvement, including ABFT correction provides no energy savings for the OoO-core. This is because ABFT correction performs checks at set locations in the program. For example, a DUE resulting from an invalid pointer access can cause an immediate program termination before a check is invoked.

Since most applications are not amenable to ABFT correction, the flip-flops protected by ABFT correction must also be protected by techniques such as LEAP-DICE or parity (or combinations thereof) for processors targeting general-purpose applications. This requires circuit hardening techniques (e.g., [Mitra 05, Zhang 06]) with the ability to selectively operate in an error-resilient mode (high resilience, high energy) when ABFT is unavailable or in an economy mode (low resilience, low power mode) when ABFT is available. However, the overheads outweigh ABFT correction benefits (details in [Cheng 16]).

4. Application Benchmark Dependence

The most cost-effective resilience techniques rely on selective circuit hardening / parity checking guided by error injection using application benchmarks. This raises the question: what happens when the applications in the field do not match application benchmarks? We refer to this situation as *application benchmark dependence*.

To quantify this dependence, we randomly selected 4 (of 11) SPEC benchmarks as a *training set*, and used the remaining 7 as a *validation set*. Resilience is implemented using the training set and the resulting design's resilience is determined using the validation set. We used 50 training/validation pairs. Table 6 indicates that validated SDC improvement is generally underestimated. Fortunately, when targeting <10× SDC improvement, the underestimation is <4%. This is due to the fact that the most vulnerable 10% of flip-flops (i.e., the flip-flops that result in the most SDCs or DUEs) are consistent across benchmarks. Since the number of errors resulting in SDC or DUE is not uniformly distributed among flip-flops, protecting these 10% of flip-flops results in the ~10× SDC improvement regardless of the benchmark. The vulnerabilities of the remaining 90% of flip-flops are more benchmark-dependent. The sensitivity may be minimized by training using additional benchmarks or through better benchmarks (e.g., [Mirkhani 15a]). An alternative approach is to apply our CLEAR framework using available benchmarks, and then replace all remaining unprotected flip-flops using LHL (Table 3, [Cheng 16]). This enables our resilient designs to meet (or exceed) resilience targets at <1.2% additional cost for SDC improvements >10×. (DUE results in [Cheng 16]).

Table 6. SDC improvement, cost before and after applying LHL to otherwise unprotected flip-flops.

Core	SDC Improvement			Cost Before LHL Insertion		Cost After LHL Insertion	
	Train	Validate	After LHL	Area	Power / Energy	Area	Power / Energy
InO	5×	4.8×	19.3×	1.6%	3.6%	3.1%	5.7%
	50×	38.9×	152.3×	2.4%	5.7%	3.3%	6.9%
	500×	433.1×	1,326.1×	2.9%	6.3%	3.4%	7.1%
	Max	5,568.9×	5,568.9×	8%	17.9%	8%	17.9%
OoO	5×	4.8×	35.1×	0.1%	0.2%	0.9%	1.8%
	50×	32.1×	204.3×	1.4%	2.1%	1.9%	2.7%
	500×	301.4×	1084.1×	2.2%	2.4%	2.4%	2.8%
	Max	6,625.8×	6,625.8×	4.9%	7%	4.9%	7%

5. Conclusions

CLEAR is a first of its kind cross-layer resilience framework that enables effective exploration of a wide variety of resilience techniques and their combinations across several layers of the system stack. Extensive cross-layer resilience studies using CLEAR demonstrate:

1. A <u>carefully optimized</u> combination of selective circuit-level hardening, logic-level parity checking, and micro-architectural recovery provides a highly cost-effective soft error resilience solution for general-purpose processors.
2. Selective circuit-level hardening alone, guided by thorough analysis of the effects of soft errors on application benchmarks, also provides a cost-effective soft error resilience solution (with ~1% additional energy cost for a 50× SDC improvement compared to the above approach).
3. Algorithm Based Fault Tolerance (ABFT) correction combined with selective circuit-level hardening (and logic-level parity checking and micro-architectural recovery) can further improve soft error resilience costs. However, existing ABFT correction techniques can be only be used for a few applications; this limits the applicability of this approach in the context of general-purpose processors.
4. Based on our analysis, we can derive bounds on energy costs vs. degree of resilience (SDC or DUE improvements) that new soft error resilience techniques must achieve to be competitive (shown in Fig. 4).
5. It is crucial that the benefits and costs of new resilience techniques are evaluated thoroughly and correctly before publication. Detailed analysis (e.g., flip-flop-level error injection or layout-level cost quantification) identifies hidden weaknesses that are often overlooked.

While this paper focuses on soft errors in processor cores, cross-layer resilience solutions for accelerators and uncore components as well as other error sources (e.g., voltage noise) may have different tradeoffs and may require additional modeling and analysis capabilities.

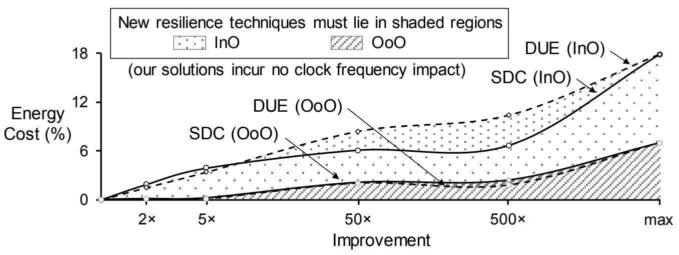


Figure 4. New resilience techniques must have cost and improvement tradeoffs that lie within the shaded regions bounded by LEAP-DICE + parity + micro-architectural recovery.

6. Acknowledgment

This research is supported in part by DARPA MTO (contract no. HR0011-13-C-0022), DTRA, NSF, and SRC. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). We thank Prof. S. V. Adve (UIUC), Prof. M. Alioto (NUS), Dr. K. A. Bowman (Qualcomm), Dr. V. Chandra (ARM), Dr. B. Cline (ARM), Dr. S. Das (ARM), Dr. S. K. S. Hari (NVIDIA), Prof. J. C. Hoe (CMU), Prof. A. B. Kahng (UCSD), Dr. J. Kao (TSMC), Dr. D. Lin (Stanford), Prof. I. L. Markov (U. Michigan), Dr. H. Naeimi (Intel), Dr. A. S. Oates (TSMC), Prof. K. Pattabiraman (UBC), and Dr. S. K. Sahoo (UIUC).

7. References

[Amusan 09] Amusan, O. A., "Effects of single-event-induced charge sharing in sub-100nm bulk CMOS technologies," *Vanderbilt University Dissertation*, 2009.

[Ando 03] Ando, H., *et al.,* "A 1.3-GHz fifth-generation SPARC64 microprocessor," *IEEE Journal Solid-State Circuits,* 2003.

[Austin 99] Austin, T. M., "DIVA: a reliable substrate for deep submicron microarchitecture design," *IEEE/ACM Intl. Symp. Microarchitecture*, 1999.

[Borkar 05] Borkar, S., "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, 2005.

[Bosilca 09] Bosilca, G., et *al.*, "Algorithmic based fault tolerance applied to high performance computing," *Journal of Parallel and Distributed Computing*, 2009.

[Bottoni 14] Bottoni, C., *et al.*, "Heavy ions test result on a 65nm sparc-v8 radiation-hard microprocessor," *IEEE Intl. Reliability Physics Symp.*, 2014.

[Bowman 09] Bowman, K. A., *et al.*, "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *IEEE Journal Solid-State Circuits*, 2009.

[Bowman 11] Bowman, K. A., *et al.*, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE Journal Solid-State Circuits*, 2011.

[Cappello 14] Cappello, F., *et al.*, "Toward exascale resilience: 2014 update," *Supercomputing Frontiers and Innovations*, 2014.

[Carter 10] Carter, N. P., H. Naeimi, and D. S. Gardner, "Design techniques for cross-layer resilience," *Design, Automation, & Test in Europe*, 2010.

[Chen 05] Chen, Z. and J. Dongarra, "Numerically stable real number codes based on random matrices," *Lecture Notes in Computer Science*, 2005.

[Cheng 16] Cheng, E., *et al.*, "CLEAR: cross-layer exploration for architecting resilience - combining hardware and software techniques to tolerate soft errors in processor cores," arXiv: [cs.OH], 2016.

[Cho 13] Cho, H., *et al.*, "Quantitative evaluation of soft error injection techniques for robust system design," *ACM/EDAC/IEEE Design Automation Conf.*, 2013.

[Cho 15] Cho, H., *et al.*, "Understanding soft errors in uncore components," *ACM/EDAC/IEEE Design Automation Conf.*, 2015.

[DARPA] DARPA PERFECT benchmark suite, http://hpc.pnl.gov/PERFECT.

[Davis 09] Davis, J. D., C. P. Thacker, and C. Chang, "BEE3: revitalizing computer architecture research," *Microsoft Tech. Rep.*, 2009.

[DeHon 10] DeHon, A., H. M. Quinn, and N. P. Carter, "Vision for cross-layer optimization to address the dual challenges of energy and reliability," *Design, Automation, & Test in Europe*, 2010.

[Gill 09] Gill, B., N. Seifert, and V. Zia, "Comparison of alpha-particle and neutron-induced combinational and sequential logic error rates at the 32nm technology node," *IEEE Intl. Reliability Physics Symp.*, 2009.

[Gupta 14] Gupta, M. S., *et al.*, "Cross-layer system resilience at affordable power," *IEEE Intl. Reliability Physics Symp.*, 2014.

[Hari 12] Hari S. K. S., S. V. Adve, and H. Naeimi, "Low-cost program-level detectors for reducing silent data corruptions," *IEEE/IFIP Intl. Conf. Dependable Systems & Networks*, 2012.

[Henkel 14] Henkel, J., *et al.*, "Multi-Layer dependability: from microarchitecture to application level," *ACM/EDAC/IEEE Design Automation Conf.*, 2014.

[Henning 00] Henning, J. L., "SPEC CPU2000: measuring CPU performance in the new millennium," *IEEE Computer*, 2000.

[Huang 84] Huang, K.-H. and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Computers*, 1984.

[Lee 10] Lee, H.-H. K., *et al.*, "LEAP: layout design through error-aware transistor positioning for soft-error resilient sequential cell design," *IEEE Intl. Reliability Physics Symp.*, 2010.

[Leon] Aeroflex G., "Leon3 processor," http://www.gaisler.com.

[Lilja 13] Lilja, K., *et al.*, "Single-event performance and layout optimization of flip-flops in a 28-nm bulk technology," *IEEE Trans. Nuclear Science*, 2013.

[Lin 14] Lin, D., *et al.*, "Effective post-silicon validation of system-on-chips using quick error detection," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* 2014.

[Lu 82] Lu, D. J., "Watchdog processor and structural integrity checking," *IEEE Trans. Computers*, 1982.

[Mahatme 13] Mahatme, N. N., *et al.*, "Impact of supply voltage and frequency on the soft error rates of logic circuits," *IEEE Trans. Nuclear Science*, 2013.

[Meaney 05] Meaney, P. J., *et al.*, "IBM z990 soft error detection and recovery," *IEEE Trans. Device and Materials Reliability*, 2005.

[Meixner 07] Meixner, A., M. E. Bauer, and D. J. Sorin, "Argus: low-cost comprehensive error detection in simple cores," *IEEE/ACM Intl. Symp. Microarchitecture*, 2007.

[Michalak 12] Michalak, S. E., *et al.*, "Assessment of the impact of cosmic-ray-induced neutrons on hardware in the roadrunner supercomputer," *IEEE Trans. Device and Materials Reliability*, 2012.

[Mirkhani 15a] Mirkhani, S., B. Samynathan, and J. A. Abraham, "Detailed soft error vulnerability analysis using synthetic benchmarks," *IEEE VLSI Test Symp.*, 2015.

[Mirkhani 15b] Mirkhani, S., *et al.*, "Efficient soft error vulnerability estimation of complex designs," *Design, Automation, & Test in Europe*, 2015.

[Mitra 05] Mitra S., *et al.*, "Robust system design with built-in soft error resilience," *IEEE Computer*, 2005.

[Mitra 14] Mitra, S., *et al.*, "The resilience wall: cross-layer solution strategies," *Intl. Symp. VLSI Technology, Systems and Applications*, 2014.

[Nair 90] Nair, V. S. S. and J. Abraham, "Real-number codes for fault-tolerant matrix operations on processor arrays," *IEEE Trans. Computers*, 1990.

[Oh 02a] Oh, N., P. P. Shirvani, and E. J. McCluskey, "Control flow checking by software signatures," *IEEE Trans. Reliability*, 2002.

[Oh 02b] Oh, N., P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Trans. Reliability*, 2002.

[Pawlowski 14] Pawlowski, R., *et al.*, "Characterization of radiation-induced SRAM and logic soft errors from 0.33V to 1.0V in 65 nm CMOS," *IEEE Custom Integrated Circuits Conf.*, 2014.

[Pedram 12] Pedram, M., *et al.*, "Report for the NSF workshop on cross-layer power optimization and management," *NSF*, 2012.

[Racunas 07] Racunas P., *et al.*, "Perturbation-based fault screening," *IEEE Intl. Symp. High Performance Computer Architecture*, 2007.

[Ramachandran 08] Ramachandran, P., *et al.*, "Statistical fault injection," *IEEE/IFIP Intl. Conf. Dependable Systems & Networks*, 2008.

[Sahoo 08] Sahoo S. K., *et al.*, "Using likely program invariants to detect hardware errors," *IEEE/IFIP Intl. Conf. Dependable Systems & Networks*, 2008.

[Sanda 08] Sanda, P. N., *et al.*, "Soft-error resilience of the IBM POWER6 processor," *IBM Journal of Research and Development*, 2008.

[Schirmeier 15] Schirmeier, H., C. Borchert, and O. Spinczyk, "Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors," *IEEE/IFIP Intl. Conf. Dependable Systems & Networks*, 2015.

[Seifert 10] Seifert, N., *et al.*, "On the radiation-induced soft error performance of hardened sequential element in advanced bulk CMOS technologies," *IEEE Intl. Reliability Physics Symp.*, 2010.

[Seifert 12] Seifert N., *et al.*, "Soft error susceptibilities of 22 nm tri-gate devices," *IEEE Trans. Nuclear Science*, 2012.

[Sinharoy 11] Sinharoy, B., *et al.*, "IBM POWER7 multicore server processor," *IBM Journal of Research and Development*, 2011.

[Spainhower 99] Spainhower, L. and T. A. Gregg, "IBM S/390 parallel enterprise server G5 fault tolerance: a historical perspective," *IBM Journal of Research & Development*, 1999.

[Synopsys] Synopsys, Inc., Synopsys design suite.

[Tao 93] Tao, D. L. and C. R. P. Hartmann, "A novel concurrent error detection scheme for FFT networks," *IEEE Trans. Parallel and Distributed Systems,* 1993.

[Wang 04] Wang N. J., *et al.*, "Characterizing the effects of transient faults on a high-performance processor pipeline," *IEEE/IFIP Intl. Conf. Dependable Systems & Networks*, 2004.

[Wang 05] Wang, N. J. and S. J. Patel, "ReStore: symptom based soft error detection in microprocessors," *IEEE/IFIP Intl. Conf. Dependable Systems & Networks*, 2005.

[Wang 07] Wang, N. J., A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," *ACM/IEEE Intl. Symp. on Computer Architecture*, 2007.

[Zhang 06] Zhang, M., *et al.*, "Sequential element design with built-in soft error resilience," *IEEE Trans. VLSI*, 2006.