Quick Error Detection Tests with Fast Runtimes for Effective Post-Silicon Validation and Debug

David Lin[1], Eswaran S[3], Sharad Kumar[3], Eric Rentschler[4], Subhasish Mitra[1,2]

[1]Department of EE and [2]Department of CS
Stanford University, Stanford, CA, USA

[3]Freescale Semiconductor
Noida, India

[4]Mentor Graphics
Longmont, CO, USA

Abstract

Long error detection latency, the time elapsed from the occurrence of an error caused by a bug to its manifestation as an observable failure, severely limits the effectiveness of existing post-silicon validation and debug techniques. Traditional post-silicon validation tests can incur very long error detection latencies of millions or even billions of clock cycles. An earlier technique called Quick Error Detection (*QED*) shortens error detection latencies to only few hundred (or thousand) clock cycles. However, *software-only QED* (i.e., QED implemented entirely in software) can result in significantly increased post-silicon validation test runtimes. We present a new technique called *Fast QED* that overcomes this drawback of software-only QED, while preserving the error detection latency and bug coverage benefits of software-only QED. Simulation results using an OpenSPARC T2-like multi-core SoC and bugs abstracted from multiple commercial multi-core SoCs demonstrate: 1. Fast QED achieves 4 orders of magnitude improvement in test runtime as compared to software-only QED, with only 0.4% increase in chip area; 2. Fast QED improves error detection latencies by up to 5 orders of magnitude compared to non-QED tests, and also achieves improved error detection latencies compared to software-only QED; and, 3. Fast QED improves bug coverage by up to 2-fold compared to non-QED tests (similar to software-only QED).

Keywords – *Debug, Post-Silicon Validation, Quick Error Detection*

1. Introduction

During post-silicon validation, a wide variety of tests such as random instruction tests, architecture-specific tests, or end-user applications [Adir 11, Bojan 07, Keshava 10] are run on manufactured integrated circuits (ICs) in actual system environments to detect bugs. Bugs can be broadly classified into: 1. *electrical bugs* caused by subtle interactions between a design and its electrical state [Patra 07], and 2. *logic bugs* caused by design errors. It has been reported that post-silicon validation costs are rising faster than design costs [Abramovici 06, Wisam 13, Yerramilli 06]. Upon detection, bugs must be localized, root-caused, and fixed. Bug localization / debug from observed failures dominates the cost of post-silicon validation [Abramovici 06, Amyeen 09, Friedler 14, Josephson 06]. Long *error detection latency*, the time elapsed from when an error is induced by a bug to when the error manifests as an observable failure, makes post-silicon debug highly challenging [Hong 10, Lin 12, 14, Reick 12]. For bugs with error detection latencies longer than a few thousand clock cycles, it is extremely difficult to trace too far back in system operation history for debugging. [Lin 12, 14] show that traditional post-silicon validation tests can incur very long error detection latencies, up to billions of clock cycles, especially for bugs inside uncore components of SoCs. *Uncore components* are neither processor cores nor co-processors. Examples include cache controllers, memory controllers, power management controllers, and on-chip interconnect networks.

Recently, [Hong 10, Lin 12, 14] presented the Quick Error Detection (*QED*) technique to create post-silicon validation tests with very short error detection latencies (few hundred or thousand clock cycles) and improved coverage (demonstrated using actual hardware results). Software-only QED uses software techniques, referred to as *QED transformations*, to transform existing post-silicon validation tests into new QED tests. A drawback of software-only QED is that the corresponding QED test runtimes can increase significantly (by up to 5 orders of magnitude depending on the QED transformations used). Since debug time is the main bottleneck in post-silicon validation, some test runtime increase (e.g., up to an order of magnitude) can be acceptable if error detection latencies and bug coverage are significantly improved. For example, 4-10X increase in test runtime is routinely observed in many post-silicon validation techniques that also incur very long error detection latencies [Adir 11, De Paula 11, Foutris 11, Wagner 08]. However, a large increase in test runtime, e.g., on the scale of 5 orders of magnitude, can be highly challenging because post-silicon validation is already severely constrained in terms of time and hardware resources [Adir 11, Bojan 07]. For example, an existing test that takes 1 minute to run may now end up taking a week. This problem gets worse for emulator / accelerator-based verification and debug since such platforms are slower than actual silicon.

In this paper, we overcome the runtime challenge of software-only QED, and make the following contributions:

1. We present the *Fast QED* technique for creating post-silicon validation tests with fast runtimes, while simultaneously preserving the short error detection latency and improved bug coverage benefits of software-only QED. Fast QED uses small hardware modifications to achieve fast runtimes.

2. We demonstrate that Fast QED enables up to 4 orders of magnitude improvement in test runtimes compared to software-only QED. Post-place-and-route results for the multi-core OpenSPARC T2 SoC [OpenSPARC] demonstrate that Fast QED incurs only 0.4% increase in chip-level area, and negligible increase in chip-level power and performance.

3. We present a list of realistic power management bug scenarios abstracted from actual bugs that occurred during the post-silicon validation of commercial multi-core SoCs. These bugs are very challenging to debug in post-silicon. This new list supplements the bug scenarios in [Lin 12, 14], which target difficult bugs inside processor cores, cache controllers, memory controllers, and on-chip interconnection networks.

4. We present simulation results for a complex OpenSPARC T2-like SoC with bug scenarios in [Lin 12, 14] and power management bugs (discussed above). We demonstrate that Fast QED preserves the improved error detection latency and coverage benefits of software-only QED (up to 4 orders of magnitude improvement in error detection latencies and up to 2-fold improvement in bug coverage compared to non-QED post-silicon validation tests), while significantly improving test runtimes (as stated above). In fact, Fast QED achieves further error detection latency improvements compared to software-only QED.

The rest of this paper is organized as follows. Section 2 presents the Fast QED technique. Section 3 presents a list of difficult power management bug scenarios. We present Fast QED results in Sec. 4. Related work is discussed in Sec. 5, followed by conclusions in Sec. 6.

2. Fast QED

The software-only QED technique [Hong 10, Lin 12, 14] transforms existing post-silicon validation tests (referred to as original tests) into QED tests using various QED transformations: software-only EDDI-V, CFCSS-V, CFTSS-V, and PLC. The software-only *Proactive Load and Check* (PLC) transformation incurs the highest runtime overhead [Lin 12, 14]. Hence, for Fast QED, we focus on Fast PLC, which improves test runtimes compared to software-only PLC.

The software-only PLC transformation inserts Proactive Load and Check operations (*PLC operations*) into the test (Fig. 1). The PLC operations execute on all threads running on all processor cores (Fig. 1a). Each PLC operation (Fig. 1b) proactively loads variables in the test and performs checks on the loaded values to quickly detect errors. The granularity at which the PLC operations are inserted is determined by the *Inst_min* and *Inst_max* parameters, defined as the minimum and maximum number of instructions from the original test that must execute before any instruction inserted by QED executes.

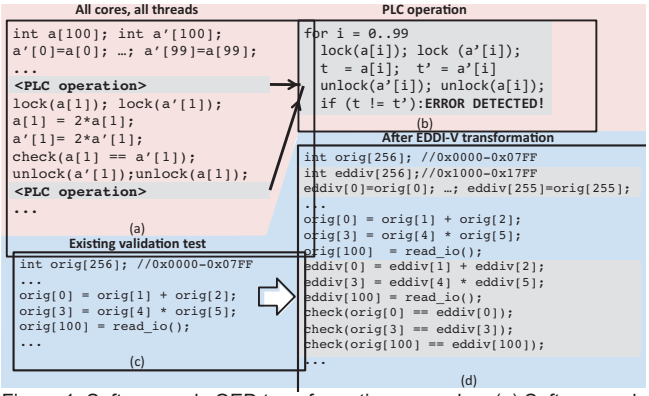

Figure 1. Software-only QED transformation examples. (a) Software-only PLC operations inserted in all threads on all processor cores. (b) Software-only PLC operation. (c) Existing validation test. (d) After EDDI-V Transformation.

2.1 Fast PLC Overview

The Fast PLC technique uses small hardware checkers, referred to as *PLC-Hardware (PLC-H) checkers*, to perform PLC operations in order to quickly detect bugs inside uncore components and to reduce test runtimes. Multiple PLC-H checkers can perform PLC operations concurrently (Sec. 2.6). We insert PLC-H checkers in all cache memories: L1, L2, and any higher-level caches (L3, L4 if they exist in the design). For OpenSPARC T2, we insert PLC-H checkers in the L1 and L2 caches (Sec. 2.6) since the design does not have any higher-level caches. We do not insert PLC-H checkers for other on-chip memories such as register files, TLB, and FIFO buffers, because bugs that affect these structures can be quickly detected using a combination of the software-only EDDI-V [Hong 10, Lin 14] and the PLC-H checkers for cache memories. We do not insert PLC-H checkers for external DRAMs because we target bugs inside SoCs. PLC-H checkers can be extended for external DRAMs (beyond the scope of this paper).

The uncore components in an SoC can be broadly classified into:

(a) Uncore components that are part of the cache subsystem: cache memories as well as the cache controllers.

(b) Uncore components (not in *a*) that communicate with processor cores using cache memories (via loads / stores). For OpenSPARC T2, this category includes memory controllers, network interface unit (network controller), PCI Express interface, and on-chip interconnection network connecting processor cores to caches.

(c) Uncore components that do not use cache memories to communicate with processor cores, but instead, use special instructions to communicate (e.g., I/O instructions). For OpenSPARC T2, this includes the programmable I/O and the interrupt-processing unit.

Fast PLC uses PLC-H checkers to target bugs inside components in the first two categories by performing PLC operations on the caches. Bugs inside components in category *c*, as well as inside processor cores, are quickly detected using the EDDI-V transformation of software-only QED. Example of EDDI-V for bugs in category *c* is shown in Fig. 1d, where we duplicate the I/O instruction and check the results. Figure 2c summarizes the uncore components covered by PLC-H checkers. The following subsections present the details of Fast PLC.

2.2 Validation Test Preparation for Fast PLC

Before a validation test can take advantage of the Fast PLC technique using PLC-H checkers, it is transformed using EDDI-V, a software-only QED transformation to ensure that bugs inside processor cores as well as uncore components in category *c* above are quickly detected. Figure 1 shows an example of the EDDI-V transformation (details in [Hong 10, Lin 14]). Figure 1c shows the existing validation test, and Fig. 1d shows the software-only EDDI-V transformed test with initialization of variables, duplication of instructions, and checking the results of original instructions vs. the results of the corresponding duplicated instructions.

2.3 PLC-H Checker Description

A PLC-H checker proactively loads from a variable in the original test (referred to as an original variable) and its corresponding variable created by the EDDI-V transformation (referred to as an EDDI-V variable), and performs a check by comparing the values of the two variables. Any mismatch indicates an error. Figure 2a shows a single PLC-H checker integrated into a memory built-in-self-test (*MBIST*) engine; if no MBIST engine exists, PLC-H does not reuse any MBIST components. Each PLC-H checker consists of the following:

1. The **PLC-H controller** controls other components in the PLC-H checker. It is responsible for determining when the PLC-H checker can perform PLC operations (details in Sec. 2.4).

2. When instructed by the PLC-H controller, the **address generator** generates the address of an original variable or the corresponding EDDI-V variable. The address ranges corresponding to the original variables and the EDDI-V variables are programmed into the address generator via debuggers (e.g., JTAG). If a separate PLC-H checker is used for each cache memory array (Sec. 2.6), the address generator only needs to generate addresses of variables that are cached in that particular cache memory array. For example, in OpenSPARC T2, the address range cached by the cache memory array 0 of bank 0 of the L2 cache is 0x0000-0x7FFF. If the address range of the original variables spans 0x6000-0x6FFF and 0x8000-0x8FFF, then the address generator of the PLC-H checker for that cache array only needs to generate addresses 0x6000-0x6FFF. The address range for each cache memory array is known from the design specification. To simplify address generation, during software-only EDDI-V transformation, our memory allocation library function partitions the memory address into two sets: one for the original variables and one for the EDDI-V variables. This is achieved by partitioning the memory addresses into "chunks" of 0x1000 addresses, alternating between addresses for original variables and addresses for EDDI-V variables (e.g., addresses 0x0000-0x0FFF, 0x2000-0x2FFF, etc. for original variables, and 0x1000-0x1FFF, 0x3000-0x3FFF, etc. for EDDI-V variables). For an original variable with address *A,* its corresponding EDDI-V variable is stored in address *A*+0x1000. For OpenSPARC T2, the smallest cache memory array (for both L1 and L2) is 8Kbytes; by dividing the memory address into 0x1000 chunks (4KBytes), this partition scheme ensures that both the original address range and its corresponding EDDI-V address range can be cached in the same array. The chunk size is a configurable parameter (we chose 0x1000 for OpenSPARC T2).

3. When instructed by the PLC-H controller, the **data register** holds the value loaded from the cache memory array.

4. The **comparator** compares the value held in the data register with the value loaded from the cache memory. Any mismatch indicates error. The error signal is mapped to on-chip debug circuit (e.g., JTAG).

5. The **multiplexers** select between PLC-H, MBIST, and normal modes. During normal and MBIST modes, the PLC-H checker does not perform any PLC operation. During MBIST mode, the PLC-H checker allows the MBIST to test the array. The MBIST mode is not needed if PLC-H checkers do not reuse MBIST components (Sec. 2.7).

2.4 PLC-H Controller

The PLC-H controller ensures that a PLC operation is performed only when the following criteria are satisfied:

Criterion 1. The cache memory array being checked does not have a load/store operation in progress. This is done by monitoring the cache array enable signal from processor cores or uncore components.

Criterion 2. The PLC-H checker should not introduce excessive intrusiveness (in order to prevent situations when a bug becomes undetected by Fast PLC due to excessive PLC operations). This is satisfied by counting the number of load / store operations (*OP_cnt* in Fig. 2b). A PLC operation is only performed when *OP_cnt* equals the configurable parameter *OP_cnt_min* (the minimum number of normal load/store operations that must execute before a PLC operation occurs).

Criterion 3. A PLC operation must not happen in between a store to an original variable and a store to the corresponding EDDI-V variable. This ensures that the PLC-H checker does not cause false fails. In software-based PLC, locks are used for this purpose. However, locks may not be available to PLC-H checkers, and improperly-implemented locks may lead to deadlocks. To satisfy criterion 3, we utilize the fact that EDDI-V always inserts EDDI-V store instructions in the same order as that of the original store

instructions (e.g., in Fig. 1d, *orig[0]* is written first, followed by *orig[3]*; correspondingly, *eddiv[0]* is written first, followed by *eddiv[3]*). As a result, we count the number of store operations (*ST_cnt* in Fig. 2b) to original and EDDI-V variables, and only perform a PLC operation when the two numbers match (i.e., *ST_cnt* == 0 in Fig. 2b). This approach is effective for strong memory ordering architecture (e.g., x86, x86_64, AMD64, and SPARC [McKenney 05]), which ensures that store operations are never reordered.

Architectures with weak memory ordering (e.g., ARM, IA64, and POWER [McKenney 05]) can reorder store operations. For such architectures, we insert a memory barrier instruction (in software) [McKenney 05] after every store instruction inserted by the software-only EDDI-V to ensure that those stores are not reordered. While the barriers can introduce some intrusiveness, the degree of intrusiveness can be systematically adjusted and controlled using the transformation parameter *Inst_min* for EDDI-V [Hong 10, 14], defined as the minimum number of instructions from the original test that must execute before any instructions (including memory barriers) inserted by the QED transformation can execute. A large *Inst_min* means that the memory barriers are inserted infrequently and longer sequences of memory instructions may execute in the original reordered state. In Sec. 4, we present results for SPARC with strong memory ordering, and an architecture with weak memory ordering. Empirical results demonstrate that the insertion of memory barriers (controlled by *Inst_min*) for weak memory ordering architectures does not change the error detection latency and bug coverage benefits of Fast QED (Sec. 4).

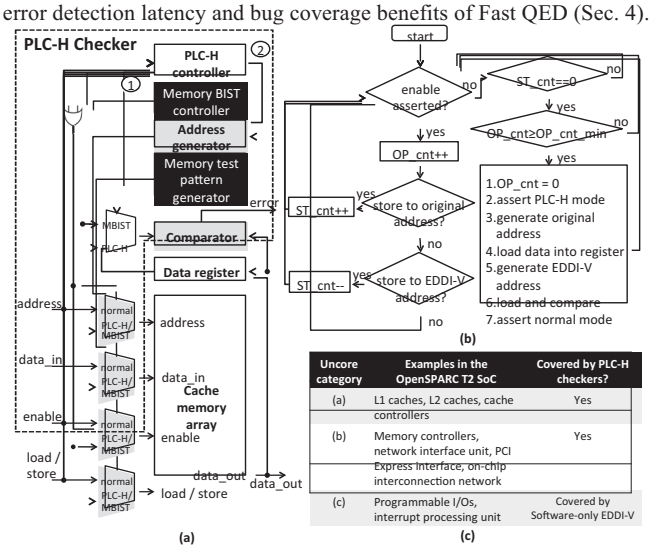

Figure 2. (a) PLC-H checker block diagram. Components shaded in gray can be shared between an MBIST engine and a PLC-H checker; components in black are part of an MBIST engine (not used by PLC-H checker). (b) Flowchart of PLC-H controller. (c) Uncore components covered by PLC-H checkers.

2.5 PLC-H Checker Example

When the PLC-H controller determines that the three criteria (Sec. 2.4) are satisfied, it asserts "PLC-H" on signal ① in Fig. 2a to perform a single PLC operation; during this time, the cache does not respond to normal load / store operations in the system (e.g., from processor cores). These operations are not lost, but are held at the input buffers of the caches (input buffers hold pending load / store operations when the cache is busy). For OpenSPARC T2, the input buffer for each cache can hold 8 pending load / store operations. If the buffers are full, further load / store operations are stalled.

Next, the PLC-H controller instructs the address generator (using signal ②) to generate the address of an original variable (e.g., 0x0000 for *orig[0]* in Fig. 1d). This address is looked up in the tag entry of the cache to determine its location in the cache memory array (e.g., the 1st entry in the L1 cache memory array). If the address does not exist in the tag entry (i.e., a cache miss), then it is not loaded and PLC operation is not performed on this variable. If the address does exist, its value is loaded into the data register. Next, the PLC-H controller instructs the address generator to generate the address of the corresponding EDDI-V variable (e.g., 0x1000 for *eddiv[0]* in Fig. 1d). This address is looked up in the tag entry to determine its location in the cache memory array (e.g., the 256th entry in the L1 cache memory array). If this address exists, it is loaded and its value is compared to the corresponding original variable's value stored in the data register. Any mismatch indicates an error. If the address does not exist, PLC operation is not performed on this variable. The PLC-H controller then asserts "normal" on signal ① in Fig. 2a to allow the cache to respond to normal load / store operations. This completes a single PLC operation.

2.6 PLC-H Checker Insertion Strategies

We use the OpenSPARC T2 SoC as a case study to discuss strategies for inserting PLC-H checkers. One option is to insert only a single PLC-H checker for the entire SoC. Since the PLC-H checker must perform PLC operations for all cache memories in the SoC, this can take an extremely long time. The OpenSPARC T2 has 4MBytes of L2 cache with a maximum read size of 64Bytes. Thus, it takes a minimum of 4MBytes / 64Bytes = 65,536 clock cycles to perform PLC operations for the entire L2 cache. Without careful insertion of PLC-H checkers, bugs may not be detected quickly. To overcome this limitation, multiple PLC-H checkers are inserted. For OpenSPARC T2, we insert 136 PLC-H checkers: one for each L1 data cache (8 total), and 16 for each bank of the eight L2 cache banks (128 total: each bank of L2 is constructed from 16 separate memory arrays). Each L1 cache contains 512 entries (cache lines), and one entry is read every cycle; thus, the entire L1 cache can be checked in 512 cycles. Each memory array in the L2 cache also contains 512 entries, but it takes 2 cycles to read an entry. Thus, it takes a minimum of 1,024 cycles to check all L2 memory arrays. Actual cycle count is higher due to *OP_cnt_min*, which controls how often PLC-H checkers check the memory array. The PLC-H checkers perform PLC operations concurrently. Figure. 3 shows the PLC-H checkers inserted in the OpenSPARC T2.

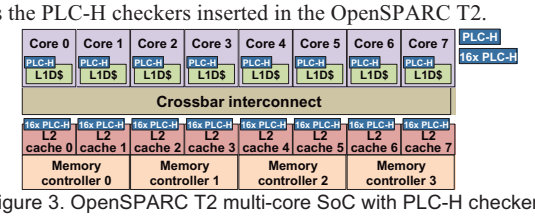

Figure 3. OpenSPARC T2 multi-core SoC with PLC-H checkers.

2.7 Memory BIST Reuse to Minimize PLC-H Checker Cost

In this subsection, we demonstrate how reusing MBIST engines in SoCs can minimize the area and power costs of PLC-H checkers. The block diagram of a PLC-H checker integrated into a "non-programmable" MBIST engine in OpenSPARC T2 is shown in Fig 2a. Programmable MBIST engines [Zarrineh 00] that exist in many SoCs can further reduce the area of PLC-H checkers: we can reuse the already-existing address generators in programmable MBIST engines.

We used the Synopsys Design Compiler with the Synopsys EDK 32nm library (standard cells and memories) for synthesis and Synopsys IC compiler for place-and-route to estimate the area of PLC-H checkers. 136 PLC-H checkers are used on the OpenSPARC T2 (Sec. 2.6), which contains 80 non-programmable MBIST engines; 56 of these engines are located in the L1 and L2, which we reuse for PLC-H checkers. The remaining 80 PLC-H checkers do not reuse MBIST engines. The 136 PLC-H checkers introduce 0.3% chip-level area overhead after synthesis, and 0.4% chip-level area overhead after place-and-route. PLC-H checkers can be used for emulation-based verification and debug, and are added only in the design mapped on the emulator (not necessarily in the final design). Therefore, they do not introduce any area, power, or performance costs in the final design.

3. Power Management Bug Scenarios

To advance post-silicon validation research, it is important to have realistic bug scenarios to quantify the effectiveness of existing and new post-silicon validation approaches. Previously, [Lin 12, 14] presented a list of bug scenarios in processor cores as well as uncore components of SoCs (cache controllers, memory controllers, and on-chip interconnection networks). [Lin 12, 14] also showed that those bug scenarios subsume bugs scenarios in [DeOrio 08, 09, Ho 95, Velev 03].

With massive integration of complex SoCs, power management features are becoming increasingly complex and very challenging to validate [Bojan 07, Keshava 10]. The slowdown of silicon CMOS (Dennard) scaling [Bohr 09] implies that increasingly complex power management is required to meet energy efficiency targets. Table 1 presents a list of bug scenarios obtained by analyzing reports of actual "difficult" bugs (primarily logic bugs) in the power management features of commercial multi-core SoCs for servers and mobile applications. The bugs are considered difficult because they took very long times to debug (e.g., several weeks for a single bug). We worked closely with the validation teams to extensively analyze the bug reports in order to abstract them into bug scenarios using high-level descriptions, while removing product-specific details. These bug scenarios supplement those in [Lin 12, 14].

Each bug scenario is decomposed into a bug activation criterion (Table 1A), and a bug effect (Table 1B). *Bug activation criterion* must be satisfied to activate a bug scenario. *Bug effect* is the incorrect behavior that results when a bug scenario is activated. In Table 1B, bug effects A-C correspond to bugs inside cache controllers, D-F correspond to bugs inside memory controllers, G-I correspond to bugs inside on-chip interconnection networks, and J-K correspond to bugs inside processor cores. A single bug scenario is formed as an ordered pair of an activation criterion and a bug effect.

Table 1A. Power management bug activation criterion.

ID	Description
1	When exiting from power-saving state.

Table 1B. Power management bug effects.

Type	ID	Description
Uncore components	A	The value of the next load operation from data cache is corrupted to all 0.
	B	Next load operation from data cache delayed (1 clock cycle) by cache controller.
	C	Data cache drops the next load operation.
	D	The value of the next load operation from main memory is corrupted to all 0.
	E	Next load operation from main memory delayed (1 clock cycle) by memory controller.
	F	Next load request to main memory is dropped.
	G	Next load operation is delayed for 1 clock cycle by the interconnection network.
	H	Next load operation is corrupted to all 0 by the interconnection network.
	I	Next load operation is dropped by the interconnection network.
Processor cores	J	Processor jumps to a random address.
	K	Next instruction is corrupted to NOP
	L	The value of the next register read is corrupted to all 0.

4. Results

To demonstrate the effectiveness of Fast QED in achieving fast runtimes, short error detection latencies, and high coverage for a wide variety of bugs, we present simulation results for the bug scenarios in [Lin 12, 14] and the power management bugs in Sec. 3. We used GEMS [Martin 05] to simulate an OpenSPARC T2-like multi-core SoC (Fig. 3), a 500-million-transistor design, with 8 processor cores (64 hardware threads total). Each processor core has private L1 caches. A crossbar connects the processor cores to 8 banks of shared L2 cache and 4 memory controllers. The OpenSPARC T2 implements Total Store Ordering [OpenSPARC] (a form of strong memory ordering).

Bug scenarios from [Lin 12, 14] were simulated using the methodology in [Lin 12, 14]. For bugs in Sec. 3, we simulated the bug effects when the system exits from the power-saving state. To simulate exit from the power-saving state, we randomly selected a sequence of 5 instructions from the *original test* to serve as a trigger for entering power-saving state (a system enters power-saving state when it executes a specific sequence of instructions, i.e., to save the system states). On the next clock cycle, the system is considered to have exited the power-saving state and the bug effect is simulated (we consulted with validation engineers to ensure the validity of this approach). The sequence of instructions for the trigger to enter power-saving state is the same for the original test, the QED test, and the Fast QED test; this allows us to quantify any intrusiveness introduced by Fast QED.

We used a combination of SPLASH-2 [Woo 95] and SPECINT 2000 [Henning 00] benchmarks as the original tests. For SPLASH-2, we used 8-threaded versions of the tests (one thread per processor core). For SPECINT 2000 (which are single threaded), each of the 8 cores runs a single instance of the test. 64-threaded versions were not used because the simulator is not cycle-accurate when simulating processor core supporting more than 1 hardware thread [Martin 05].

The runtime and area cost (post place-and-route) is summarized in Table 2. Fast PLC introduces 5-6X increase in test runtime on OpenSPARC T2 compared to the original tests because the runtime of Fast PLC is bounded by the runtime of software-only EDDI-V used by Fast PLC. For most instructions in the original test, EDDI-V inserts one duplicated instruction, one compare instruction, and a conditional branch instruction (to indicate if an error is detected). On non-superscalar processors (e.g., in OpenSPARC T2), this is approximately 4X increase in runtime (actual runtime may be slightly higher due to cache misses). On superscalar processors, the runtime overhead of EDDI-V tests is only 1.1X to 2X [Oh 02]; hence, the runtime of Fast PLC can be further reduced for superscalar processors.

Given the growing complexity and importance of uncore components in SoCs, if only uncore bugs are targeted, we can reduce the runtime of Fast QED by duplicating only store instructions from the original test. This reduces the runtime overhead of Fast QED to less than 15% ("Fast QED (uncore only)" in Table 2), and does not change the error detection latencies and coverage of Fast QED for uncore bugs (i.e., activation criterion 1 in Table 1A, bug effect A-I in Table 1B, activation criterion 1-5 and bug effect A-G in [Lin 12, 14]) (Fig. 4).

To put the runtime of Fast QED into perspective, many post-silicon validation techniques also incur long test runtimes (and may incur millions or billions cycles of error detection latencies, unlike Fast QED). The multi-pass checking technique [Adir 11] requires running a test multiple times resulting in several fold increase in test runtime and can incur millions or billions of cycles of error detection latencies. The technique in [De Paula 11] requires running original tests multiple times (e.g., 10). The technique in [Foutris 11] increases test runtime by 6X, and [Wagner 08] increases test runtime by approximately 4X.

Figure 5 illustrates the tradeoffs between number of the PLC-H checkers inserted and the distribution of error detection latencies. The horizontal axis corresponds to the number of PLC-H checkers inserted and the area cost (on OpenSPARC T2). The vertical axis represents the distribution of error detection latencies showing the minimum, median (represented by a square dot) and the maximum error detection latencies for bug scenarios in Sec. 3 and in [Lin 12, 14]. The number of PLC-H checkers start from 16 (1 for each of the 8 L1 caches and 1 for each of bank of the 8 L2 cache banks in OpenSPARC T2). We gradually increase the number of PLC-H checkers in each bank of L2 from 1 to 16 (1 PLC-H per array, 2 PLC-H per array, and so on), while the number of PLC-H checkers in L1 stays constant, until we reach 136 PLC-H checkers (Sec. 2.6). Area cost ranges from 0.05% (16 PLC-H checkers) to 0.4% (136 PLC-H checkers). With 16 PLC-H checkers (area cost of 0.05%), the median and maximum error detection latencies are 7k and 20k clock cycles, respectively for the *bzip2* and the *lu* tests. With 136 PLC-H checkers (area cost of 0.4%), the median and maximum error detection latencies are 412 and 3k clock cycles, respectively, for the *bzip2* test and 385 and 5k clock cycles, respectively, for the *lu* test. Similar trends are observed for other tests.

Observation 1: Fast QED enables up to 4 orders of magnitude improvement in runtimes compared to worst-case software-only QED tests, with only 0.4% increase in chip area. This, with Observations 2 and 3 presented later, demonstrate that Fast QED achieves major improvement in test runtimes while **simultaneously** preserving the quick error detection and coverage benefits of software-only QED.

The error detection latencies and coverage are summarized in Fig. 4, where the horizontal axes represent error detection latencies in clock cycles (logarithmic scale); we also show the median and maximum error detection latencies (*Med.*, *Max. EDL*). The vertical axes represent the cumulative percentage of bug scenarios detected with respect to all bug scenarios in Sec. 3 and [Lin 12, 14] combined for "Original test", "Fast QED test" and "Software-only QED test" and with respect to uncore bugs only for "Fast QED (uncore only)". The "Original test" represents the original test with "end result checks" that compare the

results of the test against pre-computed, known correct results. This is possible because the inputs to the tests are known *a priori* (default inputs). The "Fast QED test" is created from the original test using the Fast PLC (which includes software-only EDDI-V, as explained in Sec. 2.2). For Fast PLC, the simulated system was modified to include 136 PLC-H checkers (Sec. 2.6). Fast QED do not rely on any information about the bugs and are not specifically tailored to the bugs simulated. The "Software-only QED test" corresponds to QED tests created using software-only PLC [Lin 12, 14] (includes software-only EDDI-V). "Fast QED (uncore only)" was explained above. *Inst_min*, *Inst_max*, and *OP_cnt_min* are set to 5.

Observation 2: Fast QED retains the error detection latency benefits of software-only QED. Fast QED also achieves lower error detection latencies compared to software-only QED (Fig. 4) for all bug scenarios. The error detection latencies of the original tests can be up to hundreds of millions of clock cycles. Fast QED detects all bugs with error detection latencies less than 7 thousand clock cycles, and detects most bugs with error detection latencies of only a few hundred cycles.

Observation 3: Fast QED retains the coverage benefits of software-only QED. Fast QED detected all bug scenarios detected by the software-only QED (and original tests). Fast QED (and software-only QED as well) detected up to 2X more bugs vs. original tests. This empirically demonstrates Fast QED does not introduce excessive intrusiveness. We confirmed that the coverage benefits of Fast QED vs. original test are due to Fast QED's check operations and not because of its execution time (similar to the observations in [Lin 12, 14]).

Observation 4: Fast QED for uncore only retains the error detection latency and coverage benefits of software-only QED for uncore bugs while incurring less than 15% runtime overhead.

To demonstrate effectiveness of Fast QED for weak memory ordering architectures, we modified our simulator to simulate an SoC with weak memory ordering. For the weak memory ordering architecture only, we introduce bug activation criterion 2 to Table 1A:

2. When memory reordering occurs.

This activation criterion enables us to evaluate the intrusiveness of Fast QED's memory barriers on weak memory ordering architectures (which allow reordering of memory accesses). This activation criterion is not used for strong memory ordering architectures because memory reordering does not occur in strong memory ordering architectures. We simulated the bug scenarios in [Lin 12, 14] and Sec. 3 (with activation criterion 2) on the weak memory ordering architecture. Error detection latencies, coverage, and runtimes for *bzip2* and *lu* are reported in Fig. 6 (similar trends were observed for other tests).

Observation 5: Fast QED is effective for both strong and weak memory ordering architectures. The error detection latencies, coverage and runtimes of Fast QED for strong memory ordering and weak memory ordering are the same. The insertion of memory barriers for Fast QED in the weak memory ordering architecture did result in 5-15% fewer bug activations compared to the original tests (memory barrier instruction prevents the reordering of memory operations before the barrier with memory operations after the barrier), but did not change the error detection latency and coverage benefits of Fast QED.

5. Related Work

Similar to software-only QED [Hong 10, Lin 12, 14], this paper focuses on techniques for creating effective post-silicon validation tests with short error detection latencies and improved coverage. A drawback of software-only QED is that the QED test runtimes can increase significantly (by up to 5 orders of magnitude depending on the QED transformations used). Fast QED overcomes this challenge with only 0.4% chip-level area cost and negligible increases in system-level power and performance. Fast QED retains the short error detection latency and improved coverage benefits of software-only QED.

Fast QED is different from traditional validation tests [Adir 11, Aharon 95, Foutris 11, Raina 98, Wagner 08] in two respects: 1. Traditional tests can incur extremely long error detection latencies (up to billions of clock cycles) vs. very short error detection latencies using software-only QED and Fast QED; 2. Fast QED incorporates a fine-tuned mix of hardware and software techniques to reduce test runtime, and improve error detection latency and bug coverage.

PLC-H checkers are different from memory scrubbing techniques [Abraham 83, Shirvani 00] for fault-tolerant computing or MBIST techniques used for memory testing. These techniques generally focus on errors inside memory arrays (sometimes assisted by error-correcting codes), and do not target bugs outside memory arrays (vs. PLC-H targets bugs inside cache controllers, memory controllers, and interconnection networks). Furthermore, scrubbing and MBIST occur infrequently, and can result in very long error detection latencies. There are several techniques for improving observability during post-silicon validation: e.g., trace buffer insertion [Abramovici 06, Basu 11, Ko 08, Liu 09] and memory access logging [DeOrio 08, 09]. These techniques can benefit from the short error detection latencies and high coverage of software-only QED and Fast QED.

Similar to software-only QED, Fast QED is different from design-specific assertions [Abramovici 06, Boule 07, Ernst 07, Hangal 05, Vasudevan 10], which are difficult to keep up-to-date, and to validate their correctness [Bentley 01, Vasudevan 10]. While automatic assertion generation techniques exist [Ernst 07, Hangal 05, Vasudevan 10], it is difficult to select only the relevant set of assertions to reduce area costs while simultaneously ensuring effective debug. Assertions may depend on signals located in different regions of an IC and must be decomposed into components that only use nearby signals. In contrast, Fast QED performs extensive checks, and does not suffer from these drawbacks. Reconfigurable debug logic for hardware assertions [Abramovici 06] also can be reused for PLC-H checkers.

Upon quick bug detection using Fast QED or software-only QED, post-silicon bug localization techniques can be invoked [Abramovici 06, De Paula 11, Park 09, 10, Zhu 11].

Table 2. Normalized test runtime & area cost.

Techniques	Area cost	Normalized runtimes								
		bzip2	crafty	fft	lu	mcf	parser	vpr	quake	art
Original	0%	1X	1X	1X	1X	1X	1X	1X	1X	1X
Software-only QED (PLC) [Lin 12]	0%	52,224X	59,392X	28,672X	32,768X	58,368X	55,234X	56,134X	57,382X	51,023X
Fast QED	0.4%	5.22X	5.92X	5.52X	5.84X	6.15X	6.12X	5.78X	6.22X	6.36X
Fast QED (uncore only)	0.4%	1.13X	1.10X	1.11X	1.04X	1.14X	1.12X	1.12X	1.10X	1.14X

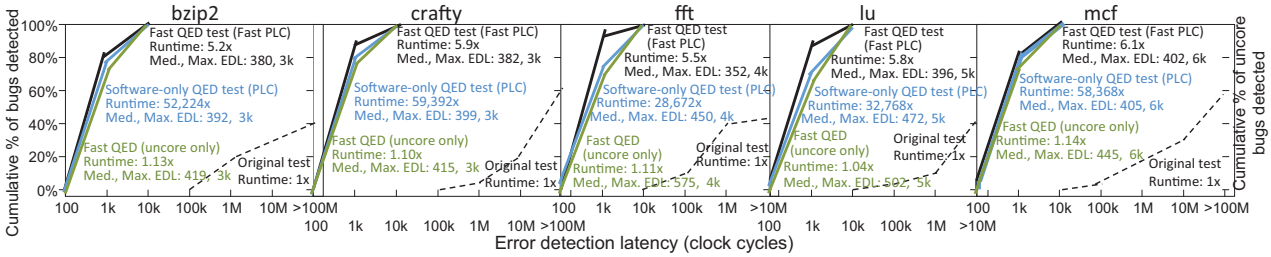

Figure 4. Error detection latency, coverage, and runtime results. The cumulative percentage of bugs detected for "Fast QED test (Fast PLC)", "Software-only QED test (PLC)", and "Original test" are reported for all power management bug scenarios (Sec. 3) and bug scenarios in [Lin 12, 14]. The cumulative percentage of bugs detected for "Fast QED (uncore only)" are with respect to only the uncore bug scenarios.

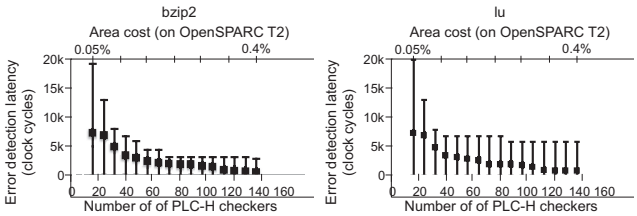

Figure 5. Error detection latencies vs. area cost and number of PLC-H checkers for OpenSPARC T2 [OpenSPARC].

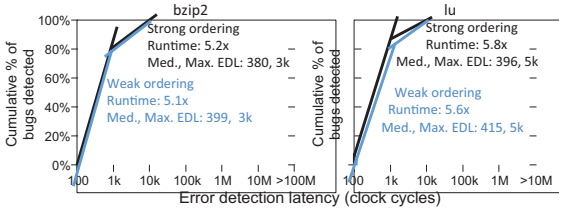

Figure 6. Error detection latencies, coverage, and runtimes for bug scenarios in Sec. 3 (with activation criterion 2 used only for weak memory architecture) and in [Lin 12, 14] for both strong and weak memory ordering architectures.

6. Conclusion

The Fast QED technique uses a fine-tuned mix of hardware and software techniques to create post-silicon validation tests with short error detection latencies and high coverage benefits of software-only QED tests, without incurring significant runtime overheads of software-only QED. Fast QED enables up to 5 orders of magnitude improvement in error detection latencies and up to 2-fold improvement in coverage compared to the original (non-QED) tests. Fast QED achieves up to 4 orders of magnitude improvement in test runtimes compared to software-only QED, and incurs 0.4% increase in chip-level area, and negligible increases in system-level power and performance. Fast QED enables flexible tradeoffs between area cost, error detection latencies, and test runtime. The use of Fast QED during emulation-based verification and debug enables quick error detection, high coverage, and low test runtime without adding area costs in the manufactured IC.

Fast QED enables several new research opportunities, including: 1. Automated bug localization by utilizing short error detection latencies of Fast QED, and by systematically analyzing which PLC-H checkers detected errors and which did not; 2. Bug detection in system software (i.e., firmware) of SoCs; and 3. Identification of failing ICs and root-causing No-Trouble-Found (NTF) ICs [Conroy 05] by using Fast QED and PLC-H checkers at the full-system level.

7. Acknowledgements

This research was supported in part by SRC, NSF, and SGF.

8. References

[Abraham 83] Abraham, J. A., E. S. Davidson, and J. H. Patel, "Memory System Design for Tolerating Single Event Upsets," *IEEE Trans. Nuclear Science*, Vol. 30, No. 6, pp. 4339-4344, December, 1983.

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

[Adir 11] Adir, A., *et al.,* "Threadmill: A Post-Silicon Exerciser for Multi-Threaded Processors," *IEEE/ACM Design Automation Conf.*, 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.

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

[Bentley 01] Bentley, B., and R. Gray, "Validating the Intel Pentium 4 Processor." *Intel Technology Journal*, Vol. 5 No. 1, pp. 1-8, 2001.

[Bohr 09] "The New Era of Scaling in an SoC World," *Proc. IEEE Solid-Sate Circuits Conf.*, pp. 1-10, 2009.

[Bojan 07] Bojan, T., *et al.,* "Functional Coverage Measurements and Results in Post-Silicon Validation of Core™2 Duo Family," *Proc. IEEE Intl. High Level Design Validation and Test Workshop*, pp. 145-150, 2007.

[Boule 07] Boule, M., J.-S. Chenard, and Z. Zilic, "Assertion Checkers in Verification, Silicon Debug and In-Field Diagnosis," *Proc. IEEE Intl. Symp. Quality Electronic Design*, pp. 613-620, 2007.

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

[De Paula 11] De Paula, F. M., *et al.,* "TAB-BackSpace: Unlimited-Length Trace Buffers with Zero Additional On-Chip Overhead," *Proc. IEEE/ACM Design Automation Conf.,* pp. 411-416, 2011.

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

[Ernst 07] Ernst, M. D., *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.

[Foutris 11] Foutris, N., *et al.,* "Accelerating Microprocessor Silicon Validation by Exposing ISA Diversity," *Proc. IEEE/ACM Intl. Symp. Microarchitecture*, pp. 386-397, 2011.

[Friedler 14] Friedler, O., *et al*., "Effective Post-Silicon Failure Localization Using Dynamic Program Slicing," *Proc. IEEE/ACM Design Automation Test in Europe*, pp. 1-6, 2014.

[Hangal 05] Hangal S., *et al*., "IODINE: A Tool to Automatically Infer Dynamic Invariants," *IEEE/ACM Design Automation Conf*., 2005.

[Henning 00], Henning, J. L., "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *IEEE Computer,* pp. 28-35, 2000.

[Ho 95] Ho, R.C., *et al.,* "Architecture Validation for Processors," *Proc. ACM/IEEE Intl. Sump. 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.

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

[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.

[Lin 14] Lin, D., *et al*., "Effective Post-Silicon of System-on-Chips Using Quick Error Detection," *IEEE Trans. CAD*, Vol. 33, No. 10, pp. 1573-1590, Oct. 2014.

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

[Martin 05] Martin, M., *et al.*, "Multifacet's General Execution-Drive Multiprocessor Simulator (GEMS) Toolset," *ACM SIGARCH Computer Architecture News*, Vol. 33, No. 4, pp. 92-99, November, 2005.

[McKenney 05] McKenney, P. E., "Memory Ordering in Modern Microprocessors, PART I," *Linux Journal*, Vol. 2005, No. 136, Aug. 2005.

[Oh 02] Oh, N., P. P. Shirvani, and E. J. McCluskey, "Error Detection by Duplicated Instructions in Super-Scalar Processors," *IEEE Trans. Reliability*, Vol. 51, No. 1, pp. 63-75, 2002.

[OpenSPARC] "OpenSPARC T2 SoC," http://www.opensparc.net.

[Park 09] Park, S.-B., T. Hong, and S. Mitra, "Post-Silicon Bug Localization in Processors Using Instruction Footprint Recording and Analysis (IFRA)," *IEEE. Trans. CAD*, Vol. 28, No. 10, pp. 1545-1558, Oct. 2009.

[Park 10] Park, S.-B., *et al.,* "BLoG: Post-Silicon Bug Localization in Processors Using Bug Localization Graph", *Proc. IEEE/ACM Design Automation Conf.,* pp. 368-373, 2010.

[Raina 98] Raina, R., and R. Molyneaux, "Random Self-Test Method Applications on PowerPC™ 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.

[Shirvani 00] Shirvani, P. P., N. R. Saxena, and E. J. McCluskey, "Software-Implemented EDAC Protection Against SEUs," *IEEE Trans. on Reliability*, Vol. 49, No. 3, pp 273-284. September, 2000.

[Vasudevan 10] Vasudevan, S., *et al.,* "GoldMine: Automatic Assertion Generation Using Data Mining and Static Analysis," *Proc. IEEE/ACM Design, Automation, Test in Euro. Conf.,* pp. 626 – 629, 2010.

[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, 2003.

[Wisam 13] Wisam, K., *et al.*, "Improving Post-Silicon Validation Efficiency by Using Pre-Generated Data," *Proc. Intl. Haifa Verification Conf*., pp. 166-181, 2013.

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

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

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

[Zarrineh 00] Zarrineh, K., *et al.,* "Self Test Architectures for Testing Complex Memory Structures," *Proc. IEEE. Intl. Test Conf.,* pp. 547-556. 2000.

[Zhu 11] Zhu, C.S., G. Weissenbacher, and S. Malik, "Post-Silicon Fault Localisation Using Maximum Satisfiability and Backbones," *Proc. IEEE/ACM Formal Methods Comp.-Aided Des.*, pp. 63-66*,* 2011.