

A Methodology for Transforming Memory Tests for In-System Testing of Direct Mapped Cache Tags*

Sultan M. Al-Harbi**
Department of EE-Systems
University of Southern California
Los Angeles, CA, 90089-2562

Sandeep K. Gupta
Department of EE-Systems
University of Southern California
Los Angeles, CA, 90089-2562

Abstract

While any efficient test developed for off-line testing of memory chips can be easily adapted for in-system testing of single level memory systems, no efficient methodology is known to transform such a test for in-system testing of multilevel memory systems that have one or more levels of cache. The main challenge is in transforming the known test to test the tags of the cache (testing of the data part of the cache is relatively straightforward). In this paper we present a general methodology to transform march tests for in-system testing of tags of direct mapped caches. The transformation has been used to obtain new versions of March B and March X tests. It is shown that the new versions of tests detect the same sets of faults in the cache tags as their original versions detect in memory chips. Finally, it is demonstrated that the proposed version of March B has significantly lower time complexity than previously proposed tests and can be applied without any modification of the memory system hardware.

1. Introduction

Today's typical system delivers performance partly by relying on a large capacity memory subsystem which includes one or more levels of caches. Since the fault-free operation of the memory sub-system is crucial to the correct operation of the system, periodic testing, say, during each system boot, of the memory subsystem is considered essential. While any of the large number of tests developed for off-line testing of memory chips [1] can be easily adapted for in-system testing of main memory as well as data part of caches, considerable modifications are required before such a test can be used for in-system testing of cache tags. The objective of this paper is to develop in-system tests for cache tags that (i) provide high fault coverage, (ii) have short test application times, and (iii) require little or no modification in the design of the memory subsystem.

A large number of memory tests with varying test time complexities and fault coverages have been developed for off-line testing of memory chips [1]. In [2], one of these tests, namely the March B test, has been transformed for in-system test of cache tags. In [3], a transformation methodology is proposed, which when applied to March B test gives the test given in [2]. As mentioned above, since the data part of the cache can be easily tested, most effort is expended to adapt March B to test the tag part of the cache. The difficulty arises due to the low observability of cache tags -- an incorrect tag value can only be observed indirectly via a read that causes an unexpected cache miss and causes the memory subsystem to supply the data from the main memory. Any error in tag value can only be observed if the data obtained in this manner is different from the data that would have been read from the cache if the tag value was error free. The method proposed in [2] adapts the March B test to test for stuck-at, transition, and coupling faults in tags of a direct mapped cache, assuming that the cache hardware is modified so that the cache can be disabled to provide direct access to the main memory. The complexity of this test is $33NT_M + 17NT_C$, where N is the number of block frames in the cache, T_M is the main memory access time and T_C is the cache access time.

In the following, we present a methodology to transform march tests and use it to obtain new versions of March B and March X tests for in-system testing of faults in the tag parts of direct mapped caches. The key aspect of our approach that leads to a test with complexity that is significantly lower than that of the previous test is that our transformation preserves the fault detection capability of the original test *without necessarily recreating an identical sequence of reads and writes*. Furthermore, as will be seen in the following, the resulting cache tests do not require the hardware modification needed by the previously proposed cache test.

This paper is organized as follows. Section 2 discusses the memory system model assumed in this paper, Section 3 discusses different factors that determine the efficiency of a cache test and Section 4 talks about the philosophy behind the proposed transformation methodology. Section 5 then explains the steps of the transformation

* This research was partly funded by NSF CAREER Award No. MIP-9502300.

** Sponsored by Kuwait University.

and shows how it can be applied, using March B and March X tests as examples. Section 6 demonstrates the fault coverage of the test algorithms derived in Section 5. Section 7 derives the complexity of the new March B algorithm and compares it with those of the algorithms reported in [2] and [3]. Finally, conclusions are presented in Section 8.

2. The Cache Model

The cache considered in this paper is write-back/read-through direct mapped cache [4]. Each block frame of a write-back cache has a *modified* bit that is set when a block is modified; if this bit is set, then the block is written back to the main memory when it has to be replaced. Note that, for a given address, the main memory and cache can have different data values before the modified data is replaced. The read-through feature enables the processor to obtain data directly from the main memory when a cache miss occurs.

The cache consists of a tag and a data part, as shown in Figure 1. In a direct mapped cache, the least significant bits of the address are used to select a cache block frame, α , while the remaining bits are compared with the tag value of the selected block to identify a hit or miss. This addressing strategy maps every consecutive N blocks of the main memory (where N is the number of block frames in the cache) with the same highest address bits (tags) to consecutive cache block frames. We call such blocks of data as a *cache page* and refer to it by the unique tag field value, which will be denoted by A or B . (Note that the proposed approach can be directly used for caches where each block frame has multiple words.)

Only direct mapped caches are considered in the following. However, the proposed test algorithms can be used to test set associative caches, provided that a deterministic replacement algorithm, such as First-In-First-Out (FIFO) or Least-Recently-Used (LRU), is used. With such replacement algorithms, we can select the target block frame in the cache for replacement by satisfying the replacement algorithm's conditions. In this manner, each set of a set-associative cache can be treated as a separate direct-mapped cache. This is sufficient for comprehensive

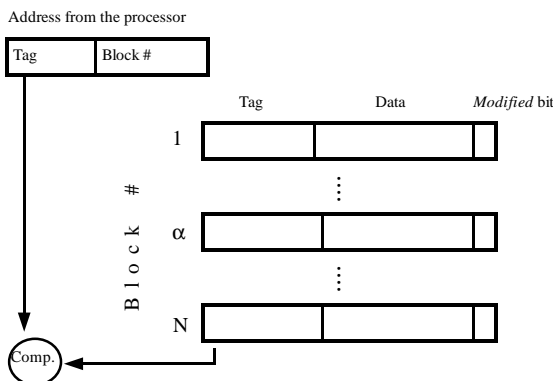


Figure 1: Model of the direct mapped cache.

testing since the sets of a set-associative cache are typically separated from each other, which eliminates the possibility of having coupling faults between cells in separate sets.

We will consider the following faults in the tags, details about the faults are given in [1].

1. Address decoder faults: The decoder used to select a block frame has a fault that makes it malfunction.
2. Stuck-At faults: A tag bit has a value 0 (SA0) or 1 (SA1) that never changes.
3. Transition faults: A tag bit accepts a transition in one direction but not the other. It can be of the following types.
 - a. $\langle \uparrow/0 \rangle$ transition fault -- a bit changes from 1 to 0 but not from 0 to 1.
 - b. $\langle \downarrow/1 \rangle$ transition fault -- a bit changes from 0 to 1 but not from 1 to 0.
 However, a coupling fault can change the value of a cell with $\langle \uparrow/0 \rangle$ ($\langle \downarrow/1 \rangle$) transition fault to 1 (0).
4. Coupling faults: A transition in one bit affects the value of another bit. They are of two types.
 - a. Inversion coupling faults (CFin): A transition in one bit inverts the value of another bit. For example, $\langle \uparrow, \downarrow \rangle$ denotes a coupling fault where a high transition in cell j inverts the value of cell i .
 - b. Idempotent coupling faults (CFid): A transition in one bit forces another bit to have a certain value. For example, $\langle \uparrow; 0 \rangle$ denotes a fault where a high transition in cell j forces cell i to have 0.
5. Bridging faults: Two bits in a tag field of a block frame have a short circuit which causes one of them to have a wrong value when opposite values are applied.

For simplicity, in the following the main memory will be simply referred to as memory.

3. Efficiency Considerations

One of the important considerations in developing an efficient cache test is to minimize direct and indirect main memory accesses. A direct main memory access occurs when a cache miss occurs during a read operation. This causes the processor to wait until it gets the data from the main memory, which increases the time complexity of the test. An indirect main memory access occurs during write misses that require moving the data in a modified block to the main memory. There is no way to eliminate the time penalty of direct main memory accesses but a write buffer can, to a limited extent, decrease the time penalty of indirect misses. A good test must minimize both direct and indirect memory accesses.

The test should also be designed to be applicable with minimal or no modification of the existing cache hardware while taking advantage of all its features. The need for hardware modifications may make the test inapplicable to memory systems that are already in use. Even in the case when a cache design can be modified, extra chip area is required to add the required hardware features which increases the memory cost. Actually, restricting the tests to use only standard cache operations will automatically eliminate any need for hardware modification.

4. Test Transformation Framework

Normal memory tests detect faults by writing a certain value to a location and then reading it, expecting to get the value that was written earlier. While this method can be applied to test data part of a cache, it can not be applied directly to test the tag part, since a tag can not be read from or written to directly. A value can be written to the tag indirectly by accessing a certain memory location. For example, 001100 can be written to the tag field of a certain block frame α by accessing the memory location [001100, α]. Observing the tag field of a cache is significantly more complicated. Since a cache miss occurs when the tag field stored in the cache does not match with the address of the location accessed by the processor, satisfying the following two conditions is sufficient for detecting a fault that must have altered, at the particular time, the value of a particular tag field:

1. The particular cache location should have a data value that is different from the data value in the corresponding memory location.
2. A read operation should be performed to obtain the data from the cache location. If the tag value is erroneous, a cache miss will occur due to a mismatch between the correct and the erroneous tag, and the data will be fetched from the main memory location.

Without loss of generality, assume that the main memory can store only one bit (0 or 1). The following is an example of a set of steps that satisfy the above conditions:

- Assume that initially the main memory location contains an unknown value, x .
- Write a 1 to the address $[A, \alpha]$ -- this will be written in cache block frame with index α , where tag value will be A and data value will be 1.
- Read from address $[A, \alpha]$. If the tag of block frame α has a fault that causes the tag value to be different from A , then a cache miss will occur.
- If the initial data in memory location $[A, \alpha]$ was 0, then the fault would be detected, since zero would be read from the memory instead of 1 from the cache.
- By repeating the above steps with data value 0, this fault can be detected in the case when the initial data in the memory location $[A, \alpha]$ was 1.

5. The Transformation Methodology

5.1 The Transformation Steps

Notations:

- $w0$ is used in the original memory tests to denote *write 0 to the memory location under test*.
- $r0$ is used to denote *read data from the location under test, expect the value 0*.
- In the following tests, $w([A, \alpha], 0)$ denotes *write 0 to the address $[A, \alpha]$, where α the index of the block frame under test (i.e. the block frame under test)*.
- $r([A, \alpha], 0)$ denotes *read the address $[A, \alpha]$, expect 0*.
- $A = a_n a_{n-1} \dots a_1$ is a given binary tag value and B is the one's complement of tag A .

- A *march element* is a set of consecutive operations performed at a block frame before moving to the next block frame, such as $\uparrow\{w([A, \alpha], 0), w([B, \alpha], 0), w([A, \alpha], 1), r([A, \alpha], 1)\}$. In all tests, we will name the march elements as $M0, M1$, and so on.
- \uparrow is the direction of a march element and indicates that α traverses through all possible block frames in some order. \downarrow indicates a traversal of all block frames in an order that is opposite of that for \uparrow .

Assumptions:

1. Data part of the cache is fault free. This is easy to justify since the data part of a cache is easy to test and hence can be assumed to have passed the tests.
2. 00...00 or 11...11 is returned when a non-existing memory location is accessed due to a fault.
3. For simplicity, we assume that \uparrow traverses block frames in the order 1, 2, ..., N .

A normal march test can be transformed for in-system testing of the tag part of a cache in the following manner.

- a. March tests usually start with the initialization step $\uparrow(w0)$. If this step exists in the original test, modify it to be $\uparrow\{w([A, \alpha], 0), w([B, \alpha], 0), w([A, \alpha], 1)\}$, otherwise add this initialization march element.
- b. For the rest of the original march test, replace each:
 - $w0$ by $w([A, \alpha], _)$,
 - $w1$ by $w([B, \alpha], _)$,
 - $r0$ by $r([A, \alpha], _)$, and
 - $r1$ by $r([B, \alpha], _)$.
- c. Add appropriate data values to the above operations in such a way that two consecutive write operations to the same address have opposite data values and each read operation has the data value corresponding to the last write to the same location.
- d. If transition faults linked with coupling faults are targeted, then for each of the four possible combinations of addresses ($[A, \alpha], [B, \alpha]$) and data (0, 1), add read operations such that they immediately follow the corresponding write operations in at least one march element with the condition that write operations to an address are not separated by a third write operation to the same address. Otherwise, it is sufficient to ensure that these four read operations appear in the test somewhere after the corresponding write operations but before the occurrence of any other write operation that would change the data value (i.e. they do not have to be in the same march element as the corresponding write operations).

We will show that the above transformation preserves the coupling fault detection capability of the original memory tests. Step d. above is necessary to cover transition faults independent of the initial data in the main memory locations, which must be assumed to be unknown.

5.2 Fault Detection Capabilities of Transformed Tests

In this section we describe some of our main results that show how the above transformation guarantees detection of various types of faults. The proofs of results can be

6.1 Address Decoder Faults

Note that the conditions for detecting address decoder faults [1] are satisfied as follows:

Condition 1: $\uparrow[r(x), \dots, w(x!)]$ is satisfied in M1, where in our case $x = A$ and $x! = B$.

Condition 2: $\downarrow[r(x!), \dots, w(x)]$ is satisfied in M3.

Proof that the satisfaction of the above conditions is sufficient for detecting AD faults is given in [1].

6.2 Stuck-at Fault

This fault occurs when a defect in a chip causes a bit to take certain fixed value (0 or 1) that can never be changed. It is denoted by SA-value.

6.2.1 SA-0 Fault in Bit 'i' of the Tag of Cache Block Frame α .

Note that:

1. When a cache access results in a miss, then it causes, among other operations, a write operation to the tag part.
2. No fault can change the value of bit i.
3. A read operation to location $[B, \alpha]$ will cause cache miss and $MData[B, \alpha]$ will be read.
4. Any value written using $w[B, \alpha]$ will be replaced to a wrong main memory location, since one of the tag bits would have a wrong value. Thus, $MIData[B, \alpha]$ will not be changed.

We will now prove the coverage of SA faults by considering two cases which correspond to the cases when the initial value in the main memory at location $[B, \alpha]$ is either 0 or 1.

◆ Case (1) - $MIData[B, \alpha] = 0$: In M1, $w[B, \alpha, 1]$ sets data at block frame α to 1 but the tag value will not be B because bit i will have the value 0 instead of 1. So, $r[B, \alpha, 1]$ will cause a cache miss and $MIData[B, \alpha] = 0$ will be read. Since the expected value is 1, the fault will be detected.

◆ Case (2) - $MIData[B, \alpha] = 1$: In M1, $r[B, \alpha, 0]$ will cause a cache miss when dealing with α because of the SA-0 fault in the block frame α . Data will be replaced to a wrong address and $MIData[B, \alpha] = 1$ will be read while a 0 is expected. Hence, the fault will be detected.

Since no other fault can change the value of bit i of the tag of block frame α , this fault can not pass our test even in the presence of other faults.

6.2.2 SA-1 Fault in Bit 'i' of the Tag of Cache Block Frame α . The detection of this fault can be demonstrated in a manner similar to above. Hence, we have the following result.

Theorem 1: The proposed March B test detects SA faults in the presence of any combination of other SA, TF and CF faults.

6.3 Transition Faults

We can now prove the detection of transition faults assuming that there is no SA fault, since any combination of faults that includes a SA fault is always guaranteed to be

detected. First, we will consider the case when only $\langle \uparrow/0 \rangle$ transition fault occurs at some bit in a tag, then only $\langle \downarrow/1 \rangle$ transition fault will be considered, and, finally, occurrence of both types of faults at different bits of the same tag will be considered. Furthermore, we will consider the first two cases assuming only a single failure among the bits of a single tag since the proof can be generalized easily.

6.3.1 $\langle \uparrow/0 \rangle$ Transition Fault at Bit i of Block Frame α .

In presence of this fault, bit i of the tag can have transition from value 1 to value 0 but not the other way. However, it is assumed that no write operation to the location $MData[B, \alpha]$ will be successful because the tag value of block frame α is set to A during initialization (M0) and will never become B unless a coupling fault changes the value of the tag bit i to 1. But, a coupling fault can not have any effect between operations on a location α within a march element, since all such operations for block frame α are performed together and no operations to another frame β can interfere. Hence, coupling faults can only have effect between march elements, since other frames are accessed and changed before applying the next march element to the frame α .

Lemma 1: In the presence of $\langle \uparrow/0 \rangle$ transition fault, either M0 ends with the values $\$Tag[\alpha] = A$, $\$Data[\alpha] = 1$ and $MData[A, \alpha] = 0$ or the first read in M1, $r[A, \alpha, 1]$, will detect the fault.

Proof:

- $w[A, \alpha, 0]$ will write 0 to $\$Data[\alpha]$.
- $w[B, \alpha, 0]$ will definitely cause a cache miss and 0 will be written to $MData[A, \alpha]$.
- $w[A, \alpha, 1]$ will write 1 to $\$Data[\alpha]$ and A to $\$Tag[\alpha]$.

After applying M0 on α , other block frames will be visited before applying M1 on α . The effect of a coupling fault on α may change its tag value from A to C, where C could be any value. If C is equal to A then the first part of the statement is true. If C is not equal to A, $r[A, \alpha, 1]$ in M1 will cause a cache miss and $MData[A, \alpha] = 0$ will be read while 1 is expected. Hence, the CF will be detected and the second part of the statement would hold. \square

Due to the above result, we can assume that either there are no coupling faults or CFs cancel the effect of each other ($C = A$) while proving that the TF will be detected.

◆ Case (1) - $MIData[B, \alpha] = 0$: In M1, $w[B, \alpha, 1]$ will not affect $MIData[B, \alpha]$ because bit i has 0 instead of 1. Also, $r[B, \alpha, 1]$ in M1 will cause cache miss due to the fault and $MIData[B, \alpha] \neq 1$ will be read while 1 is expected. Hence, the fault will be detected. As no other cells are written between these write and read operations, the TF cannot be masked by a CF.

◆ Case (2) - $MIData[B, \alpha] = 1$: In M1, $w[B, \alpha, 0]$ will not affect $MIData[B, \alpha]$ and $r[B, \alpha, 0]$ will cause cache miss due to the fault, so $MIData[B, \alpha] = 1$ will be read from the main memory while 0 is expected. Hence, the fault will be detected. As no other cells are written between these write and read operations, the TF cannot be masked by a CF.

6.3.2 $\langle \downarrow/1 \rangle$ Transition Fault at Bit i of Block Frame α .

We have also shown that the proposed test is guaranteed to detect $\langle \downarrow/1 \rangle$ transition fault in bit i of block frame α and the presence of the two different transition faults [5]. Hence we have the following result.

Theorem 2: All transition faults are detected by the proposed March B test, even in the presence of coupling faults.

6.4 Coupling Faults

We first consider the existence of CFin and CFid separately, followed by the case when both faults co-exist. Recall that a bit i is $\langle \uparrow; \uparrow \rangle$ coupled to j means that a \uparrow transition on j complements (i.e. flips) the value of i . Bits i and j can be any two bits in two tag fields of different block frames. If i and j are in the tag field of the same block frame, then they can not affect each other since they are always written at the same time.

Lemma 2: In the modified March B, if there are no transition, stuck-at and address decoder faults and no coupling fault is detected by the first read of the march element, then the fault does not have any effect on the data movement, i.e. during that march element, all operations, including replacements, occur as they would in a fault free cache.

Proof:

M0 operations will be performed correctly since no particular initial value is assumed and M0 starts with a write operation that eliminates the effect of any possible CF.

Hence, M0 ends with $MData[A, \alpha] = 0$ and $\$Data[\alpha] = 1$.

If any coupling fault causes an erroneous value in $\$Tag[\alpha]$ because of the application of M0 and M1 on the other block frames before M1 is applied to block frame α , then the $r([A, \alpha], 1)$ in M1 will cause a cache miss and obtain $MData[A, \alpha]$ which is 0 while 1 is expected. This will hence cause the detection of any faults that caused the error leading to the cache miss. On the other hand, if cache hit occurs, then the test will continue as if there is no fault and the march element will end with address $[B, \alpha]$ with data 0 in the cache and 1 in the memory. \square

It can be seen that the above statement is valid for all the other march elements.

Thus, by the above lemma, we just need to prove that there is a march element that detects a fault after passing all the previous march elements without detection, assuming that the operations in previous march elements were performed as if no fault existed.

6.4.1 Inversion Coupling Fault (CFin). As proven in [1], not all linked CFin can be detected via march tests. However, many combinations of linked CFins can be detected as shown next. The proof of detection of unlinked CFin is split into two cases determined by the position of the coupled cell relative to the coupling cells. The cell address is basically the tag index.

Let cell i in block frame α be coupled, in an identical

manner, to an odd number of cells with block frames lower than α and let β be the highest of those frames ($\beta < \alpha$) containing the coupling cell j . Then, three cases can be distinguished: i is $\langle \uparrow; \uparrow \rangle$ to j , i is $\langle \downarrow; \uparrow \rangle$ to j , or i is $\langle \uparrow; \uparrow \rangle$ and $\langle \downarrow; \uparrow \rangle$ coupled to cell j .

If i is $\langle \uparrow; \uparrow \rangle$ coupled to j , then M2 will detect the fault if it was not detected by previous march elements. Notice that if i passed M1 without fault detection, then i has the correct value before M2 starts since it is not coupled to cells at a frame with a number greater than α . In M2, \uparrow transition on j occurs which changes the value of cell i . The operation $r([B, \alpha], 0)$ will cause cache miss and 1 will be read from the main memory while 0 is expected. Hence, the fault will be detected. Notice that M1 will not detect the fault since it has two \uparrow transitions on j which will cancel the effect of each other.

It can be similarly shown that M1 will detect the faults in the other two scenarios.

The proof for the case when cell i of α is coupled only with cells in block frames $\beta > \alpha$ is similar to the above. Case (a) will be detected by M3 as well as M4, case (b) will be detected by M4, and case (c) by M3.

Theorem 3: Unlinked CFin faults are detected by the proposed March B test.

6.4.2 Idempotent Coupling Fault (CFid). This fault causes a change in bit j of a tag field in block frame β to force bit i of a tag field of another frame α to go to a certain value (either 0 or 1).

We have shown that the proposed March B test guarantees the detection of these faults. The lengthy proof has been omitted here but can be found in [5].

6.5 Bridging Faults

Since the tag fields under test consist of multiple bits, it is possible to have short circuit between two tag-bits lines causing a bridging fault. A bridging fault makes the bridged bits return one value resulting from ORing or ANDing their correct values. In order to detect this fault, we need to apply the above test with different values of A such that for each pair of bits there exists a value of A in which the bits are assigned opposite values [2].

Assume that bits i and j in frame α have a bridging fault. Take the case when the test is applied with a value of A such that $a_i \neq a_j$ (obviously $b_i \neq b_j$). All write and read operations in the test with index α will cause cache misses and replacements will never affect the correct locations in the memory because bits i and j of the tag have identical value while A and B assign different values to them. So, $MIData[A, \alpha]$ is never changed and always read. If $MIData[A, \alpha] = 1$, then the fault is detected by any $r([A, \alpha], 0)$, otherwise it is detected by any $r([A, \alpha], 1)$. Actually, it can be proven that $\hat{\Pi}\{w([A, \alpha], 0), r([A, \alpha], 0), w([A, \alpha], 1), r([A, \alpha], 1)\}$ is the only march element that needs to be repeated with different values of A to detect all bridging faults.

7. Comparison of Tests Complexities

In this section, we will compute the complexity of the new March B test and compare it with that of the previous test reported in the literature.

7.1 Complexity of the Proposed March B Test

The modified March B test has complexity of $21N$ in terms of total number of operations, where N is the number of block frames in the cache. Assuming the cache has no write-back buffer and a replacement requires a full main memory access, the complexity becomes:

- Main memory accesses: $13N$ (times when replacements occur)
- Cache accesses: $8N$ (times when cache is accessed without replacement).

The above results are obtained based on the following assumptions.

- The first memory access in $M0$ always causes a cache miss.
- Every read operation causes a cache hit since it causes a cache miss only in the presence of faults. Notice that a fault is detected when a read operation causes a cache miss (in most cases) which means that the algorithm stops execution.
- A cache miss time is the main memory access time and not the addition of the main memory and cache access times since the main memory takes a long time to complete the operation, and the cache access is easily overlapped with that operation. Note that the processor writes to the cache while the memory management unit (MMU) deals with the writing to the main memory since it handles replacements.

Furthermore, read operations in March B occur between write operations. Since a read operation always causes cache hit while write operations cause misses, the cache access in a read operation is easily overlapped with the replacement caused by a cache miss in a previous write operation. Hence, the time complexity of the modified March B is:

$$T_{\text{testing}} = 13 NT_M,$$

where T_M is the main memory access time.

7.2 Comparison With the Previous Work

We have presented a modified March B test to test the tag field of a direct mapped cache whose architecture is identical to that proposed in [2]. The complexity of our test is $13NT_M$ while that of the test proposed in [2] is $33NT_M + 17NT_C$, where T_M is main memory access time and T_C is the cache access time. In the algorithm in [2], all cache

accesses cause misses. Also, the presence of a buffer will not have any effect since the main memory accesses are all direct.

Hence, the proposed test saves at least 60.6% of the test time for the tag part of the cache memory. Furthermore, our modified March B test can work even if the cache does not have enable/disable capability and requires no hardware overhead. Note that the above test complexities are for the application of tests for a single fixed value of A . If the tests are repeated for different values of A to detect bridging faults, the complexities of both tests will increase by the same factor.

8. Conclusion

A methodology to transform march tests for in-system testing of tags of direct mapped caches is presented. The methodology is applied to obtain new versions of March B and March X tests. Unlike previously proposed tests for cache tags, these tests can be applied without making any modification to the memory system hardware. Furthermore, the proposed March B test has been shown to have significantly lower complexity than the previously proposed version.

We are currently working on applying the proposed transformation methodology to other march tests and on developing new transformation methodologies for other tests (i.e. non-march tests). We are also working on generalizing these algorithms for other cache organizations, such as set-associative caches and multilevel caches.

References

- [1] van de Goor, A.J. (1991). *Testing Semiconductor Memories. Theory and Practice*, John Wiley & Sons; Chichester, U.K.
- [2] van de Goor, A.J. and Verhallen, Th.J.W (1992). *Functional Testing of Current Microprocessors (applied to the Intel i860)*, In Proc. IEEE Int. Test Conference, pp. 684-695.
- [3] Sosnowski, J. (1995). *In System Testing of Cache Memories*, In Proc. IEEE Int. Test Conference, pp. 384-393
- [4] Hennessy, J. L. and Patterson, D. A. (1994). *Computer Organization and Design*, Morgan Kaufmann Publishers, Inc.; San Francisco, California, USA.
- [5] Al-Harbi, S. M. and Gupta, S. K. (1998). *A Methodology for Transforming Memory Tests for In-System Testing of Direct Mapped Cache Tags*. Technical Report CENG 98-01, University of Southern California.