Overcoming Post-Silicon Validation Challenges Through Quick Error Detection (QED)

David Lin[1], Ted Hong[1], Yanjing Li[1], Farzan Fallah[1], Donald S. Gardner[3], Nagib Hakim[3], Subhasish Mitra[1,2]
[1]Department of EE and [2]Department of CS
Stanford University, Stanford, CA, USA
[3]Intel Corporation
Santa Clara, CA, USA

Abstract

Existing post-silicon validation techniques are generally *ad hoc*, and their cost and complexity are rising faster than design cost. Hence, systematic approaches to post-silicon validation are essential. Our research indicates that many of the bottlenecks of existing post-silicon validation approaches are direct consequences of very long error detection latencies. Error detection latency is the time elapsed between the activation of a bug during post-silicon validation and its detection or manifestation as a system failure. In our earlier papers, we created the Quick Error Detection (QED) technique to overcome this significant challenge. QED **systematically** creates a wide variety of post-silicon validation tests to detect bugs in processor cores and uncore components of multi-core System-on-Chips (SoCs) very quickly, i.e., with very short error detection latencies. In this paper, we present an overview of QED and summarize key results: 1. Error detection latencies of "typical" post-silicon validation tests can range up to billions of clock cycles. 2. QED shortens error detection latencies by up to 6 orders of magnitude. 3. QED enables 2- to 4-fold improvement in bug coverage. QED does not require any hardware modification. Hence, it is readily applicable to existing designs.

Keywords—Debug, Post-Silicon Validation, Quick Error Detection, Testing, Verification

1. INTRODUCTION

During *post-silicon validation*, one or more manufactured integrated circuits (ICs) are tested in actual system environments to detect and fix design flaws (bugs). Design bugs can be broadly classified into two categories:

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

2. *Electrical bugs* arising from subtle interactions between a design and its electrical state. Examples include signal integrity (cross-talk, power-supply noise), thermal effects, and process variations. Electrical bugs often manifest themselves only under specific operating conditions, e.g., voltage, frequency, and temperature corners [Patra 07].

Post-silicon validation is crucial because pre-silicon verification alone is inadequate. Traditional pre-silicon verification is too slow for today's complex designs. Moreover, pre-silicon verification does not adequately address electrical bug scenarios that appear after ICs are manufactured. Fundamental barriers to silicon CMOS scaling further magnify these challenges. It is well known that the classical Dennard scaling of silicon CMOS circuits has slowed down [Bohr 09]. As a result, improvements in energy-efficiency and performance of computing systems come at the price of increased complexity: multiple processor cores, accelerators, uncore components (e.g., cache controllers, memory controllers), adaptive power, thermal and reliability management, and increased levels of heterogeneous integration. These characteristics significantly exacerbate post-silicon validation challenges [Adir 11, Mitra 10, Singerman 11].

Without scalable ways of taming such increasing levels of complexity, future systems can end up being highly vulnerable to design bugs that might jeopardize correct operation and introduce security vulnerabilities [Bose 12]. Such effects are already visible in today's System-on-Chips (*SoCs*) [Kubicki 07, Shankland 05, Shimpi 11].

Existing post-silicon practices are generally *ad hoc*, and their costs and complexity are rising faster than design cost [Abramovici 06, Adir 10, Keshava 10, Yerramilli 06]. Today's *ad hoc* post-silicon validation approaches remind us of *ad hoc* manufacturing testing techniques in the 1960s-70s [Abramovici 90, McCluskey 86, Miczo 86]. Examples include *ad hoc* testability techniques, various controllability and observability heuristics (testability measures), and (mostly) manual failure analysis. Similar examples in the context of post-silicon validation include various trace buffer insertion strategies [Abramovici 06, Basu 11], heuristics such as restoration ratio and its derivatives [Ko 08, Liu 09], and the use of assertions for post-silicon validation [Boule 07]. As noted in [Bentley 01], assertions have to be carefully crafted, and it is difficult to keep them up-to-date and to validate their correctness. Moreover, the lack of a comprehensive set of bug benchmarks hinders research progress. Benchmarks are crucial for quantitative comparison of the effectiveness of various techniques.

Manufacturing testing underwent a major transformation from *ad hoc* to systematic techniques, thanks to several innovations:

1. Scan design for testability [Eichelberger 77, Williams 73] together with advances in test pattern generation, test coverage metrics, and fault diagnosis of combinational circuits.

2. Test compression and logic BIST [Bardell 87, Hamzaoglu 99, Jas 98, Koenemann 91, Mitra 04, Touba 01].

3. Test chip experiments [Ma 95, Maxwell 93, McCluskey 00, 04, Nigh 97] that enabled quantitative comparison of the effectiveness of various testing techniques.

Similar innovations are required to advance the field of post-silicon validation through the creation of systematic approaches. To achieve this objective, it is essential to understand the key factors that make post-silicon validation challenging.

Post-silicon validation generally involves three activities:

1. Detecting a problem by applying proper stimuli.

2. Localizing the problem to a small region inside the chip.

3. Fixing the problem through software patches, circuit edits, or silicon re-spin.

The effort to localize the problem from an observed failure often dominates post-silicon validation costs [Abramovici 06, Amyeen 09, Josephson 06]. Existing bug localization techniques suffer from two major challenges: 1. Failure reproduction, i.e., returning the system to an error-free state and re-executing the failure-causing stimuli (including instruction sequences, interrupts, voltage and temperature conditions) to reproduce the failure; and, 2. Simulation to obtain expected system response. It is very difficult to reproduce complex bug scenarios, and existing simulation techniques are very slow for large SoCs.

Our experience shows that *error detection latency*, the time elapsed between the occurrence of an error triggered by a bug and its manifestation as an observable failure, is crucial for effective post-silicon validation [Hong 10, Lin 12, Reick 12]. Long error detection latencies limit the effectiveness of existing bug localization techniques. Bugs with error detection latencies longer than a few thousand clock

978-3-9815370-0-0/DATE13/©2013 EDAA

cycles are highly challenging because it is extremely difficult to trace too far back in history for bug localization. In contrast, bugs in complex SoCs can result in very long error detection latencies of several millions to billions of clock cycles [Hong 10, Lin 12, Reick 12]. As shown in [Lin 12], "typical" post-silicon validation tests consisting of end loop checks, end result checks, store readback checks, checkpointing, deadlock detection, and livelock detection are inadequate in shortening such long error detection latencies. Systematic approaches for creating post-silicon validation tests with improved (i.e., reduced) error detection latencies are crucial.

The Quick Error Detection (*QED*) technique [Hong 10, Lin 12] systematically creates post-silicon validation tests with very short error detection latencies for bugs in both uncore components and processor cores of multi-core SoCs. Experimental results obtained from hardware platforms based on quad-core Intel® Core™ i7 processors and simulation results obtained from a complex OpenSPARC T2-like multi-core SoC [OpenSPARC] demonstrate:

1. 4 to 6 orders of magnitude improvement in error detection latencies for bugs inside processor cores and uncore components. The error detection latencies of QED tests are within a few hundred clock cycles in most cases.
2. 2- to 4-fold improvement in coverage of bug scenarios detected.

QED can be implemented entirely in software with no hardware modifications. Hence, QED can be readily applied to existing designs.

In Sec. 2, we discuss difficult bug scenarios and show how "typical" post-silicon validation tests end up with very long error detection latencies. Section 3 presents an overview of the QED technique. Section 4 presents hardware results obtained from an Intel® Core™ i7 platform and simulation results for an OpenSPARC T2-like multi-core SoC. We conclude in Sec. 5.

2. "DIFFICULT" BUG SCENARIOS AND LONG ERROR DETECTION LATENCIES

A comprehensive list of bug scenarios is critical for understanding the limitations of existing post-silicon validation techniques, and for evaluating new approaches. Toward this end, we compiled a list of realistic bug scenarios by analyzing reports of "difficult" bugs detected during validation of industrial multi-core SoCs. These are primarily logic bugs involving cache controllers, memory controllers, on-chip networks, and processor cores. These bug scenarios are considered "difficult" because of very long debug times as indicated in bug reports. The details of these bug scenarios are presented in [Lin 12]. Given our discussion in Sec. 1 on the increased complexity of computing systems to achieve high performance and energy efficiency, it is not surprising that many difficult bug scenarios in [Lin 12] involve uncore components of SoCs. Other bug scenarios published in research literature include [Constantinides 08, DeOrio 08, 09, Ho 95, Van Campenhout 00, Velev 03].

We use a bug scenario in Fig. 1 (scenario 8C taken from [Lin 12]) to illustrate long error detection latencies incurred by "typical" post-silicon validation tests. Consider a multi-core SoC where each processor core has its private L1 cache with write-through policy, and a shared L2 cache with write-through and write-allocate policy. As shown in Fig. 1a, when Core 3 performs a store operation to a memory location *A*, if *A* is not already cached, a new cache line needs to be allocated for *A* in the shared cache (i.e., the cache needs to pick a free cache line for memory location *A* or evict an existing cache line if a free cache line is not available). However, due to a bug in the shared cache, a new cache line is not allocated for *A*. Instead, an existing cache line is used (i.e., one that caches the value of an **arbitrary** memory location *B*). As a result, the store operation overwrites the existing cache line and corrupts the cached value of an **arbitrary** memory location (i.e., *B)* and marks it as modified. Note that, the value of *A* is stored correctly in main memory. After a very long time, Core 7 loads the corrupted value corresponding to memory location *B* from the shared cache (since the cache line is in modified state in the shared cache) and uses this corrupted value in its computation. Consequently, Core 7 produces incorrect results. As shown in Fig. 1a, tests that rely on end result checks, which check for expected output values upon test completion, end up with very long error detection latencies.

Following the above example, the corrupted value read by Core 7 can result in a livelock / deadlock (Fig. 1b). Techniques for detecting livelocks / deadlocks [Bayazit 05, Chandy 83] are inadequate in shortening error detection latencies because the error detection latency is already very long when the deadlock occurs (i.e., long after the cached value corresponding to memory location *B* is corrupted).

Self-checking tests that focus on bugs inside processor cores are not sufficient either. Such tests check the results of instructions by comparing against a golden model [Aharon 95] or by using other instructions executing on the processor cores [Raina 98, Wagner 08]. However, since the bug only corrupts the cached value corresponding to an arbitrary memory location *B* (and not the value of memory location *A*), Core 3's self checks do not detect the error. It takes a very long time for Core 7's self checks to detect the error in the cached value of memory location *B* (Fig. 1c).

Another self-checking test variant, referred to as *store readback test* (Fig. 1d), performs a load operation on the same core that performed the store operation (Core 3 in this case) to check if the loaded value matches the stored value. As discussed above, the store operation does not corrupt the value of memory location *A*. Hence, the store readback check passes. It takes a very long time before Core 7 performs a load from memory location *B* and detects the error.

QED overcomes these challenges through a variety of transformations that systematically insert various checks to reduce error detection latencies.

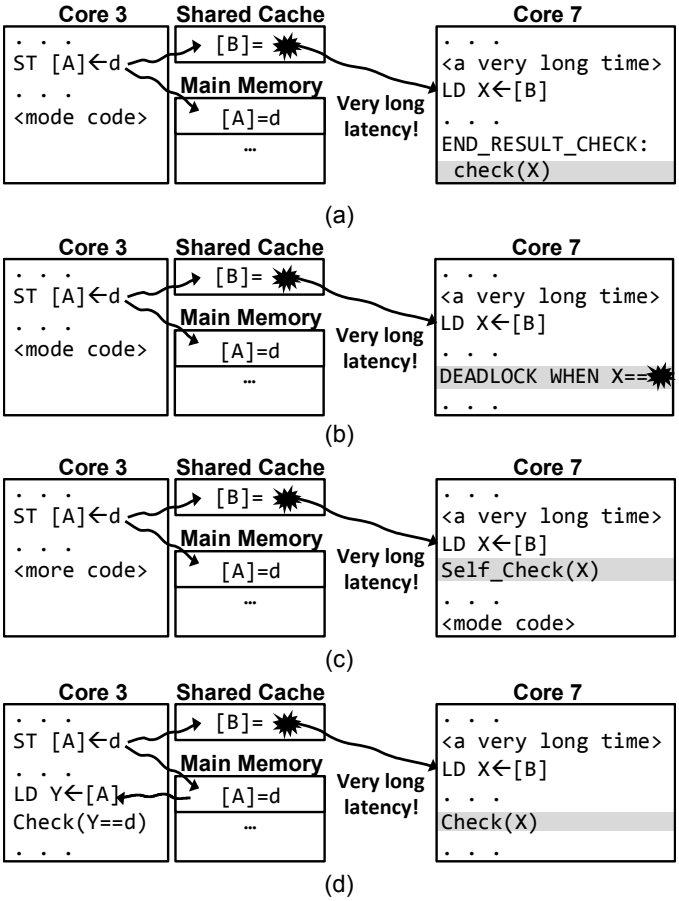

Figure 1. Long error detection latencies of typical post-silicon validation tests with (a) end result checks, (b) deadlock detection, (c) self checking test targeting processor cores, and (d) store readback test.

3. QED: Quick Error Detection

QED systematically transforms existing validation tests into new tests, called *QED tests*, with bounded error detection latencies. The corresponding transformations are referred to as *QED transformations*. As detailed in [Hong 10, Lin 12], a wide variety of QED transformations are available. The following subsections present an overview of software-only QED transformations that are readily applicable to existing designs because they don't require any hardware changes. Hardware design techniques to support some of the QED transformations are discussed in [Hong 10].

3.1 Software-Only QED Transformations

Suppose that we are given a post-silicon validation test (referred to as an "original" test in this paper) and a target error detection latency expressed in terms of the number of clock cycles (typically determined by the specific bug localization technique used). Software-only QED transformations create QED tests by systematically inserting targeted instructions throughout the original test code to perform checks on registers, variables, and memory locations such that the error detection latency target is satisfied.

A variety of software-only QED transformations can be used to create QED tests [Hong 10, Lin 12]. These transformations are inspired by Software Implemented Hardware Fault Tolerance (SIHFT) techniques [Lovelette 02]. However, as discussed in [Hong 10], special considerations are necessary to ensure that the error detection latency constraints are satisfied. We specifically append a "V" to some of the SIHFT techniques to alert the reader about error detection latency considerations that must be taken into account.

Examples of QED transformations include: Error Detection by Duplicated Instructions for Validation or EDDI-V (inspired by EDDI [Oh 02a]), EDDI-V with store readback (a variant of EDDI-V where, in addition to the EDDI-V transformation, values stored in memory are loaded and checked to determine if the loaded values match the corresponding stored values), EDDI-V with diversity (a variant of EDDI-V which incorporates diversity inspired by ED^{4}I [Oh 02b]), Control Flow Checking using Software Signatures for Validation or CFCSS-V (inspired by CFCSS [Oh 02c]), Control Flow Tracking using Software Signatures for Validation or CFTSS-V (a variant of CFCSS-V where each block of instructions executed on the processor core is tracked by a special software signature inserted into the test code but no control flow checking is performed), and Proactive Load and Check (PLC) [Lin 12]. Figure 2 illustrates some of these QED transformations. Algorithms for applying such QED transformations are discussed in [Hong 10, Lin 12].

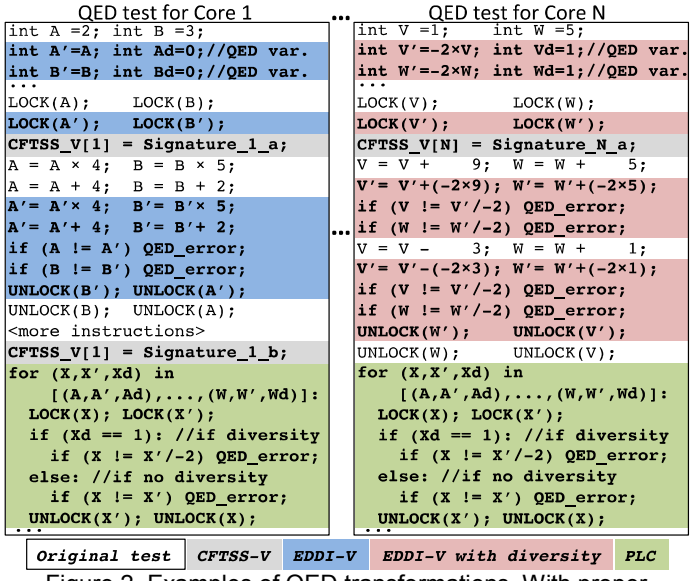

Figure 2. Examples of QED transformations. With proper locking policy, LOCK and UNLOCK of A', B', V', W', and X' may not be necessary.

While checks that are inserted as part of QED (e.g., in Fig. 2) may be viewed as assertions, QED is different from validation techniques that try to pick and choose subsets of assertions (e.g., through assertion mining) for a given design. The basic principles of QED are rooted in SIHFT techniques where bugs are detected quickly through extensive time-redundant execution (with diversity) and fine-grained checking.

3.2 SoC Components Targeted by Software-Only QED

Software-only QED techniques are applicable for the following categories of SoC components: 1. software-programmable components such as processor cores, GPUs, and DSPs; and, 2. uncore components with registers or memories that can be read and written by the processor cores via instructions (i.e., load / store or I/O instructions). Examples include caches, memories, interconnect networks, network controllers, various accelerators, and some parts of I/O interfaces and power / thermal management controllers.

3.3 Tests Targeted by Software-Only QED

QED is applicable for a wide variety of validation tests: 1. Tests written in high-level languages such as C, and C++. Examples include benchmark programs, scientific applications, and open source applications. These tests can be transformed into QED tests at compile time. 2. Tests written using assembly code. Examples include targeted tests written by validation engineers or those generated by random instruction test generators. These tests can be transformed into QED tests at assembly time. 3. Tests where only the compiled / assembled binary executables are available. Examples include closed-source applications, closed-source benchmarks, and 3rd party proprietary tests. Binary disassemblers can be used to disassemble these tests and QED transformations can then be applied to the resulting assembly code. However, if the test contains indirect branch instructions (i.e., branches to addresses stored in registers), then the register values used by these instructions must be determined prior to the application of QED transformations. This is to ensure that indirect branch instructions still branch to the correct instructions in the QED tests. It is possible to determine these register values because the inputs of post-silicon validation tests may be known *a priori* [Bentley 01].

3.4 Bug Types Targeted by Software-Only QED

The unique insight behind the effectiveness of QED for post-silicon validation is as follows: many "difficult" bugs have characteristics similar to transient hardware errors even though they may be caused by logic design mistakes. Such characteristics of "difficult" bugs make them hard to reproduce, and hence localize, using conventional techniques (as discussed in Sec. 1). These difficult characteristics make these bugs ideal candidates to be quickly detected by QED through time-redundant execution (with implicit or explicit diversity) and checking.

Software-only QED tests are capable of detecting bugs that result in erroneous architectural states (i.e., registers and memories). For example, a bug in a cache coherence protocol can corrupt a value in one processor core's private cache. Such bugs can be quickly detected by QED [Lin 12]. However, some bugs may result in systems hangs. For example, congestion on a SoC's interconnect network may drop a cache read request issued by a processor core. As a result, the processor core waits for a very long time to receive a response to its cache read request. QED techniques such as CFCSS-V and CFTSS-V, which assign software signatures to blocks of instructions, are highly effective in localizing such bugs.

We have not yet studied the applicability of QED for bugs that do not have any functional effect on the system, e.g., performance related bugs [Patra 07].

3.5. QED Coverage Considerations

It is possible for software-only QED transformations to introduce "intrusiveness" resulting in situations where the QED test may not detect a bug otherwise detected by the original test. For example, a very specific sequence of load and store instructions may

be needed to activate a particular cache coherence protocol bug. Additional instructions inserted by QED may disrupt this sequence.

Empirically, we have not observed any situation yet where the original test is able to detect a bug not detected by QED (Sec. 4). However, the flexibility of QED transformations enables several approaches to minimize possible intrusiveness while detecting bugs quickly. From a given original test, one can create a group of QED tests referred to as *QED family tests*. Each test in the family is obtained by applying a different set of QED transformation parameters. Several strategies for adjusting such parameters exist [Hong 10, Lin 12]. For example, one can create QED family tests where only a selected subset of instructions and variables (in the original test) may be transformed to create each QED test in the family. One can also create QED family tests where each test has a different value of *Inst_min* (*Inst_max*) parameters, defined as the minimum (maximum) number of instructions from the original test that must execute consecutively before executing any instructions inserted by QED [Hong 10]. For example, in Fig. 2, the QED test for Core 1 has *Inst_min* (and *Inst_max*) of 4 instructions for the EDDI-V transformation (i.e., QED code is only inserted in between every 4 consecutive instructions in the original test). The QED test for Core N has *Inst_min* (and *Inst_max*) of 2 instructions for EDDI-V with diversity transformation (i.e., QED code is only inserted in between every 2 consecutive instructions in the original test).

If the *test intent* is known, then it can be conveyed to the QED transformation tool as constraints. For example, certain sections of a test may contain specific sequences of store and load instructions to trigger a particular class of cache coherence protocol bugs. Preprocessor directives may be used to ensure that QED transformations preserve these constraints. In an extreme case, QED transformations may not modify those specific sections of the test.

4. QED Results

We demonstrate the effectiveness of software-only QED transformations using results obtained from hardware experiments on an Intel® Core™ i7 system, and simulation experiments using an OpenSPARC T2-like SoC model. In this section, we present an overview. Detailed results can be obtained from [Hong 10, Lin 12].

4.1 Hardware Results

We performed hardware error injections on a quad-core Intel® Core™ i7 processor platform (Fig. 3) by varying the operating voltage and frequency values of a processor core. Such an experiment was possible due to our access to a custom-designed system configuration with a special debug port and a temperature controller to keep the chip package at a fixed temperature. The detailed experimental methodology can be found in [Hong 10].

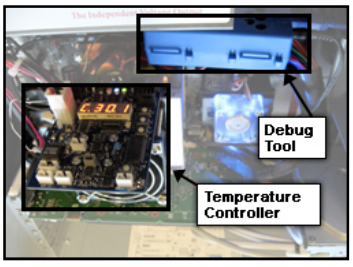

Figure 3. Quad-core Intel® Core™ i7 system with a DX58SO motherboard, a temperature controller, and a proprietary debug tool.

The results are shown in the histogram in Fig. 4 for the Linpack benchmark test [Dongarra 03]. The vertical axis represents the percentage of errors detected (normalized to the number of errors detected by QED). The horizontal axis represents error detection latencies. The original Linpack test contains "end result checks", which check the result matrix against a pre-computed (expected) result matrix stored in memory. We transformed the original Linpack test into a QED test using the EDDI-V transformation at the C source code-level. The results demonstrate that QED significantly improves error detection latencies by six orders of magnitude, and simultaneously improves error coverage. The following observations can be made from the results.

Observation 1: QED significantly reduces error detection latencies by six orders of magnitude compared to the original test with end result checks. With QED, error detection latencies are reduced from billions of clock cycles to a few thousand clock cycles or less. For 86% of the cases, the error detection latencies of QED tests are less than 1,000 clock cycles. The remaining error detection latencies are longer than 1,000 clock cycles because we performed QED transformation only at the C source code-level, and not inside any library function. Errors that occur during the execution of a library function are not detected by QED instrumentation at the C source code-level until that library function is exited.

Observation 2: QED detects errors that would otherwise remain undetected by the original test. QED results in a significant (4-fold) improvement in the ability to detect errors.

Observation 3: QED detects all errors detected by the original test; i.e., the incorporation of QED transformation does not adversely impact the ability of the test to detect errors.

These results are further confirmed by Shmoo experiments shown in Fig. 5. QED enables unique coverage enhancement while significantly improving error detection latency. This is demonstrated by the voltage and frequency operating point in Fig. 5 (labeled with a star) that passes the original test with end result checks but results in detected errors with QED. QED tests are valid tests, i.e., they do not bring the system into illegal states. Therefore, the errors detected by QED tests are actual errors in the system. Moreover, there exists no point on the Shmoo plot in Fig. 5 for which the QED test passed but the original test did not. This empirically establishes the fact that the QED test continues to activate and detect errors that are detected by the original test.

The C source code-level EDDI-V QED transformation used for the experiments in Figs. 4 and 5 does result in longer test execution times (approximately 2-fold). However, the overall debug time, and hence productivity, can improve drastically due to significantly shorter error detection latencies.

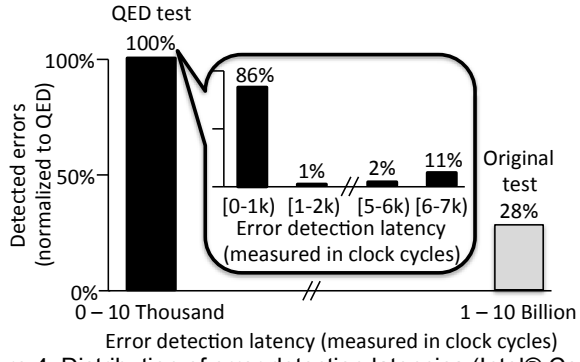

Figure 4. Distribution of error detection latencies (Intel® Core™ i7-based hardware platform) for the original Linpack with end result checks (Original test) and Linpack with QED (QED test).

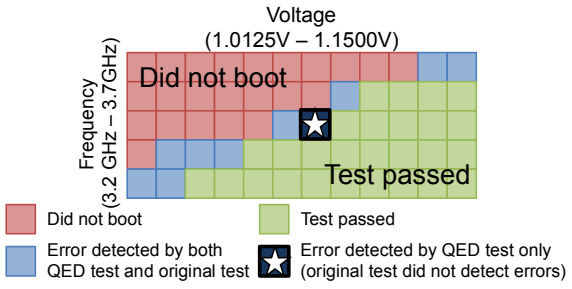

Figure 5. Linpack Shmoo plot for Intel® Core™ i7-based hardware platform, showing the errors detected by QED and original tests over a range of frequency and voltage points.

4.2 Simulation Results

We used the Multifacet General Execution-driven Multiprocessor Simulator (GEMS) [Martin 05] to simulate difficult bug scenarios in [Lin 12] (discussed in Sec. 2) on an OpenSPARC T2-like multi-core SoC [OpenSPARC] (details in [Lin 12]). This SoC corresponds to a 500 million-transistor design with 8 processor cores, 64 threads, private split L1 data and instruction caches, crossbar-based interconnects, 8-way banked shared L2, and 4 memory controllers. Figure 6 shows a block diagram of the simulated system.

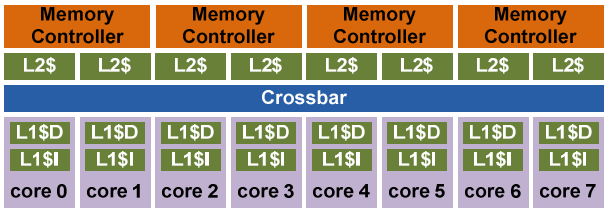

Figure 6. OpenSPARC T2-like multi-core SoC [OpenSPARC].

We used a combination of two software-only QED transformations: PLC transformation to detect uncore bugs and EDDI-V transformation to detect bugs inside processor cores. These transformations were applied at the C source code-level to create QED family tests (details in [Lin 12]). The transformations were agnostic (i.e., not specifically tailored) to the bug scenarios in the simulated system. The results are summarized for the FFT program from the SPLASH-2 benchmark suite [Woo 95] in Fig. 7a, and for a proprietary industrial post-silicon validation test targeting memory bugs in Fig. 7b. In Figs. 7a and 7b, the horizontal axis shows the error detection latencies, and the vertical axis shows the cumulative percentage of bugs detected (i.e., bug detection coverage) out of the bug scenarios in [Lin 12]. In Figs. 7a and 7b, the *Original tests* are instrumented with "end result checks" only. The following observations can be made from the results.

Observation 4: Post-silicon validation tests created using QED (EDDI-V + PLC) shorten error detection latencies by several orders of magnitude for all bug scenarios in [Lin 12]. Error detection latencies of the original tests can be extremely long: several millions or billions of cycles. QED tests with only EDDI-V transformation can result in long error detection latencies for uncore bugs. In contrast, QED tests with EDDI-V + PLC have error detection latencies of only a few hundred cycles for most bug scenarios.

Observation 5: Post-silicon validation tests created using QED detect more bug scenarios than the original tests. There is not a single bug scenario that the original tests detected but the QED tests did not.

Observation 6: QED tests detect up to 2-fold more bug scenarios that would otherwise remain undetected by the original tests.

The execution times of QED family tests with PLC and EDDI-V can be significantly longer than the original tests (details in [Lin 12]). However, in post-silicon validation, reducing error detection latency is very important because debug time, rather than test execution time, often dominates overall post-silicon validation costs [Josephson 06]. Therefore, some execution time penalties can be tolerated if error detection latencies significantly improve. If the long execution time is a concern, strategies to optimize the execution times of QED family tests are discussed in [Lin 12]. It has also been demonstrated in [Lin 12] that the error detection latency and coverage benefits of QED tests are direct consequences of QED transformations (EDDI-V + PLC in this case) and not because of longer runtimes of QED tests compared to the original test.

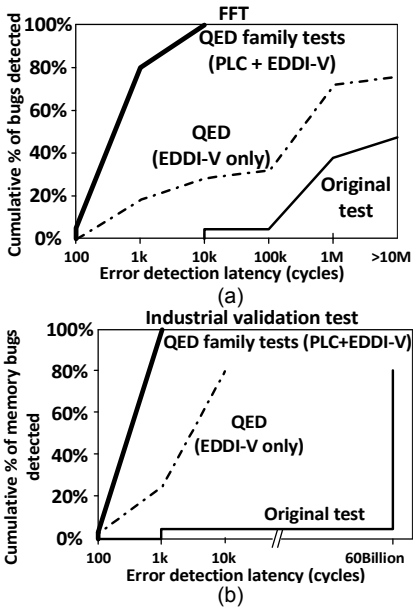

Figure 7. Error detection latencies and coverage of post-silicon validation tests. (a) FFT from SPLASH-2 benchmark suite and (b) Industrial validation test.

5. Conclusion

QED overcomes post-silicon validation challenges by **systematically** creating post-silicon validation tests that detect bugs very quickly, i.e., with very short error detection latencies. Experimental results obtained from quad-core Intel® Core™ i7-based hardware platforms and simulation results on a complex OpenSPARC T2-like multi-core SoC demonstrate up to 6 orders of magnitude improvement in error detection latencies and 2- to 4-fold improvement in bug coverage simultaneously. Such short error detection latencies can significantly improve post-silicon validation productivity. Moreover, QED does not require any hardware changes and is readily applicable to existing post-silicon validation flows.

Several opportunities exist to further enhance existing validation methodologies using QED. Examples include: 1. Automated bug localization by analyzing QED checks. 2. Systematic techniques for synthesizing hardware support for QED. 3. Effective emulation techniques using QED. 4. Use of QED for system-level identification of failing ICs and for root-causing No-Trouble-Found (NTF) ICs [Conroy 05]. 5. QED techniques for mixed-signal design blocks.

6. Acknowledgment

This research was supported in part by FCRP, GSRC, 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 Jagannath Keshava, Rahima Mohammed and Keshavan Tiruvallur of Intel for their valuable inputs.

7. References

[Abramovici 90] Abramovici, M., M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.

[Abramovici 06] Abramovici, M., "A Reconfigurable Design-for-Debug Infrastructure for SoCs," *Proc. IEEE/ACM Design Automation Conf.*, pp. 7-12, 2006.

[Adir 10] Adir, A., *et al.,* "Reaching Coverage Closure in Post-Silicon Validation," *Proc. of 6th Haifa Verification Conf.,* LNCS 6504, pp. 60-74. Springer-Verlag, 2010.

[Adir 11] Adir, A., *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.

[Aharon 95] Aharon, A., *et al*., "Test Program Generation for Functional Verification of PowerPC Processors in IBM," *Proc. IEEE/ACM Design Automation Conf.*, pp 279-285, 1995.

[Amyeen 09], Amyeen, M. E., S. Venkataraman, and M. W. Mak, "Microprocessor System Failures Debug and Fault Isolation Methodology," *Proc. IEEE Intl. Test Conf.*, pp. 1-10, 2009.

[Bardell 87] Bardell, P.H., W.H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*, Wiley, 1987.

[Basu 11] Basu, K., and P. Mishra, "Efficient Trace Signal Selection for Post-Silicon Validation and Debug," *Proc. IEEE Intl. Conf. on VLSI Design*, pp. 352-357, 2011.

[Bayazit 05] Bayazit, A. A., and S. Malik, "Complementary Use of Runtime Validation and Model Checking," *Proc. IEEE/ACM Intl. Conf. on Computer-aided Design*, pp. 1049-1056, 2005.

[Bentley 01] Bentley, B., and R. Gray, "Validating the Intel Pentium 4 Processor." *Intel Technology Journal*, Vol. 5 Issue 1, pp. 1-8, February, 2001.

[Bohr 09] Bohr, M., "The New Era of Scaling in an SoC World," *IEEE Solid-State Circuits Conf.,* pp 23-28, 2009.

[Bose 12] Bose, P., *et al.,* "Power Management of Multi-Core Chips: Challenges and Pitfalls," *Proc. IEEE/ACM Design, Automation and Test in Europe Conf.,* pp. 977-982, 2012.

[Boule 07] Boule, M., 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.

[Chandy 83] Chandy, K. M., J. Misra, and L. M. Haas, "Distributed Deadlock Detection," *ACM Trans. on Computer Systems,* Vol. 1, Issue 2, pp. 144-156, May 1983.

[Conroy 05] Conroy, Z., G. Richmond, X. Gu, and B. Eklow, "A Practical Perspective on Reducing ASIC NTFs," *Proc. IEEE Intl. Test Conf.*, pp. 1-7, 2005.

[Constantinides 08] Constantinides, K., O. Mutlu, and T. Austin, "Online Design Bug Detection: RTL Analysis, Flexible Mechanisms, and Evaluation," *Proc. IEEE/ACM Intl. Symp. on Microarchitecture*, pp. 282-293, 2008.

[DeOrio 08] DeOrio, A., A. Bauserman, and V. Bertacco, "Post-Silicon Verification for Cache Coherence," *Proc. IEEE Intl. Conf. on Computer Design*, pp. 348-355, 2008.

[DeOrio 09] DeOrio, A., I. Wagner, and V. Bertacco, "DACOTA: Post-silicon Validation of the Memory Subsystem in Multi-Core Designs," *Proc. IEEE Intl. Symp. on High-Performance Computer Architecture*, pp. 405-416, 2009.

[Dongarra 03] Dongarra, J., P. Luszczek, and A. Petitet, "The LINPACK Benchmark: Past, Present and Future," *Concurrency and Computation: Practice and Experience.* Vol. 15, Issue 9, pp. 803-820, 2003.

[Eichelberger 77] Eichelberger, E.B., and T.W. Williams, "A Logic Design Structure for LSI Testability," *Proc. IEEE/ACM Design Automation Conf.,* pp. 462-468. 1977.

[Hamzaoglu 99] Hamzaoglu, I., and J.H. Patel, "Reducing Test Application Time for Full Scan Embedded Cores," *Proc. Intl. Symp. Fault-Tolerant Computing*, pp. 260-267, 1999.

[Ho 95] Ho, R.C., *et al*. "Architecture Validation for Processors," *Proc. ACM/IEEE Intl. Symp. on Computer Architecture*, pp. 404-413, 1995.

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

[Jas 98] Jas, A., and N.A. Touba, "Test Vector Decompression via Cyclical Scan Chains and its Application to Testing Core-base Designs," *Proc. IEEE Intl. Test Conf.,* pp. 458-464, 1998.

[Josephson 06] Josephson, D., "The Good, the Bad, and the Ugly of Silicon Debug," *Proc. IEEE/ACM Design Automation Conf*., pp. 3-6, 2006.

[Keshava 10] Keshava, J., 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.

[Ko 08], Ko, and H.F., Nicolici, "Automated Trace Signals Identification and State Restoration for Improving Observability in Post-Silicon Validation," *Proc. IEEE/ACM Design, Automation and Test in Europe Conf.,* pp. 1298-1303, 2008.

[Koenemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," *Proc. IEE European Test Conference*, pp. 237-242, 1991

[Kubicki 07] Kubicki, K., "Understanding AMD's 'TLB' Processor Bug," *Daily Tech,* http://www.dailytech.com/Understanding++AMDs+TLB+Processor+Bug/article9915.htm, 2007.

[Lin 12] Lin, D., *et al.*, "Quick Detection of Difficult Bugs for Effective Post-Silicon Validation," *Proc. IEEE/ACM Design Automation Conf.,* pp. 561-566, 2012.

[Liu 09] Liu, X., and Q. Xu, "Trace Signal Selection for Visibility Enhancement in Post-Silicon Validation," *Proc. IEEE/ACM Design, Automation and Test in Europe Conf.*, pp. 1338-1343, 2009.

[Lovellette 02] Lovellette, M.N. *et al.*, "Strategies for Fault-tolerant Space-based Computing: Lessons Learned from the ARGOS Testbed," *Proc. Aerospace Conf.*, pp. 5-2109-5-2119, 2002.

[Ma 95] Ma, S.C., P. Franco, and E.J. McCluskey, "An Experimental Chip to Evaluate Test Techniques Experiment Results," *Proc. IEEE Intl. Test Conf.,* pp. 663-672, 1995.

[Martin 05] Martin, M., *et al.*, "Multifacet's General Execution-Drive Multiprocessor Simulator (GEMS) Toolset," *ACM SIGARCH Computer Architecture News*, Vol. 33, Issue 4, pp. 92-99, November, 2005.

[Maxwell 93] Maxwell, P., and R. Aitken, "Test Sets and Reject Rates: All Fault Coverages are Not Created Equal," *IEEE Design & Test of Computers*, Vol. 10, No. 1, pp. 42-51, 1993.

[McCluskey 86] McCluskey, E.J., *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[McCluskey 00] McCluskey, E.J. and C.W. Tseng, "Stuck-at Fault versus Actual Defects", *Proc. IEEE Intl. Test Conf.*, pp. 336-343, 2000.

[McCluskey 04] McCluskey, E.J., *et al.*, "ELF-Murphy Data on Defects and Tests Sets," *Proc. IEEE VLSI Test Symposium*, pp. 16-22, 2004.

[Miczo 86] Miczo, A., *Digital Logic Testing and Simulation,* Harper & Row, NY, NY, 1986.

[Mitra 04] Mitra, S., and K.S. Kim, "X-Compact: An Efficient Response Compaction Technique," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 23, Issue 3, pp. 421-432, 2004.

[Mitra 10] Mitra, S., S.A. Seshia, and N. Nicolici, "Post-Silicon Validation Opportunities, Challenges and Recent Advances," *Proc. IEEE/ACM Design Automation Conf*., pp. 12-17, 2010.

[Nigh 97] Nigh, P., *et al.*, "So What Is an Optimal Test Mix: A Discussion of the SEMATECH Methods Experiment," *Proc. IEEE Intl. Test Conf.*, pp. 1037-1038, 1997.

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

[Oh 02b] Oh, N., S. Mitra, and E. J. McCluskey, "ED^{4}I: Error Detection by Diverse Data and Duplicated Instructions," *IEEE Trans. on Computers*, Vol. 51, Issue 2, pp. 180-199, 2002.

[Oh 02c] Oh, N., P. P. Shirvani, and E. J. McCluskey, "Control Flow Checking by Software Signatures," *IEEE Trans. on Reliability,* Vol. 51, Issue 1, pp. 111-122, 2002.

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

[Patra 07] Patra, P., "On the Cusp of a Validation Wall," *IEEE Design & Test of Computers*, Vol. 24, Issue 2, pp. 193-196, March, 2007.

[Raina 98] Raina, R., and R. Molyneaux, "Random Self-Test Method Applications on PowerPC^{TM} microprocessor cache," *Proc. ACM/IEEE Great Lakes Symp. VLSI*, pp. 222-229, 1998.

[Reick 12], Reick, K., "Post-Silicon Debug," DAC Workshop on Post-Silicon Debug: Technologies, Methodologies, and Best-Practices. *IEEE/ACM Design Automation Conf., 2012.*

[Shankland 05] Shankland, S., "Intel Pushes Back Itanium Chips, Revamps Xeon," *CNET,* http://news.cnet.com/Intel-pushes-back-Itanium-chips,-revamps-Xeon/2100-1006_3-5911316.html, 2005.

[Shimpi 11] Shimpi, A. L., "The Source of Intel's Cougar Point SATA Bug," *AnandTech*, http://www.anandtech.com/show/4143/the-source-of-intels-cougar-point-sata-bug, 2011.

[Singerman 11] Singerman, E., Y. Abarbanel, and S. Baartmans, "Transaction Based Pre-To-Post Silicon Validation," *Proc. IEEE/ACM Design Automation Conf.*, pp. 564-568, 2011.

[Touba 01] Touba, N.A., and E.J. McCluskey, "Bit Fixing in Pseudorandom Sequences for Scan BIST," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pp. 545-555, 2001.

[Van Campenhout 00] Van Campenhout, D., *et al.*, "Collection and Analysis of Microprocessor Design Errors," *IEEE Design & Test of Computers,* Vol. 17, Issue 4, pp. 51-60, Oct-Dec, 2000.

[Velev 03] Velev, M. N., "Collection of High-Level Microprocessor Bugs from Formal Verification of Pipelined and Superscalar Designs", *Proc. IEEE Intl. Test Conf*., pp. 138-147, September, 2003.

[Wagner 08] Wagner, I., and V. Bertaco, "Reversi: Post-Silicon Validation System for Modern Microprocessors," *Proc. IEEE Intl. Conf. on Computer Design*, pp. 307-314, 2008.

[Williams 73] Williams, M.J.Y., and J.B. Angell, "Enhancing Testibility of Large Scale Integrated Circuits via Test Points and Additional Logic," *IEEE Trans. on Computers,* Vol. C-22, Issue 1, pp. 46-60, 1973.

[Woo 95] Woo, S. C., *et al.*, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proc. ACM/IEEE Intl. Symp. on Computer Architecture*, pp. 24-36, 1995

[Yerramilli 06] Yerramilli, S., "Addressing Post-Silicon Validation Challenges: Leverage Validation & Test Synergy (Invited Address)," *IEEE Intl. Test Conf.*, 2006.