Fault Detection Methodology for Caches in Reliable Modern VLSI Microprocessors based on Instruction Set Architectures

Georgios A. Theodorou*

National and Kapodistrian University of Athens
Department of Informatics & Telecommunications
gthe@di.uoa.gr

Abstract. This PhD thesis introduces a low cost Software-Based Self-Test (SBST) fault detection methodology for small embedded cache memories. The methodology leverages existing powerful mechanisms of modern ISAs by utilizing instructions that we call in this PhD thesis Direct Cache Access (DCA) instructions. Moreover, our methodology exploits the native performance monitoring hardware and the trap handling mechanisms which are available in modern microprocessors. By effectively combining these features of modern microprocessors the proposed methodology applies March test operations with lower cost (code size, execution time, system performance overhead) when compared with other proposed solutions in the literature for fault detection in caches through SBST. Finally, a multithreaded optimization of the proposed methodology is also presented. The proposed methodology was applied to three processor benchmarks: a) OpenRISC 1200 b) LEON3 and c) OpenSPARC T1. Experimental results both for the test code size and test execution time of several March tests demonstrate the significant improvements in terms of test time (86% for instruction L1 cache, 87% for the data L1 cache, about 40% for D-TLB and about 82% for I-TLB) and test code size (83% for instruction L1 cache, 86% for the data L1 cache, 3% for D-TLB and 35% for I-TLB) when the methodology is applied to the same benchmarks (LEON3 for L1 caches and OpenSPARC T1 for TLBs) and such DCA instructions are exploited compared to SBST solutions that don't utilize these types of instructions.

Keywords: Microprocessor testing, Reliability, Software-based Self-test, On-line testing, Memory testing, March tests

1 Introduction

During the past 30 years the semiconductor industry has been characterized by a steady path of constantly shrinking transistor geometries and increasing chip size. However, this technology achievement leads to new reliability challenges for modern systems that have not been considered in the past. Such reliability threats are either latent hardware defects that have not been detected by manufacturing tests or

* *Dissertation Advisor: A. Paschalis, Professor*

hardware defects that may occur during system operation by the increased soft error rate or by aging degradation effects. On-line testing schemes aim to detect such faults both in logic and memory modules of modern chips during their lifetime. Nowadays, in modern processors the relative chip area occupation of cache devices is up to 90%. Thus, high quality cache memory on-line testing in modern processors is essential.

Small memories, such as L1 caches and Translation Lookaside Buffers (TLBs) are not usually equipped with Memory Built-In Self-Test (MBIST) hardware or lack a programmable MBIST scheme because of its impact on chip area and performance. Software-Based Self-Test (SBST) is a flexible and low-cost solution for on-line March test application and error detection in such small memories. Although, L1 caches and TLBs are small components, their reliable operation is crucial for the system performance due to the large penalties caused when L1 cache or TLB misses occur.

This PhD thesis introduces a low cost fault detection methodology for small embedded cache memories that is based on modern Instruction Set Architectures and is applied with SBST routines. The proposed methodology applies March tests through software to detect both storage faults [1] when applied to caches that comprise SRAM memories only, e.g. L1 caches, and comparison faults [2] when applied to caches that apart from SRAM memories comprise CAM memories too, e.g. TLBs. The proposed methodology can be applied to all three cache associativity organizations: direct mapped, set-associative and full-associative and it does not depend on the cache write policy. The methodology leverages existing powerful mechanisms of modern ISAs by utilizing instructions that we call in this PhD thesis Direct Cache Access (DCA) instructions. Moreover, our methodology exploits the native performance monitoring hardware and the trap handling mechanisms which are available in modern microprocessors. By effectively combining these features of modern microprocessors the proposed methodology applies March write and read operations with lower cost (code size, execution time, system performance overhead) when compared with other proposed solutions in the literature for fault detection in caches through SBST. Moreover, the proposed methodology applies March compare operations when needed (for CAM arrays) and verifies the test result with a compact response to comply with periodic on-line testing needs. Finally, a multithreaded optimization of the proposed methodology that targets multithreaded, multicore architectures is also presented in this thesis. The proposed multithreaded optimization exploits the thread level parallelism of multithreaded, multicore architectures and the low level multiple sub-bank organization of modern cache designs to speedup March tests while preserving the March test quality. Hence, in case of multithreaded, multicore architectures that can adopt the proposed multithreaded optimization, test time is further lowered and such SBST routines can be effectively executed periodically during the system's lifetime with an acceptable performance overhead.

The proposed methodology was applied to three processor benchmarks: a) OpenRISC 1200 b) LEON3 and c) OpenSPARC T1. In detail, the methodology was applied to the L1 caches of all three processor benchmarks and to the TLBs of OpenSPARC T1 processor. The multithreaded optimization was demonstrated on the multithreaded, multicore OpenSPARC T1. Experimental results both for the test code size and test execution time of several March tests demonstrate the effectiveness of the proposed methodology, its high adaptability and the significant improvements in

terms of test time and test code size when compared with other proposed solutions in the literature for fault detection in caches through SBST that do not exploit DCA instructions [9] [13].

Finally, a test evaluation framework was implemented in this thesis for several on-line periodic test scenarios in order to evaluate the system performance overhead of the proposed methodology. Simulation results indicate that the proposed March test implementation through SBST slightly influences the system's performance, even in intensive test scenarios with high test frequency requirements.

1.1 Related Work

Several SBST approaches have been proposed [3] – [13] to apply March tests to L1 caches but none for TLBs. Besides, none of the proposed SBST approaches in the literature exploits special instructions that can be considered as DCA instructions. In [3], the first systematic approach for transforming March B test algorithm for in-system testing of Intel 860 processor cache is proposed. In [4], a March test transformation methodology is proposed for in-system testing of both data and tag arrays of data and instruction cache memories with various organizations by taking advantage of features such as enable/disable or freeze. In [5], the authors propose a methodology to transform March tests for in-system testing targeting only the data cache memory tag without providing implementation details. The transformation methodology targets direct mapped caches and is applied on March B and March X tests. In [6], the methodology of [4] is enhanced to exploit microprocessor's performance monitoring hardware and on-line hardware detectors to improve test observability. In [7], a SBST approach is proposed to develop March-based self-test programs for the data array of both instruction and data caches. Experimental results for traditional memory faults are provided for a MIPS R3000 processor model for different March tests. In [8], March tests are translated in order to effectively test the data and tag part of set-associative caches with Least-Recently Used (LRU) replacement and both write-back and write-through policies. In [9], a software-based methodology is presented for testing memory arrays and logic modules of a direct-mapped data cache. Experimental results are provided for an ARM-compatible processor both for the cache arrays and the logic modules. In [10], a hybrid SBST approach is proposed to test data and instruction cache controllers by combining instruction-based pattern generation and an I-IP module insertion for observability. Experimental results for the cache controllers of OpenRISC 1200 processor are provided. In [11], a test program generation approach is proposed to generate suitable programs for testing the replacement logic in set-associative caches. Experimental results of a cache that implements the LRU policy are provided. In [12], the capabilities and limitations of CPU-based at-speed memory testing are presented based on test routine examples for an ATMEL RISC microcontroller. Such SBST routines can be also adopted for testing cache arrays. Recently in [13], a methodology to exploit the ISA to translate generic March tests into SBST programs for set-associative instruction cache memories is presented and was applied on the instruction cache of the LEON3 microprocessor.

2 Cache arrays testability challenges

There are three basic cache organizations: direct mapped, set-associative and fully-associative. In processor design, data and instruction L1 caches are usually organized as direct mapped and set-associative caches whereas data and instruction TLBs are always organized as fully associative caches.

A typical L1 cache organization comprises of at least two SRAM memory arrays (or two SRAM arrays per way in set-associative organizations) - the data array and the tag array - whereas a fully-associative TLB organization comprises of one SRAM array - the data array - and one CAM array - the tag array. CAM is a special type of memory that compares all the stored data in parallel with incoming data and is utilized in the tag part of fully associative caches to speed up the tag comparison. Further down, those arrays will be denoted as DL1-Data, DL1-Tag, IL1-Data and IL1-Tag for the data and instruction L1 cache whereas for TLBs those arrays will be denoted as DTLB-Data, DTLB-Tag, ITLB-Data and ITLB-Tag for the data and instruction TLB, respectively.

All these cache arrays (either for L1 caches or TLBs) are implicitly accessed because they are not directly visible to the assembly programmer through the ISA. Hence, applying test patterns and observing the test responses through a software test routine is challenging. The challenges of accessing and thus testing those implicitly accessed cache arrays are summarized in Table 1 and thoroughly described in the thesis manuscript.

Table 1: Cache arrays testability challenges

Testability Challenges	***Cache arrays***							
	DL1 Data	***DL1 Tag***	***IL1 Data***	***IL1 Tag***	***DTLB Data***	***DTLB Tag***	***ITLB Data***	***ITLB Tag***
Direct access from generic ISA	×	×	×	×	×	×	×	×
Indirect March write	✓	✓	×	✓	×	×	×	×
Indirect March read	✓	×	×	×	×	×	×	×
Data Backgrounds	✓	×	×	×	×	×	×	×
Ascending Address Order	✓	✓	✓	✓	×	×	×	×
Descending Address Order	✓	✓	×	×	×	×	×	×
March Compare operation	-	-	-	-	-	×	-	×

The proposed methodology overcomes all these testability challenges for all cache arrays of both L1 caches and TLBs and optimizes the SBST routines in terms of test time and test code size by exploiting DCA instructions.

3 DCA instructions in modern ISAs

So far, previous SBST approaches cannot successfully overcome all the above mentioned challenges by using generic instructions to access cache arrays both for write and read operations. Fortunately, modern ISAs include dedicated instructions for debug-diagnostic and performance purposes that provide direct controllability and

observability of cache arrays. These instructions are extremely suitable for cache and TLB SBST; we use the term Direct Cache Access (DCA) instructions to refer to them.

In order to gain direct access to the cache arrays for all three cache organizations (direct mapped, set-associative and fully-associative) and implement a software-based March test, an ideal DCA instruction needs to contain the following fields:

Fields for selecting cache/TLB content:

- Way Selection (WS) field.
- Set Selection (SS) field.
- Line Word Selection (LWS) field.

Field for selecting internal cache array:

- Data/Tag Array Selection (AS) field.

Field for selecting March operation:

- Write/Read/Compare operation (WRC) field

Field for addressing register/memory for fetching DBs:

- From/To data Address (A) field

An ideal DCA instruction that contains all these fields gains direct access to any cache array, hence it can apply March operations through ISA to these arrays in a very effective way and overcome all the above described testability challenges. DCAs that access direct mapped caches should contain only fields *SS* and *LWS* while DCAs for accessing set associative L1 caches should contain all three *WS*, *SS* and *LWS* fields. DCAs that access fully associative caches and TLBs should contain only *WS* and *LWS* fields (fully associative caches contain 1 set with many ways) respectively. When the cache organization imposes a uniform cache line (e.g. TLBs) *LWS* field is not required. Finally, if the cache organization does not comprise a CAM memory (e.g. L1 caches), the March operation selection field can be renamed to *WR* field (only write/read operations, no compare).

In Figure 1 and Figure 2, ideal DCA instructions and the way that every field is utilized to access a 2-way set associative L1 cache and a TLB, are presented, respectively.

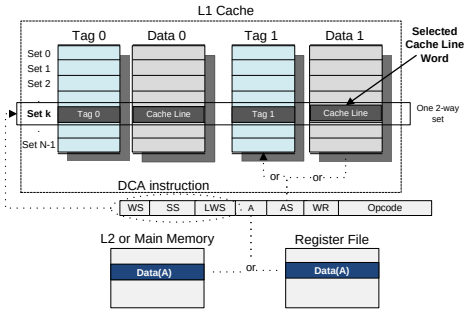

Figure 1: **Ideal DCA instruction for 2-way set associative L1 cache**

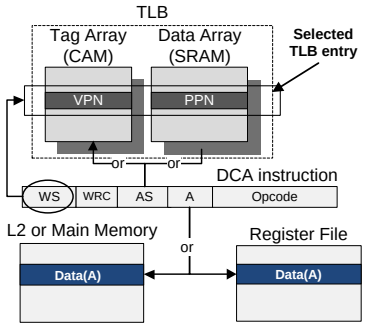

Figure 2: **Ideal DCA instruction for TLB**

In detail, in L1 set-associative caches the data or tag array (selected by the *AS* field) is accessed both for write and read operation (selected by the *WR* field). The selection of a word inside a cache line is controlled in three steps. First, the *SS* field selects the set, then, the *WS* field selects the cache way and finally the *LWS* field selects the word inside the cache line. Furthermore, all DBs can be composed by initializing either a general purpose register or a memory location that can be accessed by the *A* field. In TLBs, the data or tag array of the TLBs can be accessed by controlling the *AS* field. The March operation can be selected with the *WRC* field to write, read or compare a selected TLB entry. Compare operation is valid only for tag array. TLBs are fully associative arrays; hence the *WS* field is needed in the ideal DCA instruction to gain access to every TLB entry. Note that, such an instruction has no limitation to access a cache either in ascending or in descending order.

In practice, such an ideal DCA instruction does not exist in ISAs but it can be indirectly implemented by combining a set of existing DCA instructions that together totally cover all fields of the ideal instruction. Representative examples of such special purpose instructions, which can be considered as DCAs, are present in RISC architectures, such as MIPS, ARM and SPARC architectures and in CISC architectures such as x86 architectures. A detailed presentation of these existing DCA instructions is included in the thesis manuscript.

4 SBST methodology for small caches

The methodology targets all cache arrays for both data and instructions (either L1 caches or TLBs) and is suitable for all three cache organizations with any write policy. An SBST technique that targets L1 caches cannot be cache resident, since the actual L1 cache is under test. However, this is not a limitation in case of on-line testing, since the test routines can be stored and executed from either L2 cache or the chip's main memory that is available at test time. Moreover, when the SBST methodology targets TLBs, the SBST routine should be placed in a non-pageable memory location that is not cached to the instruction TLB since the actual TLB arrays are under test.

The proposed SBST methodology implements low cost SBST March tests that target cache arrays by taking advantage of existing debug-diagnostic instructions in

modern ISAs, as described in Section 3. These instructions must cover in total the fields of the ideal DCA instruction to overcome the testability challenges of the cache arrays. The main feature of the methodology is low cost March write/read implementation due to high controllability and observability of DCA instructions. The proposed SBST methodology is briefly summarized in Figure 3 for L1 caches and TLBs respectively and is thoroughly presented in the thesis manuscript. The main features of the proposed methodology are:

- Low cost March writes due to the high controllability of DCA write instructions.
- Low cost March reads due to the high observability of DCA read instructions.
- Low cost March comparisons due to the special features of DCA compare instructions.
- Low cost March reads for tag arrays by exploiting performance monitoring hardware, if DCA read instructions do not exist in the ISA.
- Low cost March comparisons for TLB tag arrays by exploiting the trap handler mechanism, if DCA compare instructions do not exist in the ISA.
- Test response compaction to comply with on-line testing requirements.

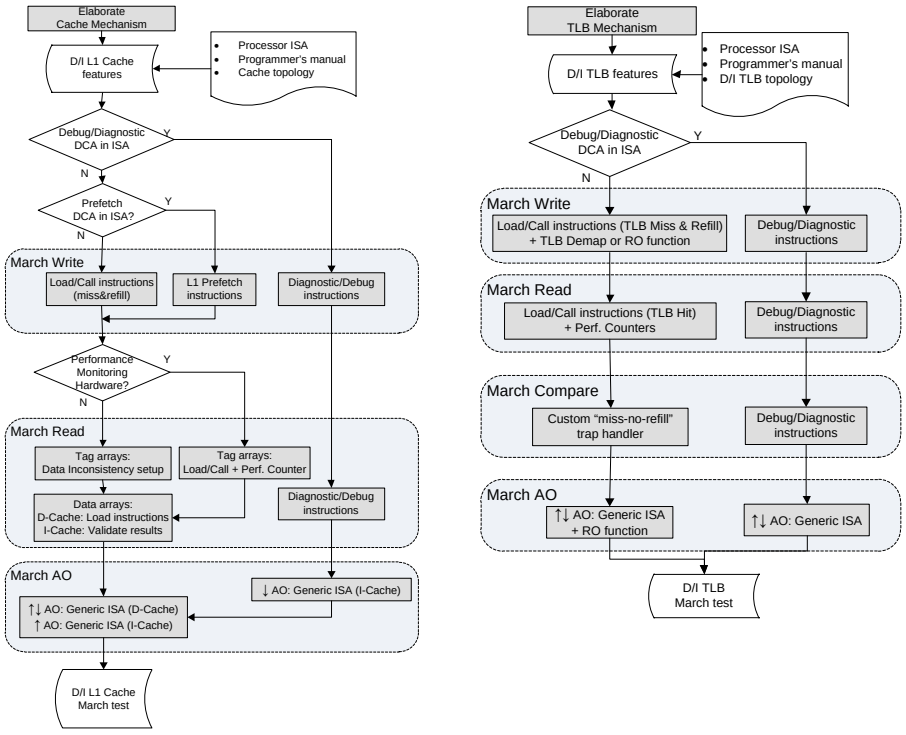

Figure 3: SBST Methodology for a) L1 caches b) TLBs

5 Multithreaded optimization for multi-bank L1 caches

We propose a multithreaded optimization of the SBST methodology that elaborates the low level multiple bank organization of modern cache designs to exploit the thread level parallelism of modern multithreaded multicore processors and speedup March test execution time.

In cache designs, SRAM arrays are partitioned into sub-banks to save power or to tweak the cache dimensions or even multiple banks to minimize internal wiring delay. Such cache designs that consist of one bank but multiple sub-banks are called Uniform Cache Architectures (UCA) whereas these that that consist of both multiple bank and sub-banks are called Static/Dynamic Non-Uniform Cache Architectures (S-NUCA/ D-NUCA) based on the way that the mapping of data into cache banks is achieved [12].

In UCA and S-NUCA architectures, the mapping of data into cache banks and sub-banks is predetermined based on the block index of the architecture and thus can reside in only one bank of the cache. Several cache modeling tools (e.g. Cacti) enable fast exploration of the cache design space by automatically choosing the optimal bank and sub-bank count, size, and orientation of UCA and S-NUCA architectures. Each sub-bank has a separate memory array, decoder, write drivers, and sense amplifier. The S-NUCA cache consists of multiple banks. Every bank is organized as a UCA with multiple sub-banks.

Each cache sub-bank in UCA and S-NUCA multibank architectures can be considered as a separate SRAM array with a distinct functional fault set, since no coupling faults can occur between memory cells of different cache banks and sub-banks. Therefore, an on-line test strategy that considers every sub-bank as an independent memory array can be developed since the memory mapping is known to test engineers.

The proposed optimization considers exclusive fault sets for every L1 cache array sub-bank and proposes a fine-tuned clustering of the applied March tests in smaller subroutines based on the information provided for the physical implementation of a multibank cache (sub-bank address mapping, address scrambling etc.). Every cluster targets a sub-bank of the cache and the logical union of all clusters ends up to the initial March test for the whole cache array under test. These March test clusters target separate cache array sub-banks which can be executed concurrently without diminishing the March test effectiveness. Such a clustering is well suited to multithreaded processors, where concurrent execution can be achieved by assigning test clusters to different threads and hence SBST March test execution time is divided by the factor of available number of test threads. The test threads are dynamically scheduled software threads among the other executed processes. These test threads should be isolated during on-line testing in order to prevent the rest of the software processes to corrupt the March test effectiveness. In case that the March tests clusters outnumber the available threads, more than one March test clusters are assigned to every thread. Finally, the proposed multithreaded optimization is suitable both for simultaneous multithreading and interleaved multithreaded architectures, since it is independent of the way that the threads are issued and it is compatible with any resource allocation policies (e.g. physical register file size, register windows, register renaming e.t.c).

6 Case studies: LEON3, OpenRISC 1200, OpenSPARC T1

We have applied the proposed SBST methodology to three processor benchmarks: a) LEON3, b) OpenRISC 1200 and c) OpenSPARC T1. We have also applied the multithreaded optimization to the L1 caches of OpenSparc T1.
In order to evaluate the effectiveness of the self-test routines we have used RAMSES memory fault simulator [14].

The first benchmark is LEON3, a publicly available processor designed by Aeroflex Gaisler that implements a SPARC V8 compliant architecture. The SPARC V8 ISA, that LEON3 implements, includes privileged store/load instructions, denoted as alternate load/store (*lda/sta* instructions). These instructions can directly access cache arrays for diagnostic purposes by specifying alternate space identifiers (ASIs) that are defined by the SPARC architecture for both write and read access at supervisor level. These instructions have been used as DCA instructions for March write/read operations to apply and read the test patterns in SBST routines.

The second benchmark is OpenRISC 1200, a publicly available processor core. OpenRISC 1200 lacks debug-diagnostic instructions in its ISA to access the cache arrays. However, it includes a cache prefetch mechanism for both data and instruction L1 caches and maps prefetch operations to valid instructions. These instructions have been utilized as DCA instructions for March write operations. For March read operations, generic load and call instructions have been used. The observability of the March tests has been improved by exploiting the performance counters.

The third benchmark is OpenSPARC T1 that includes both data and instruction L1 caches and fully functional TLBs. Therefore, the SBST methodology has been fully applied to both the L1 cache and TLB arrays. OpenSPARC T1 implements a SPARC V9 compliant ISA and includes privileged store/load instructions, denoted as alternate load/store (*ldxa/stxa* instructions). We have exploited these alternate load/store instructions for March write/read operations to directly access all cache arrays for both L1 caches and TLBs for March write and March read operations at low cost by utilizing the appropriate ASI at the hypervisor level. SPARC V9 ISA does not implement a debug/diagnostic compare instruction for implementing the March compare operation. Therefore, a custom “miss-no-refill” trap handler has been utilized to implement March compare operations to test DTLB-Tag and ITLB-Tag CAM arrays for comparison faults. We have also applied the multithread optimization to the March tests that target the four SRAM arrays of both data and instruction L1 caches of a SPARC core.

We have applied a set of March tests with different test complexities to both data and instruction caches. Solid data backgrounds (all-zero/all-ones) have been used to all tests. The test quality is verified with RAMSES for all the applied March tests and 100% fault coverage has been verified for all cache arrays.

Detailed experimental results and comparisons both for the test code size and test execution time of these March tests are provided in the thesis manuscript and demonstrate the effectiveness of the proposed methodology, its high adaptability and the significant improvements in terms of test time (86% for instruction L1 cache, 87% for the data L1 cache, about 40% for D-TLB and about 82% for I-TLB) and test code size (83% for instruction L1 cache, 86% for the data L1 cache, 3% for D-TLB and 35% for I-TLB) when the methodology is applied to the same benchmarks (LEON3

for L1 caches and OpenSPARC T1 for TLBs) and such DCA instructions are exploited compared to SBST solutions that don't utilize these types of instructions. Moreover, experimental results show a speedup of more than 1.7 (for two threads) and more than 3.7 (for four threads) in test time when the proposed multithreaded optimization is applied to the L1 caches of OpenSPARC T1.

7 Performance overhead evaluation

In this section, we present the evaluation framework that was utilized to estimate the performance overhead of the proposed SBST routines. We have implemented several on-line periodic testing scenarios and we will present detailed statistics of the performance overhead introduced in a typical workload under these test scenarios.

We have utilized a SunFire T2000 server running a set of multithreaded programs -the PARSEC benchmark suite- over Solaris 10 to evaluate the performance overhead of the deployed SBST routines. Our server is powered by a quad-core UltraSPARC T1 processor. OpenSPARC T1 processor is the free version of UltraSPARC T1 that is utilized in SunFire T2000 servers. Hence, the SBST routines that were developed above for OpenSPARC T1 processor can be directly compiled to our server to evaluate their performance overhead.

We have selected the optimized 2-thread March C- SBST routine that targets the data L1 cache (both DL1-Tag and DL1-Data) to be utilized as our self-test routine in the evaluation framework. The test time of this test routine has been measured about 1.2sec in our system. Any other SBST routine (or a set of March tests) could have been selected. Since we do not have access on the hypervisor level on the UltraSPARC T1 processor of a native system, we have slightly altered the SBST test routines in order to comply with Solaris OS limitations on executing hyper privileged instructions.

The modified SBST routines have the same memory footprint and test execution time with the OpernSPARC's T1 implemented routines, thus the modified self-test routine is sufficient for studying the performance impact of the proposed on-line self-tests.

Here after, we will utilize the terms of Test Period (TP) and Test Latency (TL) as described below:

- Test period (TP) and is the amount of time from the beginning of a self-test on a core to the beginning of the next self-test on the same core.
- Test latency (TL) is the duration of an on-line self-test.

We have applied several on-line testing scenarios with a fixed TL (1.2sec) and several short TPs (< 1min) that are suitable for detecting early-life failures on two different framework configurations, a *1core/4thread* setup and a *4core/16thread* setup as described below in detail.

- ***1core/4threads setup, TL=1.2sec, TP=2, 15 and 60 sec***

Firstly, Solaris capability of creating virtual processor sets has been exploited to isolate a single SPARC core from OpenSPARC T1 and both PARSEC workload

applications and SBST routine have been set -by utilizing Light Weight Process (LWP) binding- to be executed in this SPARC core.

Note that we have selected to isolate an idle core that does not execute any other OS process in order to evaluate the real performance overhead due to the SBST routine's periodic execution only. The PARSEC suite has been configured to be executed by four threads (the maximum number of threads in the core). These four threads share the same L1 cache. Hence, both workload and March test application access the same cache SRAM arrays. Afterwards, several TPs have been selected to represent different test scenarios. For example a demanding testing scenario may require intensive test period (e.g. TP=2sec) while a more relaxed test scenario may require less intensive test periods (e.g. TP=60sec).

- ***4cores/16threads setup, TL=1.2sec, TP=10, 30 and 60 sec***

In this configuration we have utilized all four available SPARC cores of our server. Hence, apart from the PARSEC applications and the periodic execution of the SBST routine, the OS processes were also executed in the background. The PARSEC suite has been configured to be executed by all the available sixteen threads of our system (four threads per SPARC core). A script has been composed to call the SBST routine for every SPARC core in a round-robin way in every TP. Moreover, in each SBST routine execution, the script forms a virtual processor set of the 4 threads that belong to the core under test to ensure that the SBST routine will be executed only by the selected core under test. This is a critical requirement to guarantee the test quality by preventing the test patterns to be stored in L1 cache of other cores, apart from the core under test. Several TPs have been selected to represent different test scenarios, in a similar way that was presented for the 1core/4threads setup. The selected TPs were longer in these case studies due to the need of executing the same SBST routine four times (one for the L1 cache of each SPARC core).

The system in both configuration setups was configured to execute all the 13 multithreaded programs of the PARSEC suite. All PARSEC programs have been compiled to utilize the pthreads parallelization model and the native dataset was utilized in all simulations. After estimating the workload execution time in both configuration setups without test, the PARSEC programs were executed several times while the SBST routine was scheduled to be executed in the background at a fixed TP in every test scenario for both configurations.

Detailed statistics for the performance overhead for each test scenario are provided in the thesis manuscript and even in the more demanding test scenario of a quad core processor does not exceed the 11% of the performance of the system without test. Considering that this performance penalty refers to a demanding on-line periodic self-test scenario that applies March C- algorithm to both D-Tag and D-Data SRAM arrays for all four L1 caches of the quad core system every 10sec, such performance degradation can be affordable when a strict test scenario is required. In contrary, in a relaxed test scenario (e.g. not more than a test per minute), the performance overhead is lower. For example, when the SBST routine is periodically executed every minute in a single core system (e.g. a single core system can be an embedded processor), the performance overhead is less than 1%, thus negligible. Thus, the proposed SBST methodology can periodically apply March tests to L1 caches effectively during the system's lifetime with acceptable performance overhead in workload execution.

Conclusions

We have presented a low cost SBST fault detection methodology for small embedded cache memories that leverages existing powerful mechanisms of modern ISAs by utilizing existing DCA instructions, exploits the native performance monitoring hardware and the trap handling mechanisms which are available in modern microprocessors. Furthermore, a multithreaded optimization of the proposed methodology has been presented. The methodology has been applied to three processor benchmarks: a) OpenRISC 1200 b) LEON3 and c) OpenSPARC T1. Experimental results for several March test implementations demonstrate the significant improvements in terms of test time and test code size when the methodology is applied to the same benchmarks (LEON3 for L1 caches and OpenSPARC T1 for TLBs) and such DCA instructions are exploited compared to SBST solutions that don't utilize these types of instructions. Finally, a performance evaluation framework has been presented to demonstrate that the implemented SBST routines have acceptable performance overhead for periodic on-line testing.

References

1. S. Hamdioui et al, "March SS: A Test for All Static Simple RAM Faults", in Proc. of IEEE Int'l Workshop Memory Technology, Design and Testing (MTDT), 2002
2. K.J. Lin, C.W. Wu, "Testing content-addressable memories using functional fault models and march-like algorithms", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.19, no.5, pp.577-588, May 2000
3. A.J. van de Goor, T.J.W Verhallen, "Functional testing of current microprocessors", in Proc. of International Test Conference (ITC), 1992, pp.684-695.
4. J. Sosnowski, "In-system testing of cache memories", in Proc. of International Test Conference (ITC), 1995, pp.384-393.
5. S.M. Al-Harbi et al. "A Methodology for Transforming Memory Tests for In-System Testing of Direct Mapped Cache Tags", in Proc. of VLSI Test Symposium (VTS), 1998.
6. J. Sosnowski, "Improving Software Based Self-Testing for Cache Memories," in Proc. of Design and Test Workshop (DTW), 2007
7. M. Tuna et al. "Software-Based Self-Test Strategies for Memory Caches of RISC Processor Cores", in Proc. of IEEE Latin-American Test Workshop, 2007, pp.124-130.
8. S. Alpe et al., "Applying March Tests to K-Way Set-Associative Cache Memories", in Proc. of the 13th European Test Symposium (ETS), 2008, pp.77-83.
9. Y.C. Lin, Y.Y. Tsai, K.J. Lee, C.W. Yen, C.H. Chen, "A Software-Based Test Methodology for Direct Mapped Data Cache", in Proc. of Asian Test Symposium, 2008
10. W.J. Perez et al., "A Hybrid Approach to the Test of Cache Memory Controllers Embedded in SoCs", in Proc. of IEEE International On-Line Testing Symposium (IOLTS), 2008, pp.143-148.
11. W.J. Perez et al., "On the Generation of Functional Test Programs for the Cache Replacement Logic", in Proc. of IEEE Asian Test Symposium (ATS), 2009, pp.418-423.
12. A.J. van de Goor et al., "Memory testing with a RISC microcontroller", in Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010
13. S. Di Carlo, P. Prinetto, A. Savino, "Software-Based Self-Test of Set-Associative Cache Memories," IEEE Transactions on Computers, Vol.60, no.7, pp.1030-1044, July 2011
14. C.F. Wu et al, "RAMSES: a fast memory fault simulator", in Proc. of the International Symposium on Defect and Fault Tolerance in VLSI Systems, 1999, pp.165-173