POLITECNICO DI TORINO
Repository ISTITUZIONALE

Applying March Tests to K-Way Set-Associative Cache Memories

Original
Applying March Tests to K-Way Set-Associative Cache Memories / Alpe, S., DI CARLO, S., Prinetto, P.E., Savino, A.. - STAMPA. - (2008), pp. 77-83. (IEEE 13th European Test Symposium (ETS) Verbania, IT 25-29 May 2008) [10.1109/ETS.2008.25].

Availability:
This version is available at: 11583/1845184 since:

Publisher:
IEEE Computer Society

Published
DOI:10.1109/ETS.2008.25

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

30 October 2026

Applying March Tests to K-Way Set-Associative Cache Memories

Simone Alpe, Stefano Di Carlo, Paolo Prinetto, Alessandro Savino
Politecnico di Torino, Dep. of Control and Computer Engineering
Torino, Italy
e-mail: {simone.alpe, stefano.dicarlo, paolo.prinetto, alessandro.savino}@polito.it

Abstract

Embedded microprocessor cache memories suffer from limited observability and controllability creating problems during in-system test. The application of test algorithms for SRAM memories to cache memories thus requires opportune transformations.

In this paper we present a procedure to adapt traditional march tests to testing the data and the directory array of k-way set-associative cache memories with LRU replacement. The basic idea is to translate each march test operation into an equivalent sequence of cache operations able to reproduce the desired marching sequence into the data and the directory array of the cache.

1. Introduction

Today's modern computer system performances mainly relies on a large capacity memory subsystem including up to four level of caches [1]. Cache memories are small, high-speed buffers used to temporarily hold information (e.g., portions of the main memory) which are (believed to be) currently in use. As a result, larger and larger portions of die area are occupied by cache memories. For instance, 50% of Pentium 4 chip area and 60% of StrongARM chip area are allocated to cache structures [2]. These considerations lead to the conclusion that efficient test procedures for cache memories are essential to guarantee the quality of modern computer systems.

Only few publications addressed the problem of cache testing. In [2] the authors propose a very exhaustive study of new functional fault models for particular cache technologies (drowsy SRAM caches) and propose a new march test (March DWOM) to cover them. The main drawback of this algorithm is that it reduces the problem of cache testing to the problem of testing generic word-oriented memories. It thus requires a specific design to allow cell-by-cell addressing of the cache.

Some cache testing approaches have been proposed as part of microprocessor software-based self-testing methodologies. They consider caches embedded into microprocessors and face the problem of their limited accessibility. [3] presents a random approach for testing the PowerPC microprocessor cache whereas [4] introduces a systematic approach to on-chip cache testing as part of the memory subsystem of a microprocessor. The main drawback is that the cache testing is only outlined, and a single test algorithm based on the well known March B [5] is presented.

[6] generalizes the methodology introduced in [4] proposing a *march like* test procedure taking into account whether the cache is used to store data or instructions. The main drawback of the paper is that a clear explanation of how to extend the proposed test to new fault models is missing, and moreover the application of the test requires particular hardware features that may, in turn, require hardware modifications in the cache.

To face the problem of new memory fault models, [7] proposes a methodology to translate generic march tests into test sequences for the directory array of cache memories. The procedure allows preserving the same fault detection as the original test but it is limited to direct-mapped caches with write-back policy. Even if the authors suggest that the same procedure is easily adaptable to set-associative caches, how this is possible is not detailed and moreover write-through caches are not considered.

In this paper we propose a methodology to translate generic march tests into equivalent versions for in-system testing of both directory and data array of set-associative caches with write-back or write-through policy. Among the different types of replacement algorithms for set-associative caches we focus on memories implementing the Last Recently Used (LRU) replacement. The main goal is to propose a translation

1530-1877/08 $25.00 © 2008 IEEE
DOI 10.1109/ETS.2008.25

IEEE computer society

methodology providing tests that preserve both the original fault coverage and (wherever possible) the complexity of the original march test.

The paper is organized as follows: Section 2 overviews basic notions about cache memories whereas Section 3 proposes the translation procedure. Section 4 applies the proposed methodology to a real march test and proposes some considerations about the complexity of the translated tests, and finally Section 5 summarizes the main contributions of the paper and outlines future works.

2. Cache memories overview

The purpose of cache memories is to speed up the microprocessor memory access by storing recently used data. During any processor reference to memory the cache checks whether it already stores the requested location (cache hit) or no (cache miss).

The internal organization usually comprises two memory arrays: the *data array* and the *directory array* (or *tags array*). The data array holds the actual data to store in the cache. The size of the data array determines the size of the cache. Rather than reading a single word or byte from main memory at a time, each cache entry usually holds a certain number of words, known as a *cache line* or *way*. The address space is thus divided into blocks to be mapped to the different cache lines. The directory array is a small and fast memory that stores portions (*tags*) of the main memory addresses related to data stored in the data array.

The way data are cached depends on the cache mapping schema. We consider in this paper two mapping schema: *direct-mapped* and *set-associative* caches.

In a direct-mapped cache (see Figure 1) the cache location for a given N bits memory address is determined as follows: if the cache line size is 2^O memory words then the bottom O address bits correspond to an *offset* within the cache line. If the cache holds 2^I lines then the next I address bits, called *index*, give the cache location (line). The remaining top $N-I-O$ bits represent the *tag* to store in the directory array along with the entry. In this schema, there is no choice of which cache line to replace on a cache miss since a given memory block b can only be stored in the cache line with index equal to $b \mod 2^I$.

In a set-associative cache (see Figure 2) each N bits memory address is mapped to a certain *set* of cache lines. Again, the bottom O bits represent an offset in the cache line. In this case the index (middle I bits of the address) identifies the set where the address is located. In a *k-ways set-associative* cache with 2^I sets, each set has k cache lines. In this case the cache lines

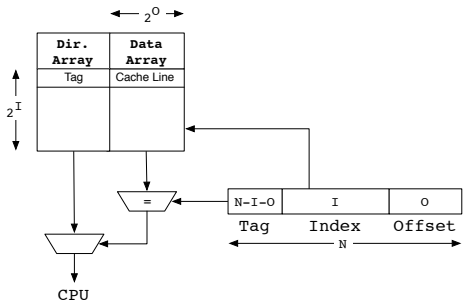


Figure 1. Direct-mapped architecture

are also referred to as *ways*. A memory block b is mapped to the set with index $b \mod 2^I$ and may be stored in any of the k cache lines in that set with its upper $T = N - I - O$ address bits used as tag. To determine whether the block b is in the cache or not, the set $b \mod 2^I$ is searched associatively for the tag. In the sequel of the paper, in order to select the line to be replaced in case of cache miss, we shall adopt the *Least Recently Used* (LRU) replacement algorithm, which discards the least recently used lines first. A direct-mapped cache is a special instance of a set-associative cache with a single way for each set.

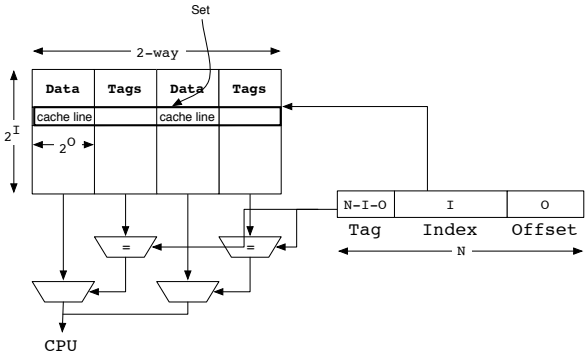


Figure 2. 2-ways set-associative architecture

To conclude cache memories can also be classified based on their write policy into two categories:

- *Write-Back*: when the system writes to a memory location that is currently held in cache, it only writes the new information to the appropriate cache line. When the cache line is eventually needed for some other memory address, the changed data is *written back* to the system memory. This type of cache provides better performance than a write-through cache, because it saves on (time-consuming) write cycles to memory;
- *Write-Through*: when the system writes to a memory location that is currently held in cache, it

writes the new information both to the appropriate cache line and the memory location itself at the same time. This type of caching provides worse performance than write-back, but is simpler to implement and has the advantage of internal consistency, because the cache is never out of sync with the memory the way it is with a write-back cache.

3. March test translation

This section introduces the actual march test translation procedure.

As introduced in Section 2, cache memories are composed of two different arrays: the data array and the directory array. March tests detect faults by performing write and read and verify operations on the memory under test [5]. While this approach is easily applicable with only few modifications to the data array, the situation is different for the directory array, since it is not directly accessible by the user. For this reason, we will provide two different translation procedures. A single march test will be thus translated into two different tests, targeting the data array and the directory array, respectively.

Before illustrating the translation procedure, we need to introduce some definitions and notations to extend traditional march tests. First of all we have to remember that we consider a k-way set-associative cache with LRU replacement. In this situation an N bits address is split into three fields (see Section 2):

- *TAG*: composed of T bits;
- *INDEX*: composed of I bits;
- *OFFSET*: composed of O bits.

We will have $S = 2^I$ sets in the cache, with k cache lines in each set.

With this assumptions, we can define the following basic operations:

- $w(\alpha t, DB)$: represents a *write operation* of a cache line. DB is a data background pattern [8] to write in the data array of the cache, i.e., a generic sequence of O bits with the only constraint that for each DB a complemented pattern $\overline{DB}$, obtained from DB by complementing its bits, must be defined. t represents the tag to write in the directory array of the cache, and α is the index that identifies the set of the cache were the line has to be written;
- $r(\alpha t, DB)$: represents a *read and verify operation*. The cache line belonging to the set with index α and associated with the tag t is read and compared with the expected data background pattern DB;
- $r(\alpha t)$: represents a simple read operation, it behaves as the read and verify operation but the result of the read is not verified. This operation will be used in Section 3.2 to introduce the so called re-ordering operations.

Moreover we define two new addressing orders called *way-in-index addressing orders* :

- $\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k}$: it identifies an ascending addressing order in terms of sets (the first arrow) and ways (cache lines) composing each set (the second arrow);
- $\downarrow_{\alpha=0}^{S-1}\downarrow_{i=1}^{k}$: it identifies a descending addressing order in terms of sets and ways as above.

Using this new notation the next subsections will overview the march test translation procedure for the data and the directory array, respectively.

3.1. Data Array

By using the previously introduced extended march test notation, translating a traditional march test into an equivalent one for the data array of a memory cache is straightforward. Table 1 reports a one to one correspondence between the traditional march test notation and the cache memory notation for the data array.

Table 1. Data array translation table

Traditional notation	Cache notation
$w1$	$w(\alpha t_i, DB)$
$w0$	$w(\alpha t_i, \overline{DB})$
$r1$	$r(\alpha t_i, DB)$
$r0$	$r(\alpha t_i, \overline{DB})$
$\Uparrow$	$\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k}$
$\Downarrow$	$\downarrow_{\alpha=0}^{S-1}\downarrow_{i=1}^{k}$

The only consideration about this translation schema concerns the definition of the tags to use during the test operations. The translation of the traditional addressing orders into the way-in-index addressing orders implies the ability of addressing by a given sequence the cache lines belonging to a single set (second arrow of the symbol). According to the LRU replacement algorithm, starting with an empty cache, and given k cache lines belonging to the same set, k consecutive write operations will be performed on different lines only if they contain k different tags. After k operations the set will be full, the oldest line will be replaced and the same addressing sequence will be reproduced again. A set of k different tags is thus required for a k-way set-associative cache in order to address for

each set all the cache lines. The notation t_i in Table 1 refers to the i^{th} tag in this set.

Table 2 shows an example of tags for a 4-way set-associative cache. The actual value of the tags is not relevant. In this test, the tags represent an indirect addressing mechanism that allows moving among the lines of a given set, but the actual addressing mechanism is managed by the replacement algorithm (LRU in our case). It allocates cache lines based on the diversity of the tags and not on their absolute value. Obviously faults can influence this mechanism but since tags are contained in the directory array they will be addressed in Section 3.2 dealing with the directory array. In this context, the only requirement is the number of different tags.

Table 2. Example of (data) tags for a 4-way set associative cache

Tag	Bit representation (T-bits)
t_0	$0\ldots000$
t_1	$0\ldots001$
t_2	$0\ldots010$
t_3	$0\ldots011$

Using this translation schema, Table 3 shows an example of two simple march elements translation.

Table 3. Data array translation example

$\Uparrow \{r1, w0, \ldots\} \longmapsto \uparrow_{\alpha=0}^{S-1} \uparrow_{i=1}^{k} \{r(\alpha t_i, DB), w(\alpha t_i, \overline{DB}), \ldots\}$
$\Downarrow \{r1, w0, \ldots\} \longmapsto \downarrow_{\alpha=0}^{S-1} \downarrow_{i=1}^{k} \{r(\alpha t_i, DB), w(\alpha t_i, \overline{DB}), \ldots\}$

This translation schema allows translating any type of march test and, the introduction of the way-in-index addressing orders allows to apply the march test to the data array in such a way that both intra-set and extra-set coupling faults can be considered.

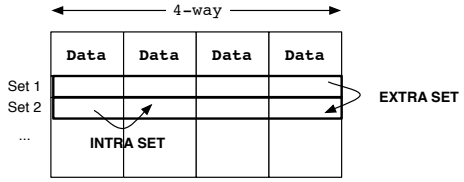


Figure 3. Intra and extra set coupling faults

Moreover, by translating march tests developed for word-oriented memories [8], it is possible to detect intra-word faults in the single cache lines.

3.2. Directory Array

The first important thing to underline when dealing with the march test translation for the directory array is that both write and read operations can be only performed in a *indirect* way working on the data array. In this situation the actual test patterns are represented by tags. When applying a march test to the directory array, tags need to change properly in order to generate the desired marching sequence.

Table 4 introduces the translation rules in the case of directory array testing where the symbol '$-$' denotes that the value written in the cache line is not relevant.

Table 4. Directory array translation table

Traditional notation	Cache notation
$w1$	$w(\alpha t_i, -)$
$w0$	$w(\alpha \overline{t_i}, -)$
$r1$	$r(\alpha t_i, -)$
$r0$	$r(\alpha \overline{t_i}, -)$
$\Uparrow$	$\uparrow_{\alpha=0}^{S-1} \uparrow_{i=1}^{k}$
$\Downarrow$	$\downarrow_{\alpha=0}^{S-1} \downarrow_{i=1}^{k}$

As already highlighted for the data array, when translating an addressing order into the corresponding way-in-index addressing order we need to address with a given sequence the tags belonging to a single set. Again this is possible by using a number of different tags equal to the number of cache lines contained in a single set (LRU replacement). Moreover, in this case, since the tag represents the actual test pattern, according to the march test theory we need to be able to generate both a test pattern and a complemented test patterns. For each tag we thus need to define a complemented tag. Table 5 shows an example of tags for a 4-ways set-associative cache.

Table 5. Example of Tags for a 4-way set associative cache

Tag	Bit representation (T-bits)	Tag	Bit representation (T-bits)
t_0	$1\ldots111$	$\overline{t_0}$	$0\ldots000$
t_1	$1\ldots110$	$\overline{t_1}$	$0\ldots001$
t_2	$1\ldots101$	$\overline{t_2}$	$0\ldots010$
t_3	$1\ldots100$	$\overline{t_3}$	$0\ldots011$

The proposed approach still presents some problems that need to be analyzed and solved.

The first issue concern the execution of a read and verify operation. When working with the directory array there is no way to read and verify the value of the stored tags. Faults into the directory array lead to a set of tags different from the expected one. Any tentative of reading the content of lines associated to a missing tag will generate unexpected cache misses that have to

be interpreted as a detected fault. The actual problem is how to identify a cache miss.

In general, in order to identify a cache miss we have to be able to generate a discrepancy between the content of a cache line and the content of the corresponding main memory location. In this situation, in case of miss, the value read from the main memory is different from the one expected in the cache and the miss can be easily identified. Depending on the write policy of the cache this condition can be obtained in two different ways.

In a write-through cache, the cache content is always consistent with the main memory content. This is the worst situation. The only way to have different values between the cache and the main memory is to temporarily disable the cache to explicitly write a given value in the main memory. Most of the modern microprocessors (e.g. Pentium, PowerPC) allow this operation using particular instructions.

In this situation the translation rules proposed in Table 4 have to be modified according to Table 6 where the notation $\lceil \cdot \rceil$ represents an operation performed with the cache disabled and the data background patterns (DB) are used to identify a cache miss.

Table 6. Write-Through Translation

Traditional notation	Cache notation
$w1$	$w(\alpha t_i, DB)\lceil w(\alpha t_i, \overline{DB})\rceil$
$w0$	$w(\alpha \overline{t_i}, \overline{DB})\lceil w(\alpha \overline{t_i}, DB)\rceil$
$r1$	$r(\alpha t_i, DB)$
$r0$	$r(\alpha \overline{t_i}, \overline{DB})$

In the case of write-back policy the cache already comprises the possibility of having an inconsistency between the cache and the main memory content. This inconsistency can be generated by the following modifications to the translation rules proposed in Table 4:

- Each write operation must include a data background pattern complemented with respect to the one used during the previous write operation of the same type (i.e $w0$ or $w1$) in the original march test;
- Each read and verify operation must include the expected data background pattern provided by the cache;
- An initialization march element has to be included: it uses a data background pattern complemented with respect to the one used by the first write operation of the march test.

In addition to these two solutions, some microprocessors provide hardware features to measure the number of cache misses both in the data and in the instruction cache. For example the Intel Pentium Family provides a set of performance monitoring facilities accessible through the $RDMSR$ instruction that include the possibility of counting the number of cache misses during the execution of an instruction or a sequence of instructions [9]. Obviously this solution is applicable both to write-back and write-through caches.

The proposed translation schema still has two major issues related to the LRU replacement mechanism that have to be addressed:

- It is not possible to directly perform two write operations (or a read operation followed by a write operation) with different tags on the same cache line;
- It is not directly possible to address cache lines belonging to a set in a reverse addressing order.

A simple solution to this problem consists in introducing a set of additional read operations in order to artificially modify the access time of the cache lines belonging to a set and to produce the desired addressing sequence.

We introduce an additional march test operation called *re-ordering* defined as:

$$RO(\alpha P_t) = \{r(\alpha t_k), \forall t_k \in P_t\} \quad (1)$$

where P_t is the set of tags associated with the cache lines of the set α with access time older than the cache line containing the tag t. This operation generates a sequence of reads (without verification) that access the cache lines used before the one containing the tag t. In this way the access time of all these lines is set properly in order to make sure the next operation will be performed on the cache line associated with t.

The re-ordering operation has to be included in general before each write operation of the march test with the following two exceptions:

- If the first march element starts with a write operation and has an ascending addressing order, the write operation does not require the re-ordering;
- If the march element begins with a write operation and its addressing order is equal to the addressing order of the previous march element, the first write operation does not require the re-ordering.

This solution obviously introduces a certain overhead in the final march test complexity. The number of additional read operations is proportional to the number of lines contained in a set which is usually lower then the number of sets of a cache. A possible way to keep this overhead as low as possible is to start wherever possible from a Single Order Addressing (SOA) march test [10] in order to avoid the re-ordering

sequence at the beginning of the march elements starting with write operations.

4. Experimental results

To show the applicability of the proposed methodology we applied the translation technique to the SOA March C– proposed in [10] considering a generic k-way set-associative cache. We will show the results of both the data and the directory array translation.

Figure 4 shows the original march test. Its complexity is $10N$, where N is the number of memory cells/words composing the memory under test.

$$\underset{M_0}{\Uparrow (w1)}; \underset{M_1}{\Uparrow (r1, w0, w1)}; \underset{M_2}{\Uparrow (r1, w0)}; \underset{M_3}{\Uparrow (r0, w1, w0)}; \underset{M_4}{\Uparrow (r0)}$$

Figure 4. SOA March C–

According to the translation rules proposed in Section 3.1, the corresponding data array march test is reported in Figure 5.

$$\begin{aligned}
&\underset{M_0}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} w(\alpha t_i, DB)};\\
&\underset{M_1}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha t_i, DB), w(\alpha t_i, \overline{DB}), w(\alpha t_i, DB)};\\
&\underset{M_2}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha t_i, DB), w(\alpha t_i, \overline{DB})};\\
&\underset{M_3}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha t_i, \overline{DB}), w(\alpha t_i, DB), w(\alpha t_i, \overline{DB})};\\
&\underset{M_4}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha t_i, \overline{DB})}
\end{aligned}$$

Figure 5. Data Array March C– for a k-way set-associative cache

The complexity of the test is expressed by eq. 2 where S is the number of sets of the cache, k is the number of ways (i.e., cache lines per set), $\#ME$ is the number of march elements composing the march test and $\#OP_{ME_j}$ is the number of operations composing the j^{th} march element (in the original march test):

$$\begin{aligned}
C_{data} &= \sum_{j=1}^{\#ME} \left[\#OP_{ME_j} \cdot k \cdot S\right] \\
&= \underset{M_0}{kS} + \underset{M_1}{3kS} + \underset{M_2}{2kS} + \underset{M_3}{3kS} + \underset{M_4}{kS} = 10kS
\end{aligned} \tag{2}$$

Looking at eq. 2 it is possible to note that the complexity of the new generated march test, in terms of memory operations, is equivalent to the original one. In fact a k-way set-associative cache with S sets is equivalent to a conventional memory with $N = k \cdot S$ memory words. In our example we started from a $10N$ march test and after the translation process we obtained a $10kS$ march test equivalent in terms of complexity.

The directory array translation is more complex, due to the introduction of the re-ordering sequences and to the different translation rules depending on the write policy of the cache. Figure 6 and Figure 7 propose the March C– translated to test the directory array of a generic k-way set-associative cache with write-trough and write-back write policy respectively.

$$\begin{aligned}
&\underset{M_0}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} w(\alpha t_i, DB)\lceil w(\alpha t_i, \overline{DB})\rceil};\\
&\underset{M_1}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha t_i, DB)\mathbf{RO}(\mathbf{P_{\bar{t}_i}}), w(\alpha \bar{t_i}, \overline{DB})\lceil w(\alpha \bar{t_i}, DB)\rceil}\\
&\qquad \mathbf{RO}(\mathbf{P_{t_i}}), w(\alpha t_i, DB)\lceil w(\alpha t_i, \overline{DB})\rceil;\\
&\underset{M_2}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha t_i, DB), \mathbf{RO}(\mathbf{P_{\bar{t}_i}}), w(\alpha \bar{t_i}, \overline{DB})\lceil w(\alpha \bar{t_i}, DB)\rceil};\\
&\underset{M_3}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha \bar{t_i}, \overline{DB})\mathbf{RO}(\mathbf{P_{t_i}}), w(\alpha t_i, DB)\lceil w(\alpha t_i, \overline{DB})\rceil},\\
&\qquad \mathbf{RO}(\mathbf{P_{\bar{t}_i}}), w(\alpha \bar{t_i}, \overline{DB})\lceil w(\alpha \bar{t_i}, DB)\rceil);\\
&\underset{M_4}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha \bar{t_i}, \overline{DB})}
\end{aligned}$$

Figure 6. Directory Array March C– for a write-through k-way set-associative cache

$$\begin{aligned}
&\underset{M_{-1}}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} w(\alpha t_i, \overline{DB}), w(\alpha \bar{t_i}, \overline{DB})};\\
&\underset{M_0}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} w(\alpha t_i, DB)};\\
&\underset{M_1}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha t_i, DB)\mathbf{RO}(\mathbf{P_{\bar{t}_i}}), w(\alpha \bar{t_i}, DB),}\\
&\qquad \mathbf{RO}(\mathbf{P_{t_i}}), w(\alpha t_i, \overline{DB});\\
&\underset{M_2}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha t_i, \overline{DB}), \mathbf{RO}(\mathbf{P_{\bar{t}_i}}), w(\alpha \bar{t_i}, \overline{DB})};\\
&\underset{M_3}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha \bar{t_i}, \overline{DB})\mathbf{RO}(\mathbf{P_{t_i}}), w(\alpha t_i, DB),}\\
&\qquad \mathbf{RO}(\mathbf{P_{\bar{t}_i}}), w(\alpha \bar{t_i}, DB);\\
&\underset{M_4}{\uparrow_{\alpha=0}^{S-1}\uparrow_{i=1}^{k} r(\alpha \bar{t_i}, DB)}
\end{aligned}$$

Figure 7. Directory Array March C– for a write-back k-way set-associative cache

Using the same formalism used for the data array we can compute the complexity of the directory array march test according to eq. 3 where $\#Write_{ME_j}$ denotes the number of write operations contained in the j^{th} march element of the original march test. This term has to be included only in case of write-through cache in order to consider the extra write operations performed in main memory.

$$C_{dir} = \sum_{j=1}^{\#ME} \left[\left(\#OP_{ME_j} \overbrace{+\#Write_{ME_j}}^{write-through} \right) \cdot k \cdot S + C_{RO_j} \right] + \overbrace{2 \cdot k \cdot S}^{write-back} \tag{3}$$

C_{RO_j} is the complexity introduced by the re-ordering sequences in the j^{th} march element and can

be computed according to eq. 4 where AO_j represents the addressing order of the j^{th} march element in the original test.

$$C_{RO_j} = \begin{cases} (\#OP_{ME_j} - 1)S \sum_{i=1}^{k-1} i & \text{if } AO_j = \Uparrow \text{ and } j = 1 \\ & \text{or} \\ & AO_j = AO_{j-1} \\ \#OP_{ME_j} S \sum_{i=1}^{k-1} i & \text{if } AO_j = \Downarrow \text{ and } j = 1 \\ & \text{or} \\ & AO_j \neq AO_{j-1} \end{cases} \quad (4)$$

By applying this calculation to the march tests of Figure 6 and Figure 7 we obtain the complexities reported by eq. 5.

$$\begin{aligned} C_{dir_{write-trough}} &= (10+6)kS + 5S \sum_{i=1}^{k-1} i \\ C_{dir_{write-back}} &= 2kS + (10)kS + 5S \sum_{i=1}^{k-1} i \end{aligned} \quad (5)$$

Looking at eq. 5, in both cases we have an increased complexity due to the extra reads introduced by the re-ordering sequences. This contribution increases with the increasing of the number of ways of the cache (in case of a direct-mapped cache it is equal to zero). Moreover, in the write-trough case we have an additional complexity of $6kS$ due to the additional write operations performed directly in main memory (this term is proportional to the number of writes contained in the original march test) whereas for the write-back cache we have a fixed overhead of $2kS$ due to the additional initialization sequence. In both cases this overhead can be reduced to zero if the system under test provides an hardware facility to detect cache misses (see Section 3).

To conclude our experiments we need to introduce some considerations about the coverage of the new tests. By construction the new generated march tests replicate exactly the same marching sequences of the original march test thus they guarantee to maintain the same coverage. There is only one exception to this rule represented by the introduction of extra read operations due to the re-ordering sequences. The introduction of additional read operations can reduce the overall coverage when dealing with complex dynamic faults.

5. Conclusion

This paper presented a procedure to translate traditional march tests developed for single chip memories into equivalent versions to target both the data and the directory array of a generic k-way set-associative cache with LRU replacement algorithm. The proposed translation technique allows to maintain, wherever possible, the same coverage and tries to minimize the overhead in term of complexity. Moreover it can be applied to cache memories with both write-back and write-through policy.

Future works will include the consideration of new fault models specifically designed for cache memory and the extension of our methodology to caches with different replacement algorithm (e.g. pseudo-random, pseudo LRU, etc.).

References

[1] R. Stacpoole and T. Jamil, "Cache memories," *Potentials, IEEE*, vol. 19, no. 2, pp. 24–29, Apr/May 2000.

[2] W. Pei, W.-B. Jone, and Y. Hu, "Fault modeling and detection for drowsy sram caches," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 6, pp. 1084–1100, June 2007.

[3] R. Raina, R.; Molyneaux, "Random self-test method applications on powerpc™microprocessor cachestm microprocessor caches," in *VLSI, 1998. Proceedings of the 8th Great Lakes Symposium on*, 19-21 Feb 1998, pp. 222–229.

[4] T. Verhallen and A. van de Goor, "Functional testing of modern microprocessors," in *Design Automation, 1992. Proceedings. [3rd] European Conference on*, 16-19 Mar 1992, pp. 350–354.

[5] A. Van De Goor, "Using march tests to test srams," *Design & Test of Computers, IEEE*, vol. 10, no. 1, pp. 8–14, Mar 1993.

[6] J. Sosnowski, "In-system testing of cache memories," in *Test Conference, 1995. Proceedings., International*, 21-25 Oct 1995, pp. 384–393.

[7] S. Al-Harbi and S. Gupta, "A methodology for transforming memory tests for in-system testing of direct mapped cache tags," in *Proceedings. 16th IEEE VLSI Test Symposium, 1998.*, 26-30 Apr 1998, pp. 394–400.

[8] A. Van De Goor, I. Tlili, and S. Hamdioui, "Converting march tests for bit-oriented memories into tests for word-oriented memories," in *Memory Technology, Design and Testing, 1998. Proceedings. International Workshop on*, 24-25 Aug 1998, pp. 46–52.

[9] J. Bhandarkar, D.; Ding, "Performance characterization of the pentium pro processor," in *Third International Symposium on High-Performance Computer Architecture, 1997*, 1-5 Feb 1997, pp. 288–297.

[10] A. van de Goor and Y. Zorian, "Effective march algorithms for testing single-order addressed memories," in *Design Automation, 1993, with the European Event in ASIC Design. Proceedings. [4th] European Conference on*, 22-25 Feb 1993, pp. 499–505.