A Software-Based Self-Test Methodology for On-Line Testing of Processor Caches

G. Theodorou, N. Kranitis, A. Paschalis, D. Gizopoulos

Dept. of Informatics & Telecom., Univ. of Athens, Greece – {gthe, nkran, paschali, dgizop}@di.uoa.gr

Abstract

Nowadays, on-line testing is essential for modern high-density microprocessors to detect either latent hardware defects or new defects appearing during lifetime both in logic and memory modules. For cache arrays, the flexibility to apply online different March tests is a critical requirement. For small memory arrays that may lack programmable Memory Built-In Self-Test (MBIST) circuitry, such as L1 cache arrays, Software-Based Self-Test (SBST) can be a flexible and low-cost solution for on-line March test application. In this paper, an SBST program development methodology is proposed for on-line periodic testing of L1 data and instruction cache, both for tag and data arrays. The proposed SBST methodology utilizes existing special purpose instructions that modern Instruction Set Architectures (ISAs) implement to access caches for debugging and performance purposes, termed hereafter Direct Cache Access (DCA) instructions, as well as, performance monitoring mechanisms to overcome testability challenges. The methodology has been applied to 2 processor benchmarks, OpenRISC and LEON3 to demonstrate its high adaptability, and experimental comparison results against previous contributions show that the utilization of DCA instructions significantly improves test code size (83%) and test duration (72%) when applied to the same benchmark (LEON3).

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 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.

In manufacturing testing, MBIST schemes target the detection of memory functional faults [1] caused by cell spot defects. To achieve this, a large set of March tests is applied under different stress combinations to ensure detecting all possible faults in the caches of microprocessors. This set is not optimal and takes excessively long test time to be applied [2].

In on-line testing, parity and Error Correction Code (ECC) schemes are widely used to enhance embedded memory reliability and they are also utilized in embedded cache memories. These schemes are used to protect the memory arrays from soft errors. ECC schemes have two weaknesses i.e. fault accumulation effect and limited detect capability of memory functional faults [3]. Combined on-line MBIST schemes and ECC schemes can be used to overcome these weaknesses.

Programmable MBIST schemes [4], which have been integrated for manufacturing testing purposes, are reusable for on-line testing, but only a subset of the abovementioned March tests can be applied due to performance overhead limitations. The programmable MBIST schemes provide the flexibility to apply different March tests, as well as, future March tests for new memory fault models which is a critical feature for on-line testing. However, the reuse of MBIST schemes during on-line testing imposes much higher power density [5] that affects system’s reliability.

Besides, small memory arrays that have size in the order of Kbytes (such as register files, FIFOs, small caches, cache tag arrays etc.) may not justify the cost of adding programmable MBIST schemes because of its impact on chip area and performance. Semiconductor industry has acknowledged this problem and industrial solutions have been proposed as a low-cost alternative to MBIST. For example, in [6], Macrotest, a scan-based technique was proposed to test a number of small embedded memories (including L1 caches) on the AMD Athlon™ processor during manufacturing testing.

Software-Based Self-Test (SBST) has recently emerged as an effective complementary solution for microprocessor and embedded processor manufacturing [7], as well as, periodic on-line testing [8]. Key microprocessor companies (Sun [9], Intel [10]) have recognized the potential of SBST adopting it in their test flows. Recently, in [11], a taxonomy for different SBST methodologies has been presented. SBST is a non-intrusive approach that embeds a “software tester” with the form of a self-test program in the processor’s on-chip memory. This way SBST imposes zero hardware and performance overhead during normal operation, as well as, ordinary power density during on-line testing. Moreover, SBST can be easily reused in field for power up diagnostics or periodic on-line testing to add dependability features. Apart from these, in the case of on-line memory testing, SBST has increased flexibility to apply any kind of March tests. Hence, SBST can face successfully the challenges of on-line testing of small memory arrays that lack programmable MBIST.

This work is partially funded by National & Kapodistrian University of Athens (N.K.U.A.)
L1 cache SRAM arrays belong to this category of small memory arrays, since L1 cache sizes are up to 32 kilobytes in most of the modern processors. L1 cache arrays due to their size may totally or partially lack programmable MBIST circuitry (e.g. MBIST circuitry is not included in the tag array). Hardware defects in L1 cache arrays during normal operation may cause either erroneous cache misses that degrade the system’s performance (defects in the tag array), or unpredicted system behaviour (defects in the data array). Hence, online testing is essential to avoid system’s performance degradation and erroneous behaviour.

Recently, research community has acknowledged the potential of utilizing SBST techniques in cache testing and several SBST approaches have been proposed [12] - [19]. Some of them describe only pseudo-instruction templates of the implemented March operations ([12], [14], [15]), while most of them provide experimental results on processor benchmarks ([13], [15] - [19]).

In this paper, an SBST program development methodology for on-line testing of processor cache arrays for L1 data and instruction caches, both for tag and data arrays, is presented. The proposed methodology can be applied to both direct mapped and set-associative cache organizations while it is independent of cache write policy. The methodology leverages the inherent power of modern ISAs by utilizing special purpose instructions that we denote as Direct Cache Access (DCA) instructions. Moreover, the methodology exploits the native monitoring hardware (performance counters) that is available in modern architectures, as well. By combining these two features the proposed methodology applies March write and read operations in a low cost way (w.r.t. previous SBST approaches) and verifies the test result with a compact response to comply with periodic on-line testing. A preliminary version of the proposed methodology (targeting only tag arrays without the concept of DCA instructions) was presented in [18].

Experimental results on two modern processor benchmarks, OpenRISC and LEON3 demonstrate the high adaptability of the proposed methodology and the significant improvements in terms of test duration and test code size when compared with the previous SBST contributions that target caches [15], [19].

The rest of the paper is organized as follows: Section 2 presents cache array testability challenges. Section 3 presents DCA instructions. Section 4 introduces the proposed SBST methodology. Section 5 demonstrates the application of the proposed methodology on the processor benchmarks. Section 6 provides comparisons while Section 7 concludes the paper.

2. Cache Array Testability Challenges

There are three cache organizations: direct mapped, fully-associative and set-associative. In processors, direct mapped and set-associative caches are used.

A typical L1 cache organization comprises of two SRAM memory arrays, the data array and the tag array. Further down, those arrays will be denoted as D-Data, D-Tag, I-Data and I-Tag for the data and instruction cache, respectively. Cache arrays are not directly visible to the assembly language programmer through a generic ISA since test patterns are not applied directly and test responses are not monitored directly through a software routine. The challenges of accessing and thus testing each of the four cache arrays are summarized in Table 1.

<table>
<thead>
<tr>
<th>Cache Arrays</th>
<th>D-Data</th>
<th>D-Tag</th>
<th>I-Data</th>
<th>I-Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct access from generic ISA</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Indirect March write (controllability)</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Indirect March read (observability)</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Direct Test Pattern composition</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Descending Order in March test</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>

Table 1: Cache Array Testability Challenges

D-Data array stores the actual information that has to be cached and can be easily accessed indirectly by using load/store generic instructions. Test patterns can be easily composed by initializing the main memory with the corresponding test vectors by using store instructions and then by writing them to D-Data array (March write operation) either by a store instruction that misses (when write-allocate is supported) or by a load instruction that misses and refills the cache lines. Generic load instructions can afterwards read the test vectors to registers, hence March read operations can be mapped to such ISA instructions.

D-Tag and I-Tag arrays store address information. Every time a cache line is inserted into the D-Data or the I-Data array, a tag entry is also introduced. As those arrays store the upper part of the addresses, test patterns should be valid addresses. Hence, test pattern composition is challenging for both arrays due to potential limitations in accessing several memory segments. Test patterns can be written to D-Tag array by using a store/load instruction that misses as above, while test patterns can be written to I-Tag array by using a call instruction. Contents of both D-Tag and I-Tag arrays are not directly readable by generic ISA instructions and thus mapping March read operations to ISA is challenging and can be only accomplished by identifying unexpected cache misses.

I-Data array encounters the majority of testability challenges since test patterns must be composed by valid instructions. Hence, the limitations mainly due to opcode encoding have to be encountered.

Moreover, both instruction cache arrays (I-Tag and I-Data) have two additional challenges that should be addressed when using SBST routines to apply test patterns. The first challenge is due to the fact that the SBST routines affect the instruction cache testing since they are also placed in the same cache during test application. The second challenge deals with the...
implementation of the descending addressing order in March operations for the instruction cache testing.

The proposed methodology not only overcomes the above mentioned testability challenges, but also optimizes the SBST routines in terms of test code size and test duration.

3. DCA Instructions in Modern ISAs

So far, all previous SBST approaches overcome the above mentioned challenges by using generic instructions to access cache arrays both for write and read operations indirectly. Fortunately, modern ISAs include special instructions for debug-diagnostic and performance purposes that provide direct controllability and observability of cache arrays. These instructions are suitable for cache SBST and are termed Direct Cache Access (DCA) instructions further down.

According to cache structure, an ideal DCA instruction that overcomes all mentioned above testability challenges has to contain the following fields:

- Cache Way Selection (WS) field.
- Cache Set Selection (SS) field.
- Cache Line Word Selection (LWS) field.
- From/To data Address (A) field.
- Data/Tag Array Selection (AS) field.
- Write/Read operation (WR) selection field.

![Figure 1: Ideal DCA instruction](image)

In Figure 1 the format of an ideal DCA instruction and the way that every field is utilized to access a specific 2-way set associative cache, is presented.

In detail, the data or tag array of this cache (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 test patterns could be composed by initializing either a general purpose register or a memory location, whose access is controlled by the A field. Note that, such an instruction has no limitation in accessing a cache either in ascending or in descending order.

In practice, such an ideal DCA instruction is not present in ISAs. But it can be indirectly implemented by combining a set of existing DCA instructions that cover in total all fields of the ideal one. Representative examples of such special purpose instructions, which can be characterized as DCA instructions, are present in RISC architectures, such as SPARC and ARM.

SPARC architecture implements alternate space identifier (ASI) store/load instructions (e.g. sta/lsta instructions in LEON3 and stx/lx instructions in OpenSPARC T1). These instructions are utilized to access embedded RAMs through a SPARC diagnostic access bus [20] that bridges these embedded RAMs with the main memory or the processor’s register file. Every embedded RAM, including L1 cache arrays, can be accessed by using ASI store/load instructions in order to implement write/read operations. Especially, for L1 cache arrays, dedicated ASIs for all the four cache arrays (D-Data, D-Tag, I-Data, I-Tag) are defined to access cache in hypervisor level. This way, the ASI field of the ASI store/load instructions maps to the AS field of the ideal instruction. Also, the ASI store/load instructions contain fields that map to WS, SS and LWS fields of the ideal instruction, respectively. Hence, these instructions, when combined, cover all fields of the ideal DCA instruction and can be effectively used to implement March operations.

ARM architecture implements system control coprocessor (CP15) debug operations (MRC and MCR instructions) for accessing L1 cache arrays. These instructions are executed in secure privileged mode and provide great visibility into the cache state by interrupting the program flow to execute them. They can access both tag and data arrays of processor L1 caches for write/read operations by transferring the array contents from/to system array debug data registers without executing store/load instructions. Hence, these instructions are quite similar to the ideal DCA one and can be effectively used to implement March operations.

Apart from this, instructions that implement the L1 cache prefetch mechanism can be considered as DCA write-only instructions to achieve cache controllability. Prefetch mechanisms are utilized to initialize cache contents in order to reduce cache miss ratio of applications. Such prefetch mechanisms are present in all modern microprocessors (e.g. dcbt instruction in IBM’s PowerPC, prefetch0 in Intel’s Pentium, pld instruction in ARM’s Cortex and Intel’s Xscale). The instructions that implement these mechanisms initialize cache sets with main memory blocks that are selected by an instruction field. The prefetch instructions, when utilized, can access any cache line, but not any word inside the cache line for write operations. Prefetch instructions have to be combined with generic instructions to achieve cache observability, as well.
4. Cache SBST Methodology

This section introduces an SBST methodology that applies March tests to cache arrays.

4.1 Notations of March Tests

First, a short description of March tests notations will be presented. A March test consists of a sequence of March elements. Each March element consists of a sequence of operations (writes and reads). The way the sequence proceeds to the next step is determined by the address order (AO) and is denoted by the "↑" and "↓" symbols for ascending or descending order, respectively. Every March test is delimited by the bracket pair "{"", every March element (M0, M1,...) is delimited by the bracket pair "( )" and March operations are symbolized as W, W, R, R, respectively [21]. Word-oriented March SS [1], a representative contemporary March test, is shown below.

\[
\begin{align*}
\text{March Write} & \quad \text{March Read} \\
\{ (W_{DB})_1 \} \quad \{ (R_{DB})_1 \} & \quad \{ (R_{DB}, W_{DB})_1 \} \\
\{ (R_{DB}, W_{DB})_2 \} \quad \{ (R_{DB})_1 \} & \quad \{ (R_{DB}, W_{DB})_1 \} \quad \{ (R_{DB}, W_{DB})_2 \} \quad \{ (R_{DB})_3 \} \\
\{ (R_{DB})_4 \} \quad \{ (R_{DB}, W_{DB})_3 \} \quad \{ (R_{DB}, W_{DB})_4 \} & \quad \{ (R_{DB}, W_{DB})_5 \} \\
\end{align*}
\]

4.2 SBST March Test Development

The proposed methodology leverages the inherent power of the ISA to implement March test operations in a low cost way by utilizing the above mentioned DCA instructions.

The methodology targets both data and instruction caches and it is suitable for both direct mapped and set-associative cache topologies with any write policy. Such an SBST technique that targets L1 caches cannot be cache resident, since the actual 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. The proposed SBST methodology is summarized in Figure 2. The main features of the proposed methodology are:

- Low cost March write implementation due to high controllability of DCA write instructions.
- Low cost March read implementation due to the high observability of DCA read instructions.
- Low cost March read implementation for tag arrays by exploiting performance monitoring hardware, if DCA read instructions do not exist in the ISA.
- Test response compaction to comply with on-line testing requirements [8].

The methodology begins by extracting all available information about the cache system (e.g. the cache topology, the replacement algorithm, the size of the cache arrays, existence of performance monitoring hardware for caches etc.), as well as, a thorough examination of the implemented ISA. This information is provided by the microarchitecture manual, the designer’s manual and the programmer’s manual.

![Figure 2: Cache SBST March test development methodology](image-url)
4.2.1 Data Cache

The proposed methodology implements March element operations for the data cache arrays as follows:

- **Write operations** \((W_m, W_v)\) are implemented by utilizing debug-diagnostic write instructions when available in the ISA. As above mentioned such instructions access both D-Tag and D-Data arrays and cover the desired fields \((WS, SS, WLS, A)\) fields) to initialize any cache line word from the lower memory hierarchy level (or the register file). This memory block (or register) must contain the test pattern to initialize either the D-Data array or the D-Tag array. If debug-diagnostic instructions do not exist in the ISA, L1 cache prefetch instruction is utilized for March write operation. Prefetch instructions initialize cache lines with blocks from memory. Tag is initialized with the high part of the addresses. Hence, a targeted selection of both the content and the address of the memory block can initialize both arrays with the desired test patterns. In case that DCA write instructions are not present in the ISA, generic load instructions for cache load miss and refill can be utilized to implement March write operation as proposed in [13] and [15].

- **Read operations** \((R_m, R_v)\) are implemented by utilizing debug-diagnostic read instructions when available in the ISA. These instructions access both D-Tag and D-Data arrays for read access in the same way that debug-diagnostic write instructions do (through WS, SS, WLS and A fields) to copy any cache line word to a memory location (or a register). Afterwards, the content of this memory location (or register) can be easily verified to complete March read operation. If debug-diagnostic instructions do not exist in the ISA, March read operations can be indirectly implemented by generic load instructions. In a load instruction, the expected test pattern in D-Data array (the one that was written by a March write operation) will be copied to a register where it can be verified. Furthermore, the same load instruction should produce a cache hit (or avoid a cache miss) if a successful March write operation in D-Tag has been preceded. This hit can be concurrently monitored by a performance counter with no extra cost, if available. In case that neither debug-diagnostic instructions nor performance counters are available, data inconsistency should be set up to validate March write operations for the D-Tag array as proposed in [13] and [15].

- **March addressing order (AO)** \(\uparrow, \downarrow\) is implemented by utilizing generic ISA instructions. In data cache, the implementation of both ascending and descending AO is straightforward. Generic ISA instructions can be easily composed to implement a software loop to march D-Data and D-Tag arrays both in ascending and descending AO.

### March SS for data cache

```c
// set associative : \(N_s = \# \text{ of sets}
// direct mapped : \(N_s = \# \text{ of lines}
```

```c
D = DB_{DATA}
D = DB_{DATA}
```

```c
//M1 element \( I ((R_m, R_v, W_m, R_m, W_m)) \)
for \((i=0; i<N_s; i=i+1)\) // Access all sets/lines of cache
{
A = create_address \(DB_{TAG} :: i \rightarrow B\) // Address tag = DB_{TAG}
A = create_address \(DB_{TAG} :: i \rightarrow B\) // Address tag = DB_{TAG}
DD_read \((A, D)\) // or m x Load \((A)\) + Perf Counter
DD_read \((A, D)\) // or m x Load \((A)\) + Perf Counter
DD_write \((A, D)\) // or Prefetch_block \((A)\)
DD_write \((A, D)\) // or Prefetch_block \((A)\)
DD_write \((A, D)\) // or Prefetch_block \((A)\)
}
```

**Figure 3: SBST routine for March SS test for data caches**

The proposed algorithmic notation of an SBST routine for both D-Data and D-Tag arrays of a data cache is shown in Figure 3. \(N_s\) is the number of cache sets (cache lines in case of a direct mapped organization) in a set-associative organization, \(B\) is the offset part of address to access a word inside the cache line, \((DB_{DATA}, DB_{DATA})\) is the DB pair for the D-Data array, \((DB_{TAG}, DB_{TAG})\) is the DB pair for the D-Tag array. Addresses \(A\) and \(A\) are created by the concatenation of the tag DBs, the parameter \(I\) (set/line index) and a value for \(B\) to access a word inside the cache line. Diagnostic-debug (DD) instructions as DCA instructions are used both for write and read operations when available. \(DD_write(A, D)\) instructions initialize a cache line/word in address \(A\) with content \(D\). \(DD_read(A, D)\) instructions read a cache line/word in address \(A\) with content \(D\). If such instructions do not exist in the ISA, a Prefetch \((A)\) instruction is utilized to initialize an L1 cache line with the memory block that is located in address \(A\) and a sequence of \(m\) load instructions \((Load(A))\) is utilized to access all words in the cache line \((m\) is the number of words per line). In every load instruction, a performance counter concurrently monitors for a miss to verify a successful March write operation. If the diagnostic-debug instructions cannot access every cache way explicitly (lack of WS field), the routine is repeated \(k\) times for a \(k\)-way set-associative cache with LRU replacement with \(k\) different DB sets to cover the cache lines in every cache way.

The methodology can develop SBST routines for any March test. If debug-diagnostic instructions are utilized, the methodology can apply any DB including the most common ones that are used in the industry tests (solid, checkerboard, column stripes and row stripes [4]) to both D-Data and D-Tag arrays. Otherwise, if such instructions do not exist, the DB pair \((DB_{TAG}, DB_{TAG})\) should be defined in a memory data segment that is allowed by the virtual memory mechanism.
Finally, the proposed algorithmic notation can be easily simplified to test only the D-Tag array, if the D-Data array test is not required (e.g., a programmable MBIST scheme is present for the D-Data array).

### 4.2.2 Instruction Cache

Instruction cache is used to store instructions fetched from lower levels of memory hierarchy. Even though instruction cache is similar in structure to data cache, the additional three testability challenges that have been described in section 2 should be addressed.

The first challenge is due to the fact that the SBST routines affect the instruction cache testing since they are also placed in the same cache during test application. Fetching the actual SBST routine in the instruction cache spoils the effectiveness of the test as the test patterns are substituted by the SBST code. In order to overcome this challenge, the SBST routine should be placed in a non-cacheable area of main memory. Alternatively, if the architecture cannot define a non-cacheable area, the cache enable/disable mechanisms can be used to isolate the SBST routine from cache. The utilization of the cache enable/disable mechanism increases the test code size and test duration of the SBST routines. The rest of the challenges are overcome as described below. The methodology implements March operations for the I-Data and I-Tag arrays, as follows:

- **Write operations** \( (W_{on}, W_{pi}) \) are implemented by utilizing debug-diagnostic write instructions when available in the ISA. The instructions initialize both I-Data and I-Tag array similarly to data cache arrays. Debug-diagnostic write instructions are not limited to utilize valid instructions for I-Data initialization, since the cache content must be invalidated when such instructions are executed. Hence, debug-diagnostic instructions overcome the testability challenge of composing test patterns for the I-Data array as they can initialize the I-Data array with any desired DB. If debug-diagnostic instructions do not exist in the ISA, L1 cache prefetch instruction is utilized for March write operation similarly to data cache. When prefetch instructions are utilized, the test patterns should be valid instructions and are composed by instructions that have complementary opcode/fields. Every cache line should contain such valid instructions and the combination of them will form the desired test pattern. One return instruction should be included in every cache line to facilitate the control flow back to the SBST routine. In case that DCA write instructions are not present in the ISA, generic call instructions for cache miss and refill can be utilized to implement March write operation as proposed in [13] and [19].

- **Read operations** \( (R_{on}, R_{pi}) \) are implemented by utilizing debug-diagnostic read instructions when available in the ISA. These instructions copy the content of any cache line word (I-Data) or any tag entry (I-Tag) to a memory location (or a register) by controlling the \( WS, SS, WLS \) and \( A \) fields, similarly to data cache. Afterwards, the memory content (or register) can be easily verified to complete March read operation. The debug-diagnostic read instructions have high observability features since they implement direct access to the instruction cache arrays that is missing through generic ISA instructions. If debug-diagnostic instructions do not exist in the ISA, a call instruction that targets the desired cache line should be utilized to fetch and execute all the instructions of the cache line. In order to verify successful March read operations for the I-Data, the executed instructions that have been used in the March write operation as test patterns should produce an unambiguous result that can be easily verified for its correctness as proposed in [19]. In order to verify successful March read operations for the I-Tag array, a call instruction should be executed to fetch instructions that are placed in the memory in a segment that is dictated by the desired test patterns (the high part of the addresses should compose the test pattern for the I-Tag array). If a successful March write operation in I-Tag has been preceded, this call instruction will produce a cache hit (or avoid a cache miss) that can be directly monitored by a performance counter, if available. Note that a successful test will leave the performance counter’s value to zero (zero cache misses) at the end of the test. In case that no performance monitoring hardware is available, data inconsistency should be set up to validate successful March write operations for the I-Tag array as described in [19].

- **March addressing order (AO)** \( (^{\uparrow}, ^{\downarrow}) \) is implemented by utilizing generic ISA instructions, when feasible, similarly to data cache arrays. While the implementation of an ascending AO is straightforward, the implementation of a descending AO is challenging, because instructions are fetched only in ascending AO during normal system operation. When debug-diagnostic instructions are utilized, the high controllability and observability of these instructions allow a software routine to bypass the limitation of accessing cache lines in descending AO. A descending AO can be implemented by controlling the \( WS, SS, WLS \) and \( A \) fields to form a software loop. Moreover, when prefetch instructions are utilized and are combined with performance counters, a descending AO can be easily implemented for I-Tag by controlling the address \( A \) that defines the cache line to be accessed through a software loop.

The proposed algorithmic notation of an SBST routine for both I-Data and I-Tag arrays of an instruction cache is shown in Figure 4. The symbol notations remain the same as in Figure 3.
March SS for instruction cache

\[
\begin{align*}
\text{// set-associative : } & N_s = \# \text{ of sets} \\
\text{// direct mapped : } & N_s = \# \text{ of lines} \\
I & = DB_{DATA} \\
\bar{I} & = DB_{DATA} \\
\text{DC} & \quad // \text{Disable cache (or placed in non-cacheable area)} \\
\text{// M1 element } (R_0, R_{ns}, W_{in}, R_{ns}, W_{n+1}) \\
\text{for } (i=0; i<N_s; i=i+1) & \quad // \text{Access all sets/lines of cache} \\
A & = \text{create_address} (U_1: i : B) \quad // \text{Address to access test patterns} \\
\bar{A} & = \text{create_address} (U_2: i : B) \quad // \text{Address to access test patterns} \\
\text{DD read} (A, 1) & \quad // \text{EC ; Call (A) ; DC + Perf Counter} \\
\text{DD read} (A, 1) & \quad // \text{EC ; Call (A) ; DC + Perf Counter} \\
\text{DD write} (A, 1) & \quad // \text{Prefetch block (A)} \\
\text{DD read} (A, 1) & \quad // \text{EC ; Call (A) ; DC + Perf Counter} \\
\text{DD write} (\bar{A}, 1) & \quad // \text{Prefetch block (A)} \\
\end{align*}
\]

Figure 4: SBST routine for March SS test for instr. caches

When diagnostic-debug instructions are present in the ISA, this template is valid for both I-Tag and I-Data. U1 and U2 upper parts of address are any valid memory segments that are initialized with the test patterns that will be fetched to the arrays.

On the contrary, when prefetch and call instructions are utilized, the selection of U1 and U2 memory blocks is more challenging. In order to apply March write and read operations to the I-Data array, these two memory blocks are initialized to contain cache lines with valid instructions that comply with the chosen \((DB_{DATA}, DB_{DATA})\) pair. In this case, U1 and U2 can be any valid memory segment in the cacheable instruction segment. Each block should contain instructions that form the actual test pattern. One return instruction should be included in every cache line to facilitate the next March test iteration [19]. The verification of March read operations is performed by validating the execution result of the instruction sequence that forms the test patterns. If a \((DB_{DATA}, DB_{DATA})\) pair cannot be defined in an ISA, two or more pairs of instructions with convenient formats (partially complementary adjacent bits) can be utilized [12], [13]. In order to apply March writes and reads to the I-Tag array, U1 and U2 upper parts of address are the desired \((DB_{DATA}, DB_{DATA})\) pair and must be selected carefully as in the majority of processor architectures the virtual memory mechanism does not allow to map instruction segments to any memory segment during on-line operation. The content of these blocks is not of high importance. The sole requirement is that every cache line should contain one return instruction in order to achieve continuous address order.

In the proposed template an enable/disable cache mechanism is utilized. Instruction cache is only activated, through the enable cache (EC) operation, when the call instruction is utilized in order to activate the performance counter’s monitoring for the I-Tag.

5. Case Study

The SBST methodology has been applied to the L1 cache arrays of two different open source processor benchmarks to demonstrate its high adaptability. Both of them are RISC architectures but differ in their features and the supported ISA. Hence, the methodology has been applied in a unique way on each benchmark.

In order to evaluate the effectiveness of the self-test routines we have used RAMSES memory fault simulator [22]. RAMSES consists of a simulation engine and numerous fault descriptors. The simulation engine reads the test inputs and sets the operation flag for each memory cell. Fault coverage is determined by checking the fault descriptors for predefined conditions and considers fault injection in every cell. When coupling faults are concerned, the rest of the array cells except from the aggressor cell are possible victim cells. We have extended RAMSES fault simulator to include the fault models on the basis of FPs [1] by adding new fault descriptors and we have implemented a test framework to bridge captured cache traces of ModelSim simulator with RAMSES to fault grade the arrays of the caches.

5.1 LEON3

The first benchmark is LEON3, a publicly available processor designed by Aeroflex Gaisler and implements a SPARC V8 compliant architecture. We have configured the benchmark to include two 4KB 2-way set-associative L1 data and instruction caches with 64 sets and 8 words per cache line. This configuration includes two - one per way - 64x29 tag arrays (D-Tag and I-Tag) and two 512x32 data arrays (D-Data and I-Data).

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. In detail, alternate store \((sta)\) instructions have been used to implement March write operations and alternate load \((lda)\) instructions have been used to implement the March read operations. These instructions access all the cache SRAM arrays by utilizing the appropriate address indexing and the corresponding ASI. Note that when utilizing diagnostic accesses to a cache array, the cache should be invalidated afterwards. Hence, even in I-Data array, whose test patterns are formed by valid instructions, an SBST routine that takes advantage of alternate load instructions can apply any data background pair \((DB_{DATA}, DB_{DATA})\) with no limitation because the cache will be invalidated after the test.
An assembly code snippet for March element M1 of March SS test for way 0 of I-Data array is shown in Figure 5. Instruction \texttt{lda \{r3\} \text{0xd, r5} \text{fetches I-Data array contents of the address provided by \%r3 register (ASI 0xd is mapped to I-Data)} to the general purpose register \%r5. Instruction \texttt{sta \{r1\}, \{r3\} \text{0xd initializes a cache content line in a similar way with a predefined value (the desired DB). Moreover, read validations are performed in every read operation and the validation of the result is compacted in register \%r7. At the end of a successful test, the expected value of register \%r7 is zero. The same test with different address indexing targets way 1 of I-Data array. SBST routines for the rest of the cache arrays (D-Tag, D-Data, I-Tag) are formed in a similar way with different ASI values.

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 program statistics for both caches are shown in Table 2 and Table 3. The complexity of the March tests are expressed by their test lengths (n: number of bits of the array). The test routines are very effective in terms of test code size and test duration due to the utilization of the DCA instructions.

Finally, we evaluated test effectiveness of the test routines by utilizing RAMSES fault simulator. The port activity of cache arrays has been monitored and captured using ModelSim during the execution of SBST routines and then evaluated by RAMSES fault simulator for all unlinked static faults\textsuperscript{2} [1] to provide the achieved fault coverage. The coverage is complete (100\%) for the fault models that every March test guarantees for both LEON3’s data and instruction cache SRAM arrays.

\textsuperscript{2}Single Cell Faults: State (SF), Transition (TF), Write Destructive (WDF), Read Destructive (RDF), Deceptive Read Destructive (DRDF), Incorrect Read (IRF)

\textbf{Cell Coupling Faults (2-cell):} State (CFst), Disturb (CFds), Transition (CFtr), Write Destructive (CFwd), Read Destructive (CFrd), Deceptive Read Destructive (CFdr), Incorrect Read (CFir)

5.2 OpenRISC 1200

The second benchmark is OpenRISC 1200, a publicly available processor core. The processor has been parameterized to utilize 4KB direct mapped write-through L1 caches that include 256x20 tag arrays (D-Tag & I-Tag) and 1024x32 data arrays (D-Data array & I-Data array). OpenRISC 1200 has been extended to include programmable performance counters that monitor data and instruction cache misses based on the specification of the designer’s manual.

OpenRISC 1200 lacks diagnostic-debug instructions in its ISA to access the cache arrays. However, it includes a cache prefetch mechanism for both L1 caches and maps prefetch operations to valid instructions. These instructions have been used 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 cache prefetch operation in OpenRISC 1200 is implemented through a special purpose register (DCBPR register for data cache and ICBPR register for instruction cache). Indirect access to cache array has been used to implement March read operations by utilizing the load (l.lwz) instruction for the data cache and the jump and link register (l.jalr) instruction for the instruction cache. An enable/disable cache mechanism has been utilized since OpenRISC 1200 lacks the ability to define a non-cacheable area. In D-Data array, read validation has been performed by comparing the \texttt{l.lwz} instruction result with a golden value in every March test iteration. In I-Data array, two instructions with complementary opcodes (a register vector addition \texttt{lv.adds.h} and the immediate store \texttt{l.sw}) have been used to form the test patterns. At the end of every cache line we have placed a jump register (\texttt{l.jr}) instruction. The March read operations have been validated by elaborating the result of both \texttt{lv.adds.h} and \texttt{l.sw} execution. We have executed twice the same routines for I-Data array. The first execution with the \texttt{l.jr} instruction located at the last word of every cache line and the second one with the \texttt{l.jr}
instruction located to another word inside the cache line to detect the remaining faults that are masked by the utilization of a jump instruction [19]. Cache misses monitored by performance counters for D-Tag and I-Tag arrays verify March read operations at the end of the test. An assembly code snippet for March SS M1 element for D-Tag array is shown in Figure 6.

Figure 6: Code snippet for OpenRISC 1200 D-Tag array

We have implemented the same March tests that we presented for LEON3. The test program statistics for both caches are shown in Table 4 and Table 5. As shown in tables, test programs, are cost-effective both in terms of code size and test duration. Routines for instruction cache have longer test duration as they have been executed with the instruction cache disabled. Moreover, test duration is even longer for the I-Data array because of code size and test duration. Routines for instruction cache enabled to activate the performance counter in tables, test programs, are cost-effective both in terms of test size and duration. Table 4 presents for LEON3. The test program statistics for March tests that we implemented for OpenRISC 1200 D-Cache: SBST routines statistics.

### Table 4: OpenRISC 1200 D-Cache: SBST routines statistics

<table>
<thead>
<tr>
<th>March test</th>
<th>Complexity (n)</th>
<th>Test size (bytes)</th>
<th>Test Duration (cycles)</th>
<th>Test size (bytes)</th>
<th>Test Duration (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>March C-</td>
<td>10</td>
<td>280</td>
<td>16,667</td>
<td>1,016</td>
<td>58,217</td>
</tr>
<tr>
<td>March U</td>
<td>13</td>
<td>288</td>
<td>18,000</td>
<td>1,436</td>
<td>81,060</td>
</tr>
<tr>
<td>March MSS</td>
<td>18</td>
<td>344</td>
<td>22,666</td>
<td>1,496</td>
<td>95,023</td>
</tr>
<tr>
<td>March SS</td>
<td>22</td>
<td>384</td>
<td>27,000</td>
<td>2,200</td>
<td>147,129</td>
</tr>
</tbody>
</table>

### Table 5: OpenRISC 1200 I-Cache: SBST routines statistics

<table>
<thead>
<tr>
<th>March test</th>
<th>Complexity (n)</th>
<th>Test size (bytes)</th>
<th>Test Duration (cycles)</th>
<th>Test size (bytes)</th>
<th>Test Duration (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>March C-</td>
<td>10</td>
<td>744</td>
<td>658,334</td>
<td>800</td>
<td>1,385,836</td>
</tr>
<tr>
<td>March U</td>
<td>13</td>
<td>780</td>
<td>779,666</td>
<td>832</td>
<td>1,411,222</td>
</tr>
<tr>
<td>March MSS</td>
<td>18</td>
<td>893</td>
<td>970,420</td>
<td>1,120</td>
<td>2,540,104</td>
</tr>
<tr>
<td>March SS</td>
<td>22</td>
<td>1,128</td>
<td>1,180,400</td>
<td>1,296</td>
<td>3,185,278</td>
</tr>
</tbody>
</table>

### 6. Comparisons

In this section, we compare our methodology against the previous ones that provide pseudo-instruction sequences ([12] and [15]) and experimental results of the March operations ([13], [15] and [19]) for cache arrays. In [12] and [15] four pseudo-instructions (a cache disable, a main memory write, a cache enable and a load miss and refill) were used to apply the March write operations for the tag arrays due to the need of setting up data inconsistency. A similar pseudo-instruction sequence is utilized for March read operation. In the proposed methodology, March write and read operations can be performed in a low cost way with a single pseudo instruction by utilizing the DCA instructions.

Moreover in [12] and [13], pairs of instructions that complement partially in some bits are utilized to overcome the test pattern composition challenge of I-Data array. The whole algorithm had to be repeated multiple times with multiple sets to cover all instruction bits. In [19], a more effective approach was utilized that fills the cache lines with instructions that have complementary bits but still the whole test has to be repeated at least two times to cover all coupling faults. Both solutions have a serious effect on test duration. In this paper, the utilization of DCA instructions overcomes this difficulty. Any DB can be utilized as the cache will be invalidated at the end of the test.

Finally, in [12] and [19], two different solutions were proposed to implement a descending AO in instruction cache. In [12], the test execution is performed in processor’s trace mode, while in [19] a reordering function that includes a set of read access, modifies the access history of the cache lines. Indeed, both solutions overcome the abovementioned difficulty but have a significant effect in test duration. The proposed methodology overcomes the challenge to implement a March descending AO in instruction cache, by controlling the WS, SS and LWS fields of DCA instructions to setup a cache access descending AO.
Table 6: Statistics comparison for D-Cache (March C-)

The abovementioned reasoning is also verified by comparing the statistical results with the different SBST approaches. In Table 6, benchmarks with similar ISA for a given March test (March C-) are compared. The experimental results in [15] for the data cache of an ARM-compatible processor are compared with our experimental results for the data cache of the LEON3 processor. Both caches have similar size (each of LEON’s cache ways has similar size to the cache of [15]). Even though a direct comparison is not feasible between different benchmarks, the methodology favors the one that is presented in [15] as it can test similar tag arrays in much less test time for the same March test.

Furthermore, in Table 7 a direct comparison on the same benchmark between the methodology of [19] and our methodology is presented. Both methodologies apply the same March test (March SS) to the instruction cache of LEON3 processor. Note that the proposed methodology significantly improves both test code size and test duration of March SS application to LEON3’s instruction cache. The total test code size for both arrays (I-Tag & I-Data) is improved by 83%, while the total test duration is improved by 72%. This improvement is achieved by exploiting DCA instructions to effectively implement the March test operations.

Table 7: Statistics comparison for I-Cache (March SS)

8. Conclusion

We presented an SBST program development methodology for on-line testing of L1 cache arrays based on March test algorithms. Our methodology overcomes testability challenges of L1 cache arrays by leveraging existing special purpose instructions that modern ISAs implement to access caches for debug-diagnostic and performance purposes, as well as, performance monitoring mechanisms. Experimental results on two processor benchmarks, OpenRISC 1200 and LEON3, demonstrate the effectiveness of the presented methodology in terms of test duration and test code size. Experimental results for LEON3, when compared with other SBST approaches, show a significant improvement in test duration (72%) and test code size (83%).

References