Cross-Layer Resilience Challenges: Metrics and Optimization

(*Invited Paper*)

Subhasish Mitra Kevin Brelsford
Robust Systems Group
Department of EE and Department of CS
Stanford University, Stanford, CA

Pia N. Sanda
IBM Corporation
Poughkeepsie, NY

ABSTRACT

With increasing sources of disturbances in the underlying hardware, a key challenge in design of robust systems is to meet user expectations at required cost. Cross-layer resilience techniques, implemented across multiple layers of the system stack and designed to work together, can help system designers build effective robust systems at the desired cost point. This paper brings to the forefront two major cross-layer resilience challenges:

1. Quantification and validation of the effectiveness of a cross-layer resilience approach to robust system design in overcoming hardware reliability challenges.

2. Global optimization of a robust system design using cross-layer resilience techniques.

I. INTRODUCTION

Most system designs, with the exception of a few such as high-end mainframes and safety-critical systems, **assume** that the underlying hardware will always produce correct outputs during system operation; i.e., possibilities of incorrect outputs due to hardware errors are not explicitly considered during design of most systems. At advanced technology nodes, hardware failure mechanisms that were largely benign in the past are becoming visible at the system level [Borkar 05, Kogge 08, Nassif 10]. Hence, robust systems must be designed such that they meet user expectations despite rising levels of disturbances in the underlying hardware.

Design of fault-tolerant systems, while non-trivial, is achievable but expensive [Ando 03, Bernick 05, Meaney 05]. Hence, the most significant challenge is to achieve required levels of robustness at very low power, performance, area, and design and validation costs.

The following resilience techniques constitute major components of a robust system that can overcome hardware reliability challenges:

1. Circuit-level error correction.
2. Failure prediction / early indication of (possible) failure.
3. Error detection.
4. On-line self-test and diagnostics.
5. Recovery and adaptation.

One approach which can potentially enable design of **cost-effective** robust systems is to use *cross-layer resilience techniques* [Carter 10, DeHon 10] – i.e., one or more of the above resilience techniques may be **implemented across various layers of the system stack** – circuits, architecture, runtime, and application – such that **they work together** to achieve required reliability levels at low cost.

There can be a wide variety of possible implementations of cross-layer resilience techniques. Here are three examples:

1. Error detection may be implemented at logic level, e.g., using duplication, parity and residue techniques, while error recovery may be implemented at architectural and higher layers of abstraction [Ando 03, Meaney 05, Spainhower 99].

2. The implementation of error detection may span multiple layers of abstraction, e.g., combining circuit-level timing error detection [Bowman 09, Ernst 03, Franco 94] with logic-level parity and residue codes [Mitra 00], application-specific checking [Huang 84, Pattabiraman 07] and software-implemented checking [Oh 02a, 02b]. Depending on the target application, the price paid for resilience may be further reduced by combining hardware-level resilience techniques with error-resilience characteristics of target algorithms and intelligent software optimizations [Leem 10].

3. Circuit-level error correction techniques, e.g., flip-flop-level Built-In Soft Error Resilience (BISER) or DICE [Calin 96, Mitra 05, Zhang 06], may be used to correct radiation-induced soft (transient) errors. Circuit failure prediction (to provide early indication of impending circuit failures) [Agarwal 07, Mitra 08, Li 09a], enabled by on-line self-test and diagnostics [Inoue 08, Li 08, 10], may be used to overcome reliability challenges related to transistor aging and gate-oxide early-life failures. Such on-line self-test and diagnostics may be implemented in hardware and orchestrated using software layers [Li 09b]. Adaptation and recovery may span multiple abstraction layers [Mintarno 10].

A cross-layer resilience approach, while attractive, introduces its own set of challenges. This paper focuses on major challenges that must be addressed for success of this approach to robust system design:

1. How do we quantify and validate that a system designed using a cross-layer resilience approach "truly" achieves required levels of reliability?

2. How do we optimize an overall system design using cross-layer resilience techniques and how do we quantify cost-benefit trade-offs at various abstraction layers?

3. While there has been a lot of focus on resilience in processors, future efforts must focus on cross-layer resilience techniques for heterogeneous System-on-Chips (SoCs) with a wide variety of components (processors, accelerators, signal processing engines, FPGAs, and uncore components such as on-chip networks, and memory and I/O controllers).

This paper focuses on the first two challenges. The third challenge related to cross-layer resilience of SoCs is addressed in [Carter 10].

II. METRICS AND VALIDATION

For a given set of resilience techniques, its effectiveness in achieving desired system-level reliability must be evaluated [Muller 10], and associated costs, such as system-level power and performance costs, must be quantified. High-level metrics for reliable (dependable) systems, e.g., reliability, availability, data integrity, mean time to failure, mean time to repair (and several others including performability and maintainability), exist in the fault-tolerant computing literature and have been used for quantifying the benefits of reliable systems [Siewiorek 98]. With cross-layer resilience, such high-level metrics alone may not be sufficient. This is because there is a big gap between such system-level metrics and transistor- or interconnect-level failure sources. (This is somewhat similar to trying to quantify, for example, system-level performance of a multi-core system using SPICE or RTL simulations – this approach suffers from major scalability challenges). While earlier publications addressed some of these issues in the context of hierarchical system-level dependability analysis, e.g., [Goswami 97], systematic development of metrics and abstractions that capture cross-layer aspects of resilience is necessary.

For example, consider the reliability metric. The *reliability* of a system at time t is the probability that the system produces correct outputs up to time t (assuming it produces correct outputs at time 0). In the context of cross-layer resilience, this definition introduces several challenges related to the definition of system boundaries and the levels of abstraction at which various resilience techniques are implemented.

For a concrete example, consider the reliability metric in the context of radiation-induced soft errors in flip-flops. Assume that flip-flop-level soft error rates are known and, for simplicity, assume that all flip-flops have the same soft error rate. To estimate the system-level reliability metric, we also need to quantify the conditional probability that the system produces "correct" outputs given that a soft error has occurred.

For resilience techniques implemented entirely in hardware, i.e., with no software support (e.g., using self-correcting flip-flops such as BISER [Mitra 05, Zhang 06]), quantification of this probability is manageable using techniques such as [Sanda 08, Seshia 07]. (This is a non-trivial task since there are several challenges related to the scalability and accuracy of these techniques). However, for cross-layer resilience solutions, quantification of this probability becomes tricky. Next, we illustrate this fact using two examples.

Example 1

Consider a cross-layer resilience approach where error detection is implemented entirely in hardware while error recovery is implemented in software. In that case, any chip-level analysis technique must separately report situations where incorrect outputs are produced by the chip but the corresponding errors are detected by the implemented error detection schemes in hardware. We refer to these cases as *detected errors*. (For some incorrect outputs, errors will be detected. However, for some other incorrect outputs, the corresponding errors may not be detected depending on the coverage of implemented error detection techniques. Chip-level analysis techniques must distinguish between these two cases). Situations of detected errors will initiate error recovery. (Since error detection is implemented entirely in hardware, recovery may not be initiated for any situation in which incorrect output is produced but the corresponding error is not detected). However, simply reporting cases of detected errors alone is not enough. From full system perspective, it is also important that the recovery techniques (implemented in software) successfully recover the system from these detected errors. For that purpose, other related information must also be reported. Examples include *error detection latency* (the amount of time elapsed between the occurrence of an error and detection of that error), I/O activities during the time elapsed between the occurrence of an error and its detection, and information about whether the system was executing application vs. operating system code during the appearance / detection of the error.

Depending on error sources and associated error rates, very frequent error recovery can impose system-level performance overheads (e.g., discussions in [Bowman 09]). Hence, low-level error rate information may also be required by high-level analysis techniques to estimate overall system performance overheads of implemented resilience techniques.

Example 2

In this example of cross-layer resilience, error detection is implemented at a higher abstraction layer. Consider a packet-processing chip targeting networking applications which does not contain any hardware that checks for errors. Upon error injection simulations, erroneous packets will be observed at the chip outputs when compared to fault-free simulation. Without knowledge of protocol-level error detection (e.g., detection of incorrect packet content because of specific encoding of packet data, detection of incorrect packet sequences), these erroneous packets may be pessimistically classified as "incorrect outputs." However, in reality, many of the packet errors may be detected and packets may be retransmitted successfully. Simple assumptions about detectability of all errors that result in incorrect packets, without knowledge of actual hardware design, system configuration or protocol, aren't sufficient either and may lead to optimistic reliability estimation.

As illustrated by the above examples, with cross-layer resilience, simply evaluating error rates at low levels of the system stack alone may not give useful information required to decide whether an overall system meets the reliability goals of the target application. In addition, metrics and abstractions used for evaluating overall system resilience may also depend on the nature of failure sources, e.g., temporary vs. so-called hard (or non-temporary) failures. For example, depending on the application, one may need to consider situations leading to occurrences of temporary errors on a piece of hardware containing a hard failure, and evaluate their effects on system data integrity. Such situations have been considered in the past in the context of Totally Self-Checking (TSC) circuits [McCluskey 90, Siewiorek 98]. A detailed discussion of such dependencies is beyond the scope of this paper.

Based on the above discussions, three related challenges for cross-layer resilience are:

1. Statistical validation and related metrics: There is a need for tools and layer-wise abstractions that can help estimate overall system-level statistical metrics, e.g., reliability and availability, and validate claims about such system-level

metrics. Such estimation requires tight confidence intervals. Hence, smart estimation techniques are required for statistically significant results without explosion in execution times of estimation, e.g., techniques based on rare event sampling methods. Recently, rare-event sampling has been used for quantifying circuit-level process variations [Singhee 07].

It may also be desirable to report "worst situations" of error propagation through the system stack (defining such "worst error situations" can be challenging), and worst execution sequences that can produce such worst situations. Such a worst-case analysis may be required for designs targeting a broad spectrum of applications with varying reliability requirements. This is because the vulnerability of a system to hardware failures strongly depends on executed application, e.g., emerging Recognition, Mining and Synthesis (RMS) workloads may have some degree of inherent resilience to errors [Leem 10] while others may not.

For complex SoCs integrating intellectual-property (IP) blocks from multiple sources, additional complications may be introduced due to possible unavailability of detailed hardware descriptions for intellectual property reasons. For example, widely-used error injection techniques often rely on hardware design descriptions for simulation purposes. In the absence of such design descriptions, the applicability of error injection and the accuracy of obtained results can be questionable.

2. Verification of resilience techniques: It is imperative to verify that resilience techniques correctly perform their tasks under all possible system operation scenarios. Here are a few examples that may be relevant in this context:

(a) Given a set of sequential elements (latches or flip-flops), is the system "truly" protected from errors in those elements? While it may be fairly straightforward to answer this question with circuit-level resilience techniques, Examples 1 and 2, discussed previously, illustrate how this problem can get complicated with cross-layer resilience techniques.

(b) A simple yes / no answer to question (a) may not be sufficient. Designers may be interested in knowing "how often" and "under what scenarios" a system isn't protected from errors in a given set of sequential elements. For example, as shown in [Zhang 06, Seshia 07] and in several other publications, not all flip-flops are equally important in reducing the overall soft error rate of a given design (even if every flip-flop has the same raw soft error rate). Error injection simulations on an Alpha-like microprocessor show that the chip-level soft error rate may be improved by 10 times (vs. a design with no protection) by protecting close to 50% of all flip-flops from soft errors [Zhang 06]. Such analysis requires information about the fraction of overall soft error rate contributed by each flip-flop.

(c) Given a set of error detection techniques, does the system "properly" recover from erroneous states under all scenarios? Error recovery techniques implemented entirely in hardware may be manageable, while error recovery with firmware or higher-level support (e.g., application-level checkpointing) can significantly complicate this problem.

(d) For systems with built in self-tuning and adaptation in the presence of early-life failures and transistor aging (e.g., [Li 09a, Mintarno 10]), how do we verify that the system operates at optimal power-performance-reliability points throughout lifetime? A related issue is the *error rate verification problem*: how do we ensure "correct" operating points (e.g., voltage, frequency) of a system over lifetime such that hardware-induced errors, together with cross-layer resilience techniques to mitigate them, do not compromise overall system energy efficiency (e.g., discussions on this topic in [Bowman 09]).

(e) For large-scale heterogeneous systems, e.g., cloud computing environments consisting of a mix of heterogeneous computing hardware with diverse reliability specifications, what are the effects of scheduling and workload transitions across servers on overall application-level resilience? The higher layers of the system stack must account for this variability in order to ensure consistent reliability to end users.

3. Reliability grades: There is an emerging need for "grading" a system with respect to system-level metrics such as reliability and data integrity. In fact, the required reliability grade of a system (or a sub-system) may vary dynamically depending on the workload. For certain domains, e.g., scientific computing or aerospace, such grades exist. However, it is not clear how to systematically generalize this concept and take advantage of it for low-cost resilience. This aspect will be discussed more in the context of optimization of cross-layer resilience (Sec. III).

III. OPTIMIZATION

Error resilience must be an essential component of overall system design optimization. This is because most designs are power- and performance-constrained. Arbitrary insertion of resilience techniques can violate such constraints. As shown in this section, cross-layer resilience techniques can result in better designs (cost-wise) with proper optimization. Hence, optimization of cross-layer resilience techniques is a major challenge for future systems.

We demonstrate this point using a simple yet illustrative example: how to choose techniques for protecting a given set of flip-flops in a design from radiation-induced soft errors affecting the flip-flops? While this particular example focuses on optimization across circuit- and logic-levels, more opportunities and challenges exist for optimization across other layers of the system stack including the application.

As discussed extensively in the literature on soft errors, (unprotected) flip-flops can be significant contributors to overall system soft error rates in sub-45nm technologies. At the circuit level, special soft-error-resilient flip-flop designs, e.g., BISER [Mitra 05, Zhang 06], DICE [Calin 96], LEAP [Lee 10], may be used to minimize chip-level soft error rates. For example, *BISER*, an acronym for *Built-In Soft Error Resilience*, modifies flip-flop designs such that soft errors affecting flip-flops can be self-corrected. However, such flip-flop designs require additional transistors with increased power, area and delay.

At a higher level of design abstraction, logic-level error detection techniques may be used for soft error resilience. Examples include parity techniques with a variety of constraints on logic sharing [Mitra 00], residue checks for arithmetic units, assertion checking (using design-level properties or logic implications [Nepal 08]), and algorithm-specific error detection techniques (e.g., [Huang 84]).

These circuit- and logic-level solutions span a wide spectrum of cost vs. error resilience trade-offs. Hence, a major challenge is to identify the "optimized" solution for a given

design to achieve maximized resilience at minimized cost. For example, consider the following optimization problem:

Given a design and a set of flip-flops (out of all flip-flops in the given design) to be protected from soft errors, identify the subset of flip-flops (from the given set) to be protected using BISER. The remaining flip-flops from the given set will be protected using a single parity bit. The objective is to minimize associated system-level costs (power, delay and area).

The above problem assumes the existence of techniques, e.g., [Sanda 08, Seshia 07], for identifying the set of flip-flops that need to be protected from soft errors.

One way of protecting the design is to replace all flip-flops from the given set using BISER flip-flops (Fig. 3.1a). (Here, BISER is used for illustration purposes. One can use other soft-error-resilient flip-flops). As discussed earlier, a BISER flip-flop incurs additional costs at the library cell level [Zhang 06]. Also, a single BISER flip-flop protects the corresponding flip-flop in the original design (and no other flip-flop). However, BISER does not require error signal routing or error recovery because soft errors are corrected at the flip-flop level.

In contrast, a parity technique can protect multiple flip-flops from soft errors using a single parity bit (the probability of multiple flip-flops getting simultaneously affected by soft errors is very small). Figure 3.1b shows a parity technique where all flip-flops from the given set are protected using a single parity bit. This technique introduces additional parity prediction and parity checking logic (for detailed descriptions of parity prediction and parity checking for arbitrary logic circuits, the reader is referred to [Mitra 00]). Since we are interested in flip-flop soft error protection only, we do not require logic sharing constraints (unlike the case when combinational logic error protection is required). Since parity is an error detection technique, error signal routing is required and recovery must be initiated using hardware (e.g., instruction-level retry [Meaney 05]) or software techniques.

Given the trade-offs between BISER and parity, as discussed above, a third option is to protect a subset of flip-flops from the given set using BISER and the rest of the flip-flops (from the given set) using a single parity bit (more complex examples can be created using multiple parity bits). This is shown in Fig. 3.1c. The question is: which subset from the given set should be protected using BISER (the remaining flip-flops from the given set will be protected using a single parity bit) such that the associated system-level power, performance and area costs are minimized ?

For this problem, several scenarios are possible:

1. There is a single best solution, BISER or single-bit parity, irrespective of the design or the given set of flip-flops to be protected (similar to Figs. 3.1a or 3.1b).

2. Given a design, there is a single best solution (BISER or single-bit parity) irrespective of the set of flip-flops to be protected (similar to Figs. 3.1a or 3.1b).

3. Given a design and the set of flip-flops to be protected, there is a single best solution such that all flip-flops from the given set are protected entirely using BISER or single-bit parity (i.e., there is no need for combining the two techniques) (similar to Figs. 3.1a or 3.1b).

4. Depending on the design and the given set of flip-flops to be protected, a subset of these flip-flops need to be protected using BISER and the remaining ones using a single parity bit (similar to Fig. 3.1c).

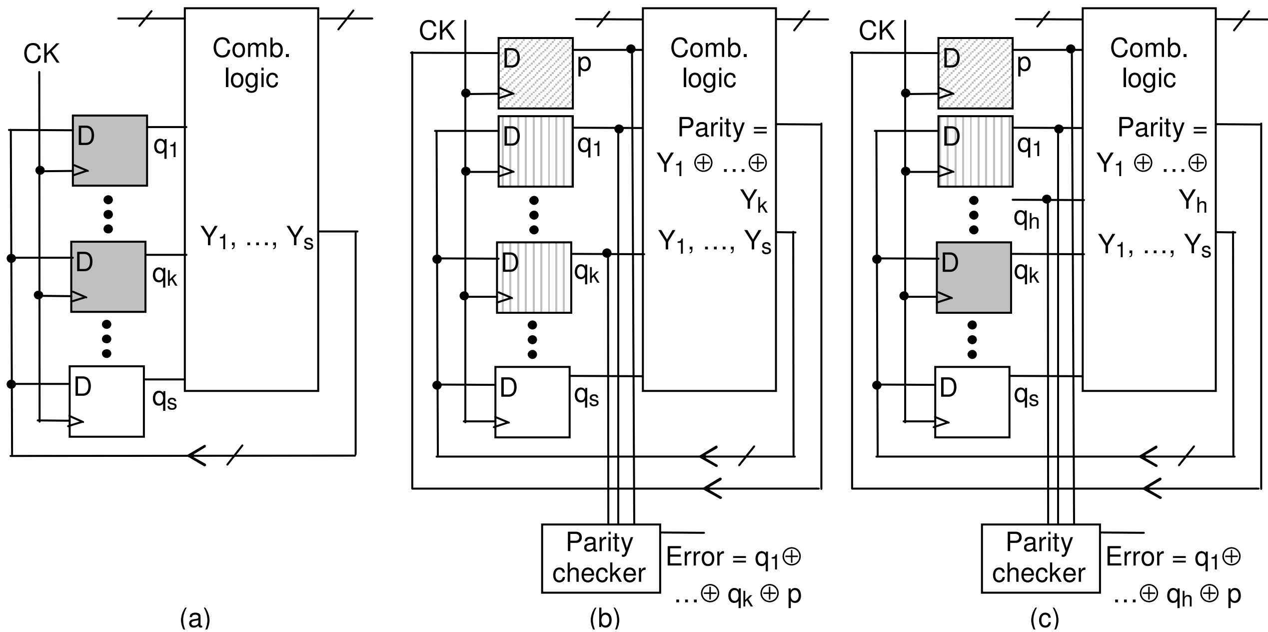

Figure 3.1. (a) Design with a given set of flip-flops (with outputs $q_1, \ldots, q_k$) protected from soft errors using BISER. Remaining flip-flops (with outputs $q_{k+1}, \ldots, q_s$) not protected. (b) Same design as Fig. 3.1a. Flip-flops (with outputs $q_1, \ldots, q_k$) protected using a single parity bit. Please note additional combinational logic output (Parity), additional flip-flop (output p) and additional parity checker. (c) Design in Fig. 3.1a with a subset (flip-flops with outputs $q_1, \ldots, q_h$) from the given set of flip-flops protected using a single parity bit (flip-flop q_h not shown in the figure). Remaining flip-flops (with outputs $q_{h+1}, \ldots, q_k$) from the given set protected using BISER.

For the first scenario, we do not need optimization. For the second and third scenarios, we can simply generate two versions of the design, one with BISER (Fig. 3.1a) and the other using a single parity bit (Fig. 3.1b), and then choose the lowest-cost solution. Scenario 4 requires optimization. Next, in Fig. 3.2, we present simulation results on an actual design to demonstrate that we actually encounter Scenario 4.

Figure 3.2 shows synthesis results (using Synopsys Design Compiler and 45nm Nangate OpenCell library) for the SimpleSPI design from http://www.opencores.org. Reported power estimates were correlated with power numbers extracted from layout. We did not target array-structured register file in the design for soft error protection. For designs using BISER, the area and power costs of a BISER flip-flop are assumed to be 2.3 times that of the corresponding unprotected flip-flop (with minimal delay impact). The power cost is similar to [Zhang 06] and we assumed pessimistic area cost (unlike [Zhang 06] which reuses scan test and debug resources).

In Fig. 3.2, we present results from two sets of experiments:

Experiment Set 1: 50% of all flip-flops are randomly chosen to form the given set of flip-flops to be protected from soft errors. Figures 3.2a-c show power penalties obtained from synthesis results for three randomly chosen sets of flip-flops.

Experiment Set 2: Similar to Experiment Set 1 except that 90% of all flip-flops are randomly chosen to form the given set of flip-flops to be protected from soft errors (Figs. 3.2d-f).

In the graphs in Fig. 3.2, each point represents a design where a certain subset of flip-flops (with cardinality indicated by its x-axis value) is randomly chosen from the given set of flip-flops to be protected. This subset is protected using a single parity bit and the remaining flip-flops from the given set are protected using BISER. For each case, it is clear that the best design (i.e., with smallest power penalty) uses a mix of BISER and single-bit parity demonstrating the effectiveness of cross-layer resilience and the need for optimization across multiple abstraction layers. All designs in Fig. 3.2 were synthesized to achieve clock frequency of 1 GHz. For single-bit parity, the error signal was routed to a primary output and no on-chip recovery unit was used. Area results obtained from synthesis also show similar trends (as in Fig 3.2).

Additional opportunities exist for application-aware optimization of resilience. In Sec. II, we had a short discussion on reliability grades. With dynamically changing reliability grades, it may be possible to "dial" reliability vs. costs on-the-fly. However, this requires dynamic reliability management across multiple abstraction layers that can turn resilience features (e.g., error checking) on / off according to application demands trading off power / performance vs. reliability.

Here is a concrete example of such application-aware optimization to reduce overall power impact of resilience techniques. BISER can be configured, during system operation, to operate in one of two modes – an *error resilient mode* in which BISER protection is turned on, and an *economy mode* in which BISER protection is turned off. Such **configurability** can be practically implemented in hardware and may be activated with software orchestration [Zhang 06]. It can minimize system-level power cost of BISER by turning on the error-resilient mode only for critical computation. However, information flow across abstraction layers to utilize such configurability during system operation is an open question.

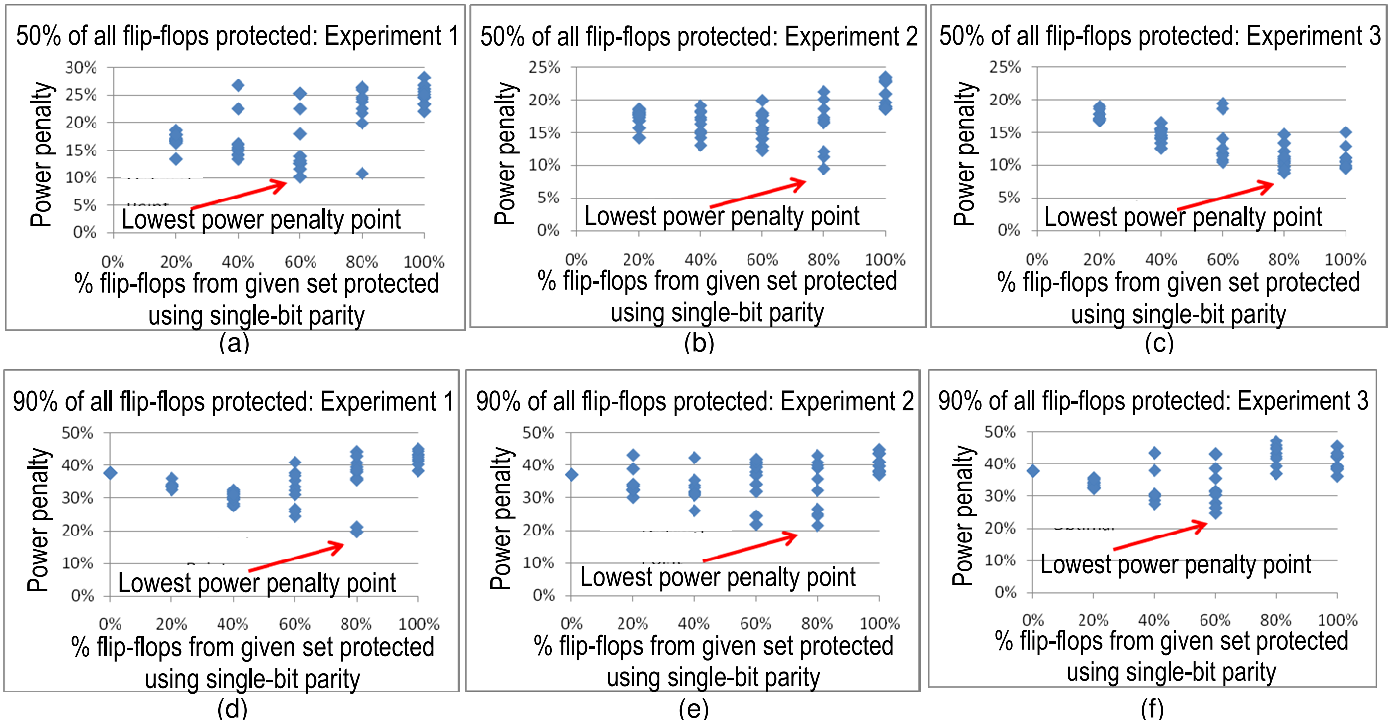

Figure 3.2. Synthesis results demonstrating the effectiveness of cross-layer resilience techniques and the need for optimization of resilience techniques across multiple abstraction layers. (a)-(c): 50% of all flip-flops to be protected from soft errors. (d)-(f): 90% of all flip-flops to be protected from soft errors. For each graph, the 0% (100%) point on the X-axis corresponds to a design where all flip-flops are protected using BISER (a single parity bit). As shown in all graphs, the best design obtained from synthesis (i.e., the one with lowest power penalty) contains a mix of BISER and single-bit parity.

Another opportunity for optimization arises from possible reuse of resilience features for other Design for Excellence (DFX) activities such as Design for Testability (DFT), Design for post-silicon Validation and debug (DFV/DFD) and Design for Yield (DFY). For example, the area impact of BISER can be significantly reduced by reusing on-chip scan resources for post-silicon validation and testing [Mitra 05, Zhang 06].

IV. CONCLUSION

Future robust systems must accept the fact that the underlying hardware will be imperfect, and implement special techniques to ensure resilience to hardware imperfections. The biggest challenge in this context is to implement resilience at lowest cost. Cross-layer resilience techniques can potentially enable low-cost robust systems. As discussed in this paper, challenges related to metrics, validation and optimization of cross-layer resilience must be overcome for such a system design approach to be successful.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 0637190 to the Computing Research Association. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the CRA and NSF. Thanks to Lori Bechtold, Nicholas Carter, John Daly, André DeHon, Eliezer Dekel, Prabhakar Kudva, Jim Van Oosten, Charles Recchia, Bianca Schroeder, Mark Seager, Gary Swift and Jimi Xenidis for their inputs.

REFERENCES

[Agarwal 07] Agarwal, M., *et al.*, "Circuit Failure Prediction and Its Application to Transistor Aging," *Proc. IEEE VLSI Test Symp.*, pp. 277-286, 2007.

[Ando 03] Ando, H., *et al.,* "A 1.3-GHz Fifth-Generation SPARC64 Microprocessor", *IEEE Journal Solid-State Circuits,* Vol. 38, Issue 11, pp. 1896-1905, Nov. 2003.

[Bowman 09] Bowman, K., *et* al., "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance," *IEEE Journal Solid-State Circuits,* Vol. 44, Issue 1, pp. 49-63, Jan. 2009.

[Bernick 05] Bernick, D., *et al.*, "Non-Stop Advanced Architecture," *Proc. IEEE Intl. Conf. Dependable Systems and Networks*, pp. 12-21, 2005.

[Borkar 05] Borkar, S.Y., "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro*, pp. 10-16, Nov.-Dec. 2005.

[Calin 96] Calin, T., M. Nicolaidis, and R. Velaco, "Upset Hardened Memory Design for Submicron CMOS Technology," *IEEE Trans. Nucl. Sci.,* Vol. 43, No. 12, pp. 2874-2878, Dec. 1996.

[Carter 10] Carter, N.P., *et al.*, "Design Techniques for Cross-Layer Resilience," *Proc. Design Automation and Test in Europe,* 2010.

[DeHon 10] DeHon, A., *et al.*, "Vision for Cross-Layer Optimization to Address the Dual Challenges of Energy and Reliability," *Proc. Design Automation and Test in Europe,* 2010.

[Ernst 03] Ernst, D., *et al.*, "Razor: A Low-Power Pipeline based on Circuit-level Timing Speculation," *Proc. IEEE Intl. Symp. Microarchitecture,* pp. 7-18, 2003.

[Franco 94] Franco, P., and E.J. McCluskey, "On-line Delay Testing of Digital Circuits," *Proc. IEEE VLSI Test Symp.*, pp. 167-173, 1994.

[Goswami 97] Goswami, K.K., R.K. Iyer L. Young, "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis," *IEEE Trans. Computers*, Vol. 46, pp. 60-74, Jan. 1997.

[Huang 84] Huang, K.H., and J.A. Abraham, "Algorithm Based Fault Tolerance for Matrix Operations," *IEEE Trans. Computers*, Vol. C-33, No. 6, pp. 518-528, June 1984.

[Inoue 08] Inoue, H., Y. Li and S. Mitra, "VAST: Virtualization-Assisted Concurrent Autonomous Self-Test," *Proc. IEEE Intl. Test Conf.*, 2008.

[Kogge 08] Kogge, P., *et al.*, "Exascale Computing Study: Technology Challenges in Achieving Exascale Systems," 2008.

[Lee 10] Lee, H., *et al.*, "LEAP: Layout Design through Error-Aware Placement for Soft-Error Resilient Sequential Cell Design," *Proc. IEEE Intl. Reliability Physics Symp.*, 2010.

[Leem 10] Leem, L., *et al.,* "ERSA: Error Resilient System Architecture for Probabilistic Applications," *Proc. Design Automation and Test in Europe*, 2010.

[Li 08] Li, Y., S. Makar and S. Mitra, "CASP: Concurrent Autonomous Chip Self-Test using Stored Test Patterns," *Proc. Design Automation and Test in Europe*, pp. 885-890, 2008.

[Li 09a] Li, Y., *et al.*, "Overcoming Early-Life Failure and Aging for Robust Systems," *IEEE Design and Test of Computers*, Nov.-Dec. 2009.

[Li 09b] Li, Y., O. Mutlu and S. Mitra, "Operating System Scheduling for Efficient Online Self-Test in Robust Systems," *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design,* 2009.

[Li 10] Li, Y., D.S. Gardner and S. Mitra, "Concurrent Autonomous Self-Test for Uncore Components in SoCs," *Proc. IEEE VLSI Test Symp.*, 2010.

[McCluskey 90] McCluskey, E.J., "Design Techniques for Testable Embedded Error Checkers," *IEEE Computer*, Vol. 23, No. 7, pp. 84-88, July 1990.

[Meaney 05] Meaney, P., *et al.*, "IBM Z990 Soft Error Detection and Recovery," *IEEE Trans. Device and Materials Reliability*, Vol. 5, Issue 3, pp. 419-427, Sept. 2005.

[Mintarno 10] Mintarno, E., "Optimized Self-Tuning for Circuit Aging," *Proc. Design Automation and Test in Europe*, 2010.

[Mitra 00] Mitra, S., and E.J. McCluskey, "Which Concurrent Error Detection Schemes to Choose?" *Proc. IEEE Intl. Test Conf.,* pp. 985-994, 2000.

[Mitra 05] Mitra, S., *et al.*, "Robust System Design with Built-In Soft Error Resilience," *IEEE Computer*, Vol. 38, pp. 43-52, Feb. 2005.

[Mitra 08] Mitra, S., "Globally Optimized Robust Systems to Overcome Scaled CMOS Reliability Challenges," *Proc. Design Automation and Test in Europe*, 2008.

[Muller 10] Muller, K.P., and P.N. Sanda, "Soft Error Assessments for Servers," *Proc. Intl. Reliability Physics Symp.*, 2010.

[Nassif 10] Nassif, S.R., *et al.*, "A Resilience Roadmap," *Proc. Design Automation and Test in Europe,* 2010.

[Nepal 08] Nepal, K., *et al.,* "Using Implications for Online Error Detection," *Proc. IEEE Intl. Test Conf.*, 2008.

[Oh 02a] Oh, N., P.P. Shirvani and E.J. McCluskey, "Error Detection by Duplicated Instructions in Super-Scalar Processors," *IEEE Trans. Reliability*, Vol. 51, Issue 1, pp. 63-75, March 2002.

[Oh 02b] Oh, N., S. Mitra and E.J. McCluskey, "ED^4I: Error Detection by Diverse Data and Duplicated Instructions" *IEEE Trans. Computers,* Vol. 51, No. 2, pp. 180-199, Feb. 2002.

[Pattabiraman 07] Pattabiraman, K., et. al., "Automated Derivation of Application-Aware Error Detectors using Static Analysis," *Proc. IEEE Intl. Symp. On-line Testing*, pp. 211-16, 2007.

[Sanda 08] Sanda, P.N., *et al.*, "Soft Error Resilience of the IBM POWER6 Processor," *IBM Journal Research and Development*, Vol 52, Number 3, 2008.

[Spainhower 99] Spainhower, L., and T.A. Gregg, "S/390 Parallel Enterprise Server G5 Fault Tolerance," *IBM Journal Res. and Dev.*, Vol. 43, pp. 863-873, Sept./Nov., 1999.

[Seshia 07] Seshia, S., W. Li and S. Mitra, "Verification Guided Soft Error Resilience," *Proc. Design Automation and Test in Europe*, pp. 1442-1447, 2007.

[Siewiorek 98] Siewiorek, D.P., and R.S Swarz, *Reliable Computer Systems: Design and Evaluation*, 1998.

[Singhee 07] Singhee, A., and R. A. Rutenbar, "Statistical Blockade: A Novel Method for Very Fast Monte Carlo Simulation of Rare Circuit Events, and its Application," *Proc. Design Automation and Test in Europe*, pp. 1379–1384, 2007.

[Zhang 06] Zhang, M., *et al.*, "Sequential Element Design with Built-In Soft Error Resilience," *IEEE Trans. VLSI*, Vol. 14, Issue 12, pp. 1368-1378, Dec. 2006.