QED Post-Silicon Validation and Debug: Frequently Asked Questions

David Lin
Dept. of Electrical Engineering
Stanford University
Stanford, CA, USA

Subhasish Mitra
Depts. of Electrical Engineering and Computer Science
Stanford University
Stanford, CA, USA

Abstract

During post-silicon validation and debug, one or more manufactured integrated circuits (ICs) are tested in actual system environments to detect and fix design flaws (bugs). According to several industrial reports, the costs of post-silicon validation and debug are rising faster than design costs. Hence, new techniques are essential to reverse this trend. QED, an acronym for Quick Error Detection, is such a technique that effectively overcomes several post-silicon validation and debug challenges. QED systematically creates a wide variety of validation tests to quickly detect bugs, not only inside processor cores, but also inside uncore components (i.e., components in an SoC that are neither processor cores nor co-processors) of multi-core SoCs. In this paper, we present a brief overview of QED through a series of frequently asked questions.

Keywords

Debug, Post-Silicon Validation, Quick Error Detection, Verification

I. Frequently Asked Questions

Question 1. Why is post-silicon validation / debug important?

Post-silicon validation / debug is crucial because traditional pre-silicon verification alone is inadequate for today's complex integrated circuits (ICs). As a result, critical design bugs escape pre-silicon verification and are detected only during post-silicon validation [2], [3], [16], [23], [29]. Design bugs can be broadly classified into two categories:

(a) *Logic bugs* that are caused by design errors. In addition to bugs in the hardware implementation, this category includes incorrect interactions between the hardware implementation and the low-level system software (e.g., firmware).

(b) *Electrical bugs* that are caused by subtle interactions between a design and its "electrical" state. Electrical bugs often manifest themselves only under specific operating conditions, such as voltage, frequency, and temperature corners [36]. Examples of electrical bugs include signal integrity (e.g., cross-talk, power-supply noise), thermal effects, and process variations.

Traditional pre-silicon verification is too slow for "difficult" logic bugs, and it also does not adequately address electrical bugs that appear only after ICs are manufactured. These challenges get further magnified with the slowdown of the classical silicon CMOS (Dennard) scaling [8], as ICs incorporate tremendous design complexity to meet performance and energy-efficiency requirements. Examples include the use of multiple processor cores; co-processors and accelerators such as GPUs; uncore components such as on-chip cache controllers, memory controllers, and network controllers; adaptive power, thermal and reliability management; and, proliferation of heterogeneous integration. Here uncore components (also referred to as the nest or the northbridge) are defined as components in a system-on-chip (SoC) that are neither processor cores nor co-processors.

Existing post-silicon validation and debug practices are generally *ad-hoc*, and several sources [1], [44] indicate that their costs are rising faster than design costs. For example, it may take several weeks to debug a single bug during the post-silicon phase. Without systematic and scalable ways of taming such increasing levels of complexity, future systems can end up being highly vulnerable to bugs that may compromise correct operation and introduce security risks.

Question 2. Why is post-silicon validation and debug difficult?

During post-silicon validation and debug, manufactured ICs are tested in actual systems in order to detect and fix bugs. When a failure is detected, the bug that caused the failure is localized and root-caused (and then fixed). Several sources indicate that the effort to localize bugs from observed failures dominates the overall cost of post-silicon validation and debug [1], [6], [22], [23].

Post-silicon bug localization is difficult because existing approaches suffer from two major challenges:

(a) Reliance on failure reproduction, which involves returning the system to an error free state and re-executing the failure-causing stimuli (i.e. sequence of instructions, interrupts, voltage, temperature and frequency operating conditions). Failure reproduction is very difficult for complex SoCs with multiple clock domains and asynchronous I/Os.

(b) Reliance on system-level simulation to generate expected or golden responses. System-level simulation is several orders of magnitude slower than actual silicon (e.g., 1,000 cycles / second in simulation vs. 1 billion cycles / second for a 1GHz chip).

Question 3. What can be done to overcome post-silicon validation and debug challenges?

"Systematic" and "automated" techniques are essential to overcome the scalability challenges of existing post-silicon validation and debug approaches. Our analysis of bug databases for several commercial SoCs (enabled by our collaborations with several industrial partners) clearly indicates that these challenges are primarily caused by long error detection latencies [20], [26], [27], [38]. Error detection latency is defined as the time elapsed from when a test activates a bug and creates an error in the system to when the error manifests as an observable failure (e.g., system crash, timeout, deadlock, exceptions, or incorrect results). Ideally, error detection latencies should not exceed a few thousand clock cycles because most ICs contain trace buffers that can record on the order of 1,000 clock cycles of history [1]. However, during post-silicon validation and debug, error detection latencies for "difficult" bugs can exceed several millions or even billions of clock cycles, especially for bugs inside uncore components of multi-core SoCs [26]. Such

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

long error detection latencies are highly challenging because it is extremely difficult to trace very far back into the history of system operation for debug.

We created the Quick Error Detection (QED) technique [20], [26], [27] to overcome the long error detection latency challenge. QED is organized as a variety of QED transformations that systematically transform existing tests (referred to as original tests) into new QED tests with bounded error detection latencies and improved coverage. A wide variety of tests, e.g., architecture-specific focused tests, random instruction tests, or end-user applications, can be transformed using QED. Examples of QED transformations include Error Detection using Duplicated Instructions for Validation (EDDI-V), EDDI-V with diversity, Control Flow Checking using Software Signatures for Validation (CFCSS-V), Control Flow Tracking using Software Signatures for Validation (CFTSS-V), and Proactive Load and Check (PLC). The details of these QED transformation techniques are presented in [20], [26], [27]. The QED transformations can be implemented entirely in software, in which case they are readily applicable during post-silicon validation of existing ICs. QED also can be augmented with hardware support to further improve error detection latency and coverage, and to reduce QED test execution time.

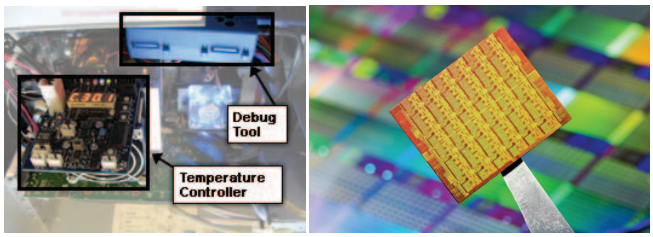
The basic principles of QED are rooted in Software Implemented Hardware Fault Tolerance (SIHFT) techniques [28], [30], [31], [32]. However, there are several major differences between QED for post-silicon validation and debug vs. SIHFT for fault-tolerant computing. For example, during post-silicon validation and debug, error detection latency is extremely important because debug time, rather than test execution time, dominates the overall cost [22]. Furthermore, QED tests must continue to detect bugs that are detected by the original tests [20], [26]. In contrast, performance impact (which is related to execution time) plays a very important role for fault-tolerant computing.

Question 4. What are the published QED results?

The following presents a summary of our published results.

(a) For the Intel® Core™ i7-based hardware platform (Fig. 1a), QED improves error detection latencies of electrical bugs by 6 orders of magnitude while simultaneously improving coverage 4-fold [20]. This is shown in Fig. 2, where the vertical axis represents the percentage of errors detected (normalized to QED, due to confidentiality reasons) and the horizontal axis represents the error detection latencies. The coverage improvement of QED is also demonstrated in Fig. 3; there is not a single operating point (frequency and voltage pair) for which the original test failed but the QED test passed. Furthermore, the operating point labeled with a star in Fig. 3 demonstrates coverage improvement using QED: for this operating point, the original test passed, whereas the QED test detected errors very quickly. QED creates valid tests that do not bring the system into any "illegal" states. Therefore, errors detected by QED are actual errors in the system. These results were also confirmed using the Intel® 48-core Single-Chip Cloud Computer [21] (Fig. 1b). For the processor core labeled with a star in Fig. 4, the original test passed but the QED test detected errors very quickly.

(b) We simulated an extensive list of "difficult" logic bugs (abstracted from actual bug databases) inside processor cores and uncore components using an OpenSPARC T2-like SoC [33]. The details of the logic bugs and our simulation setup are presented in [26]. The results are summarized in Fig. 5 (details in [26]), where the vertical axis represents the absolute cumulative percentage of bugs detected, and the horizontal axis shows the error detection latencies. QED improves error detection latencies by several orders of magnitude, from tens of millions or even billions of clock cycles to (mostly) a few hundred clock cycles, and simultaneously improves coverage up to 2-fold [26].



(a) (b)

Figure 1. (a) Quad-core Intel® Core™ i7-based hardware platform with a DX58SO motherboard, a temperature controller, and a proprietary debug tool. (b) Intel® 48-core Single-Chip Cloud Computer [21].

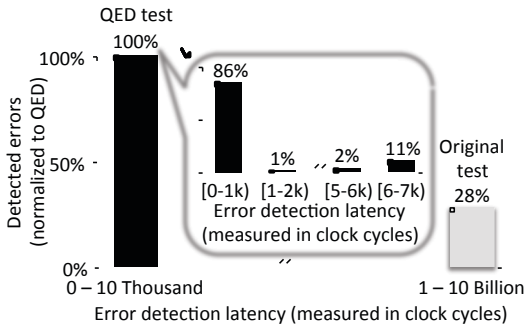


Figure 2. Distribution of error detection latencies for the Intel® Core™ i7-based hardware platform for the original Linpack test [14] with end result checks (Original test) and the Linpack test with QED (QED test). For confidentiality reasons, percentage of detected errors is normalized to QED. (The details are in [20]).

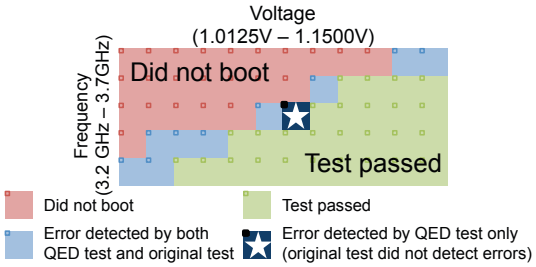


Figure 3. Shmoo plot for Intel® Core™ i7-based hardware platform running the Linpack test [14]. The operating point labeled with a star demonstrates coverage improvement using QED: the original test passed whereas the QED test detected errors very quickly.

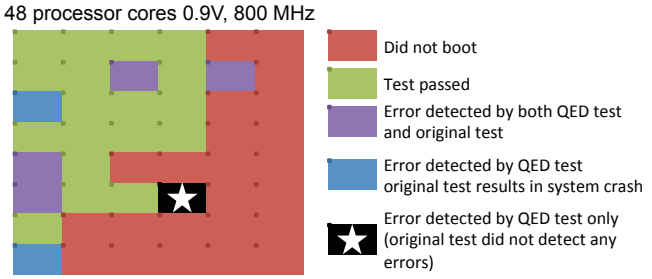
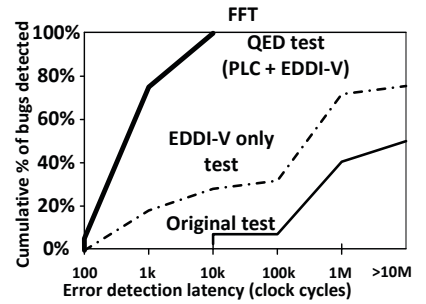
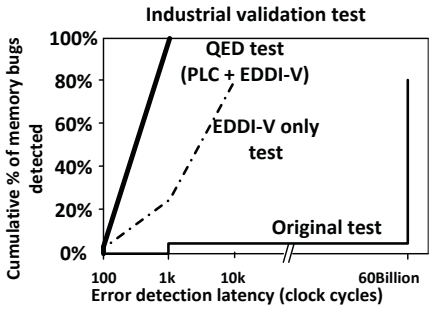


Figure 4. The Intel® 48-core Single-Chip Cloud Computer running the Linpack test [14]. Each box represents an individual processor core. The processor core labeled with a star demonstrates coverage improvement using QED: the original test passed whereas the QED test detected errors very quickly.



(a)



(b)

Figure 5. Error detection latencies and coverage of post-silicon validation tests on a simulated OpenSPARC T2-like SoC [33] using "difficult" logic bugs abstracted from actual bug databases. (a) FFT program from SPLASH-2 benchmark suite [42]. (b) Industrial validation test. The cumulative percentage of bugs detected represents absolute (and not normalized) percentage values.

Question 5. Is QED effective only for bugs inside processor cores?

QED targets bugs inside both processor cores and uncore components of SoCs. In [26], we simulated an extensive list of bugs, abstracted from actual "difficult" bugs that occurred in several commercial multi-core SoCs, to evaluate the effectiveness of QED. These are primarily logic bugs inside uncore components such as cache controllers, memory controllers, and on-chip interconnection networks (in addition to bugs inside processor cores). These bugs are considered to be difficult because the corresponding bug reports indicate very long debug times. Results in [26] (also summarized in Fig. 5) demonstrate that QED is highly effective for improving error detection latencies and coverage for bugs inside uncore components, as well as bugs inside processor cores.

Question 6. Is QED effective only for electrical bugs?

QED is effective for both electrical and logic bugs. [26] demonstrated that QED is highly effective for an extensive list of logic bugs abstracted from difficult logic bugs that occurred in several commercial multi-core SoCs (Fig. 5). [20] demonstrated that QED is highly effective for electrical bugs using the Intel® Core™ i7-based hardware platform (Fig 2. and Fig. 3).

Question 7. Is QED different from self-checking tests?

QED tests are different from self-checking tests that check the results of instructions by comparing the results against those derived using a golden model [5], or using instructions executed on processor cores [37], [43]. As demonstrated in [26], [27], such self-checking tests can incur extremely long error detection latencies, especially for bugs inside uncore components. In contrast, QED systematically inserts fine-grained and proactive checks to bound error detection latencies as well as to improve coverage for bugs inside both processor cores and uncore components.

Question 8. Is QED different from assertions?

QED is different from assertions used during post-silicon validation. The use of assertions during post-silicon validation suffers from several challenges:

1. Numerous assertions may be generated for a given design. These assertions have to be carefully crafted, and it is difficult to keep them up-to-date and to validate their correctness [7], [40]. While there exist techniques for automatically generating assertions (e.g., [15], [18], [39], [40]), it may be difficult to implement all such assertions in hardware.
2. Reconfigurable logic can somewhat ease the implementation of assertions in hardware [1], [9], [17]; however, one must be careful about selecting the relevant set of assertions for effective post-silicon bug localization.
3. An assertion may depend on signals that are located in two different regions of a chip; such an assertion must be decomposed into components that only use nearby signals [29].

In contrast, QED enables "structured" and "systematic" ways of performing extensive checks while avoiding the drawbacks of design-specific assertions.

Question 9. Doesn't the test execution time go up with QED?

QED does increase test execution time [26]. However, the costs of post-silicon validation and debug are dominated by the time required to localize bugs rather than the time required for running tests [22]. Since QED can detect bugs very quickly (e.g., a few hundred clock cycles after the bug is activated [20], [26]), bug localization can become significantly easier, resulting in a net improvement in productivity. Furthermore, the execution time impact of QED can be reduced using hardware support.

Question 10. Since QED utilizes redundant execution, can QED miss bugs that affect both executions in the same way?

There are a wide variety of QED transformations, and time-redundant re-execution of instructions (EDDI-V) is only one of the techniques used by QED. For example, the EDDI-V with diversity transformation incorporates data diversity techniques based on the principles of ED^{4}I [31]. This ensures that the instructions inserted by QED execute differently vs. the original instructions, thereby minimizing the chances of a bug affecting the original instructions and the QED instructions identically. Moreover, for difficult bugs, time-redundant re-execution itself creates diversity (as shown by our results in [26]). Furthermore, QED techniques such as CFCSS-V and CFTSS-V do not rely on instruction re-execution.

Question 11. How do you localize bugs after quick detection using QED?

After quick detection using QED, many post-silicon bug localization techniques can be used: IFRA and BLoG [34], [35], simulation-based debug [10], [24], or debug techniques based on formal methods [11]. These techniques can directly benefit from the extremely short error detection latencies and improved coverage of QED. With the increasing complexity of uncore components in SoCs, new techniques for localizing bugs inside uncore components are required.

Question 12: Why does QED improve coverage?

QED improves the coverage of post-silicon validation tests by detecting errors that would otherwise be masked or undetected due to insufficient checking. For example, a bug may create an error in the value stored in a register, say register R1. Other instructions in the test that write to register R1 may mask that error by overwriting the erroneous value. Unless the value stored in register R1 is checked before it is overwritten, the error will be masked. QED inserts fine-grained and proactive checks to quickly detect such errors, as a result, coverage improves.

Question 13: Does QED introduce intrusiveness?

Since software-implemented QED transformations insert additional instructions, intrusiveness is possible; i.e., there may exist pathological examples for which the original test may detect a bug, but the corresponding QED test may not. To overcome this, QED transformations support special parameters, *Inst_min* and *Inst_max*, that enable systematic approaches to minimize possible intrusiveness due to QED transformations. *Inst_min* is defined as the minimum number of instructions in the original test that must execute consecutively before executing any instructions inserted by QED transformations. *Inst_max* is defined as the maximum number of instructions in the original test that must execute consecutively before executing any instructions inserted by QED transformations. For example, by increasing *Inst_min*, we can possibly reduce the intrusiveness introduced by QED. This is because a longer sequence of instructions from the original test will execute consecutively (uninterrupted by QED). However, excessively large *Inst_min* increases error detection latency. Strategies for adjusting these transformation parameters, e.g., through the creation of QED family tests with a range of *Inst_min* and *Inst_max* values, are discussed in [20], [27]. Moreover, hardware support for QED can minimize the intrusiveness introduced by QED transformations.

Question 14. Can QED be used for pre-silicon verification?

Pre-silicon verification uses emulation / acceleration platforms extensively [3], [4], [23], [25] to improve productivity. QED tests can be used to improve both error detection latencies and coverage in these environments as well.

Question 15. Are logic bug benchmarks available?

Thanks to our industrial collaborators, we received unprecedented access to bug reports for difficult bugs that occurred in several state-of-the-art commercial multi-core SoCs. From these bug reports, we compiled an extensive list of bug scenarios. We worked closely with the respective validation teams to abstract the bug reports into higher-level descriptions (and remove product-specific details). This allows the bugs to be simulated using various micro-architectural and RTL simulators. The list of bugs is presented in [26], and is also available at http://www.bugslist.org. Researchers can also submit their own bug scenarios on the website to benefit the post-silicon validation and debug research community.

Other researchers have also published logic bugs found in research chips, class projects, and errata pages [12], [13], [19], [41]. The list of bugs in [26] subsumes these bugs.

II. Acknowledgement

This research was supported in part by NSF, SRC and SGF. The authors thank Eric Rentschler of AMD, Eswaran S. and Sharad Kumar of Freescale, Wisam Kadry, Ronny Morad, Amir Nahir and Avi Ziv of IBM, and Donald S. Gardner, Nagib Hakim, Jagannath Keshava, and Keshavan Tiruvallur of Intel for their valuable inputs.

III. References

[1] M. Abramovici, "A Reconfigurable Design-for-Debug Infrastructure for SoCs," *Proc. IEEE/ACM Design Automation Conf.*, pp. 7-12, 2006.

[2] A. Adir, A. Nahir, A. Ziv, C. Meissner, and J. Schumann, "Reaching Coverage Closure in Post-Silicon Validation," *Proc. Haifa Verification Conf.*, pp. 60-74, 2010.

[3] A. Adir, *et al.* , "A Unified Methodology for Pre-Silicon Verification and Post-silicon Validation," *Proc. IEEE/ACM Design, Automation and Test in Europe Conf.*, pp. 1-6, 2011.

[4] A. Adir, *et al.* "Leveraging Pre-Silicon Verification Resources for the Post-Silicon Validation of the IBM POWER7 Processor," *Proc. IEEE/ACM Design Automation Conf.* pp. 569-574. 2011.

[5] A. Aharon, *et al.,* "Test Program Generation for Functional Verification of PowerPC Processors in IBM," *Proc. IEEE/ACM Design Automation Conf.,* pp. 279-285, 1995.

[6] M. E. Amyeen, S. Venkataraman, and M. W. Mak, "Microprocessor System Failures Debug and Fault Isolation Methodology," *Proc. IEEE Intl. Test Conf.*, pp. 1-10, 2009.

[7] B. Bentley, and R. Gray, "Validating the Intel Pentium 4 Processor," *Intel Technology Journal*, Vol. 5 No. 1, pp. 1-8, 2001.

[8] M. Bohr, "The New Era of Scaling in an SoC World," *Proc. IEEE Solid-State Circuits Conf.,* pp. 23-28, 2009.

[9] M. Boule, J.-S. Chenard, and Z. Zilic, "Assertion Checkers in Verification, Silicon Debug and In-Field Diagnosis," *Proc. IEEE Intl. Symp. on Quality Electronic Design*, pp. 613-620, 2007.

[10] K.-H. Chang, I. L. Markov, and V. Bertacco, "Automated Post-Silicon Debugging and Repair," *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, pp. 91-98, 2007.

[11] F. M. De Paula, A. J. Hu, and A. Nahir, "nuTAB-BackSpace: Rewriting to Normalize Non-Determinism in Post-Silicon Debug Traces," *Proc. Intl. Conf. on Computer Aided Verification*, pp. 513-531, 2012.

[12] A. DeOrio, A. Bauserman, and V. Bertacco, "Post-Silicon Verification for Cache Coherence," *Proc. IEEE Intl. Conf. Computer Design*, pp. 348-355, 2008.

[13] A. DeOrio, I. Wagner, and V. Bertacco, "DACOTA: Post-Silicon Validation of the Memory Subsystem in Multi-Core Designs," *Proc. IEEE Intl. Symp. High-Performance Computer Architecture*, pp. 405-416, 2009.

[14] J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: Past, Present and Future," *Concurrency and Computation: Practice and Experience.* Vol. 15, No. 9, pp. 803-820, 2003.

[15] M. D. Ernst, *et al.*, "The Daikon System for Dynamic Detection of Likely Invariants," *Science of Computer Programming*, Vol. 69, No. 1-3, pp. 35-45, Dec. 2007.

[16] T. J. Foster, D. L. Lastor, and P. Singh, "First Silicon Functional Validation and Debug of Multicore Microprocessors," *IEEE Trans. Very Large Scale Integration Systems,* Vol. 15, No. 5, 2007.

[17] M. Gao, H.-M. Chang, P. Lisherness, and K.-T. Cheng, "Time-Multiplexed Online Checking: A Feasibility Study," *Proc. IEEE Asian Test Symp*., pp. 371-376, 2008.

[18] S. Hangal, S. Narayanan, N. Chandra, and S. Chakravorty, "IODINE: A Tool to Automatically Infer Dynamic Invariants," *Proc. IEEE/ACM Design Automation Conf*., pp. 775-778, 2005.

[19] R. C. Ho, C. H. Yang, M. A. Horowitz, and D. L. Dill, "Architecture Validation for Processors," *Proc. ACM/IEEE Intl. Symp. Computer Architecture*, pp. 404-413, 1995.

[20] T. Hong, *et al*., "QED: Quick Error Detection Tests for Effective Post-Silicon Validation," *Proc. IEEE Intl. Test Conf.*, pp. 1-10, 2010.

[21] J. Howard, *et al.,* "A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS," *Proc. IEEE. Intl. Solid-State Circuits Conf.,* pp. 7-11, 2010.

[22] D. Josephson, "The Good, the Bad, and the Ugly of Silicon Debug," *Proc. IEEE/ACM Design Automation Conf*., pp. 3-6, 2006.

[23] J. Keshava, N. Hakim, and C. Prudvi, "Post-silicon Validation Challenges: How EDA and Academia Can Help," *Proc. IEEE/ACM Design Automation Conf*., pp. 3-7, 2010.

[24] A. Krstic, L.-C. Wang, K.-T. Cheng, and T. M. Mak, "Diagnosis-Based Post-Silicon Timing Validation Using Statistical Tools and Methodologies," *Proc. IEEE Intl. Test Conf*., pp. 339-348, 2003.

[25] C. A. Krygowski, *et al.,* "Functional Verification of the IBM System z10 Processor Chipset," *IBM Journal of Research and Development,* Vol. 53, No. 1, pp. 1-11, 2009.

[26] D. Lin, T. Hong, F. Fallah, N. Hakim, and S. Mitra "Quick Detection of Difficult Bugs for Effective Post-Silicon Validation," *Proc. IEEE/ACM Design Automation Conf.,* pp. 561-566, 2012.

[27] D. Lin, *et al.*, "Overcoming Post-Silicon Validation Challenges Through Quick Error Detection (QED)," *Proc. IEEE/ACM Design, Automation and Test in Europe Conf.,* pp. 320-325, 2013.

[28] M. N. Lovellette, *et al.*, "Strategies for Fault-tolerant Space-based Computing: Lessons Learned from the ARGOS Testbed," *Proc. Aerospace Conf*., pp. 5-2109-5-2119, 2002.

[29] S. Mitra, S. A. Seshia, and N. Nicolici, "Post-Silicon Validation Opportunities, Challenges and Recent Advances," *Proc. IEEE/ACM Design Automation Conf*., pp. 12-17, 2010.

[30] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error Detection by Duplicated Instructions in Super-Scalar Processors," *IEEE Trans. on Reliability*, Vol. 51, No. 1, pp. 63-75, 2002.

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

[32] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Control Flow Checking by Software Signatures," *IEEE Trans. on Reliability,* Vol. 51, No. 1, pp. 111-122, 2002.

[33] "OpenSPARC: World's First Free 64-bit Microprocessor," http://www.opensparc.net.

[34] S.-B. Park, T. Hong, and S. Mitra, "Post-Silicon Bug Localization in Processors Using Instruction Footprint Recording and Analysis (IFRA)," *IEEE. Trans. Computer Aided Design Integrated Circuits Systems*, Vol. 28, No. 10, pp. 1545-1558, 2009.

[35] S.-B. Park, A. Bracy, P. Wang, and S. Mitra, "BLoG: Post-Silicon Bug Localization in Processors Using Bug Localization Graph", *Proc. IEEE/ACM Design Automation Conf.,* pp. 368-373, 2010.

[36] P. Patra, "On the Cusp of a Validation Wall," *IEEE Design & Test of Computers*, Vol. 24, No. 2, pp. 193-196, 2007.

[37] R. Raina, and R. Molyneaux, "Random Self-Test Method Applications on PowerPC™ Microprocessor Cache," *Proc. ACM/IEEE Great Lakes Symp. VLSI,* pp. 222-229, 1998.

[38] K. Reick, "Post-Silicon Debug - DAC Workshop on Post-Silicon Debug: Technologies, Methodologies, and Best-Practices," *IEEE/ACM Design Automation Conf.,* 2012.

[39] F. Rogin, T. Klotz, G. Fey, R. Drechsler, and S. Rulke, "Automatic Generation of Complex Properties for Hardware Designs," *Proc. IEEE/ACM Design Automation Conf.,* pp. 545-548, 2008.

[40] S. Vasudevan, *et al.,* "GoldMine: Automatic Assertion Generation Using Data Mining and Static Analysis," *Proc. IEEE/ACM Design, Automation, Test in Europe Conf.,* pp. 626 – 629, 2010.

[41] M. N. Velev, "Collection of High-Level Microprocessor Bugs from Formal Verification of Pipelined and Superscalar Designs," *Proc. IEEE Intl. Test Conf*., pp. 138-147, 2003.

[42] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proc. ACM/IEEE Intl. Symp. Computer Architecture*, pp. 24-36, 1995.

[43] I. Wagner, and V. Bertacco, "Reversi: Post-Silicon Validation System for Modern Microprocessors," *Proc. IEEE Intl. Conf. Computer Design,* pp. 307-314, 2008.

[44] S. Yerramilli, "Addressing Post-Silicon Validation Challenges: Leverage Validation & Test Synergy," Keynote, *IEEE Intl. Test Conf.*, 2006.