Fault-coverage Maximizing March Tests for Memory Testing

Feng Yun, Yunkun Lin, Lou Yunfei, Lei Gao,Vaibhav Gera, Boxuan Li, Vennela Chowdary Nekkanti,

Aditya Rajendra Pharande , Kunal Sheth, Meghana Thommondru, Guizhong Ye, Sandeep Gupta

Ming Hsieh Department of Electrical and Computer Engineering

University of Southern California

Los Angeles, California, 90089, USA

{fengyun, yunkunli, yunfeilo, leig, vgera, boxuanl, vennelac, pharande, kunalche, thommond, gye, sandeep}@usc.edu

*Abstract—**Every well-known march test for memories was generated to efficiently achieve 100% coverage of a target set of fault types. The question we pursue is: What to do if 100% coverage of the given target set cannot be achieved under tight constraints on test cost? We first study an obvious option: Remove some fault types from the given target set until a new or well-known test can cover 100% of the remaining fault types under the given test cost constraint. We find that this approach leaves significant room for improvement.*

*We then pursue a different option and develop a new method which uses the original target set of fault types and generates a march test that maximizes the fault coverage under the given tight constraint on test cost. Our method generates fault-coverage maximizing tests for a wide range of target sets of fault types. A comparison with well-known march tests with equal lengths demonstrates that our new march tests provide significantly higher coverage for various sets of fault types. Importantly, our new march tests provide graceful decrease in fault coverage as we tighten constraints on test length.*

*Hence our method and new march tests enable tradeoffs between test quality and test cost and provide a new direction of memory test research focused on fault-coverage-maximization.*

# Introduction

Since its inception, memory test research has heavily focused on achieving 100% coverage of the target set of fault types. Practically every well-known march test, say MTx, started with the selection of a target set of fault types (SFT), say SFTx, followed by manual/automated generation of the shortest march test that achieved 100% coverage of all instances of faults in SFTx. E.g., March C- [2][3] was generated to provide 100% coverage for the faults in the following target set of fault types: {SAF, TF, RDF, IRF, CFin, CFid, CFst, CFtr, CFrd, CFir}. (We use the standard names and terms for fault types (see Table 1); [13] provides the behavior for each fault type.) As new fault types emerged with CMOS scaling, numbers of operations in march tests increased. Continued increases in sizes of memories further compounded the increase in memory test costs.

We consider the scenario where a test team is asked to target a set of fault types SFTx but is also given a constraint on the test cost. Specifically: What to do under ***tight constraints*** on test cost, i.e., when the constraint is such that it is not possible to achieve 100% coverage for all fault instances in SFTx?

We first study an obvious approach. We select a subset of fault types SFTy SFTx, such that there exists a well-known march test MTy that covers 100% faults in SFTy and meets the given tight test cost constraint. Then we evaluate the coverage provided by MTy for our actual target, i.e., SFTx. As MTy covers 100% faults in SFTy, this study focuses on the ***collateral coverage*** provided by MTy, i.e., coverage for fault types in SFTx that are not included in SFTy. Hence, we answer the question: Do well-known tests provide a ***high collateral fault coverage*** of fault types they were not designed to cover?

Table 1：The fault types studied in this paper (source: [13])

|  |  |  |
| --- | --- | --- |
| **Type** | **Faults** | **Full name of Fault** |
| One cell fault | SAF | Stuck at fault |
| TF | Transition fault |
| WDF | Write destructive fault |
| RDF | Read destructive fault |
| IRF | Incorrect read fault |
| DRDF | Deceptive read destructive fault |
| Two cell fault | CFin | Inversion coupling fault |
| CFid | Idempotent coupling fault |
| CFst | Static coupling fault |
| CFds | Disturb cell coupling fault |
| CFtr | Transition coupling fault |
| CFwd | Write destructive coupling fault |
| CFrd | Read destructive coupling fault |
| CFdrd | Deceptive read destructive coupling fault |
| CFir | Incorrect read coupling fault |

*Specifically, we study static one-cell and two-cell fault types and focus on single faults. Since an 18N test, March MSS, provides 100% coverage for our most comprehensive set of fault types, we study well-known march tests from 10N to 18N.* For each well-known test, we compute the fault coverage for various sets of fault types (SFTs) for which the march test could only provide <100% fault coverage. This study showed that most of the well-known tests do not provide high collateral coverage. March C-, a 10N test, is one notable exception (Figure 1).

We then faced the following question: Is the low collateral coverage for the well-known tests the maximum any test of that length can achieve? Fortunately, our first study (Figure 1) made it clear that this not the case. E.g., for our most comprehensive sets of fault types, our first study showed that the 10N test March C- provides higher fault coverage than many tests with higher lengths. This proved that there must exist other march tests with the same lengths as these well-known 11N-to-14N tests that would provide higher coverage for these sets of fault types.

To pursue the new direction due to the above surprising result, we developed a new method that, *for a given set of fault types SFTx and a tight constraint on the test length, generates a march test* ***that maximizes coverage*** *of faults in SFTx.*

Our new method generates completely new tests with much higher fault coverage for our most comprehensive SFTs, compared to well-known tests with the same number of operations. Importantly, the tests we generate provide a graceful decrease in fault coverage as we gradually tighten constraints on test length. To the best of our knowledge, the only method that solves a similar problem is [17] and we have shown that our method provides higher coverage for some sets of fault types.

Finally, we identify future research opportunities for further expanding the benefits of our new paradigm for memory testing based on fault-coverage maximization.

# Collateral coverage for well-known march tests

In this section, we present our fault-coverage oriented analysis of well-known march tests, analyze the results, and identify the new opportunity to pursue.

## The march tests and sets of fault types

We compiled a list of several well-known march tests from the research literature. Since our initial study focused on static single-faults, we collected well-known tests between 10N (e.g., March C-) and 18N (e.g., March MSS) shown in Table 2. We use the standard notation for march tests (see the references in Table 2). A march test is a sequence of *march elements*. A march element has two parts, a *loop* that traverses every memory address (denoted as , , or ) and contains a s*equence of write and read operation(s)* that is applied to one memory address before the loop moves to another address.

We also compiled a list of the sets of fault types used to generate some of these march tests to also create a library of set of fault types (SFTs) shown in Table 3. We use the standard names and terms for fault types shown in Table 1; [13] provides the behavior for each fault type. We use the term *one-cell fault* to describe any type of fault where every instance of the fault only changes the behavior of one cell (denoted as ). Also, we use the term *two-cell fault* for any type where every instance of the fault involves two cells (denoted as and ), where the fault causes the value at the cell (the victim) to change due to a specific operation/transition at cell (the aggressor) and, for some fault types, also at cell .

## Our completely-scalable fault simulator for memories

While several fault simulators (e.g., [1] [4][7][14] [16]) have been developed, the focus of the early simulators is to check whether 100% coverage is achieved for each target fault type. More recent fault simulators, e.g., [4][7][14][16], do report fault coverage values for each fault type but many do *not* combine the coverage for different fault types into a single coverage value. All these simulators use a finite-state-machine model to describe the behavior of each fault type/subtype, which we also adopt.

The early methods partition each fault *by subtypes*. Also, for two cell faults, further partition each subtype *by location* into two cases, namely and , where and are respectively the addresses of the aggressor and the victim cell. *Each partition is then treated as an equivalence set of faults* and used to reduce fault simulation run time. In contrast, the method in [14] simulates each instance of fault but does so efficiently by considering only the faults associated with a memory location when a march element performs operations on that location. Consequently, the complexity of this method ([14]) is , where is the number of locations in the memory under simulation and denotes a lower bound on the asymptotic complexity. One benefit of this is that in [4], this simulator identified corner cases for detection of dynamic faults that were *missed by previous simulators which treated each partition described above as an equivalent fault set*.

The above discovery of corner cases provided the motivation for our method where we develop a new *formal definition of fault equivalence* as well as a method to formally identify equivalent fault sets. This not only captures the corner cases discovered by [4] [14] but also ensures that corner cases are not missed as new fault types emerge. We use this new formal definition, plus associated new methods, to address the complexity challenge due to increasing memory size . Specifically, we develop a new *memory fault simulation method that is* ***completely scalable****, i.e., has complexity which is independent of the size of memory under test, i.e., it is* , and ***guaranteed to be correct***, i.e., guaranteed to not miss any corner cases. As will become clear ahead, this simulator is extremely useful since our test generator uses fault simulation extensively.

Table 2：Well-known march tests used in our studies

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Test length** | **Test Patterns** | **Ref.** |
| March C- | 10N |  | [2]  [3] |
| March AB1 | 11N |  | [4] |
| PMOVI | 13N |  | [5] |
| March U | 13N |  | [6] |
| March RAW1 | 13N |  | [7] |
| March 1/0 | 14N |  | [8] |
| March LR | 14N |  | [9] |
| March SR | 14N |  | [10] |
| March A | 15N |  | [11] |
| March B | 17N |  | [12] |
| March MSS | 18N |  | [13] |

Table 3：The sets of fault types used in our studies

|  |  |  |
| --- | --- | --- |
| **SFT Name** | **Types of faults in SFT** | **Source** |
| SFT\_10 | SAF, TF, WDF, RDF, IRF, DRDF, CFin, CFid, CFst, CFds, CFtr, CFwd, CFrd, CFdrd, CFir | March MSS [13] |
| SFT\_9 | SAF, TF, WDF, RDF, IRF, DRDF, CFin, CFid, CFst, CFds, CFtr, CFwd, CFrd, CFir | March SC16x |
| SFT\_8 | SAF, TF, WDF, RDF, IRF, DRDF, CFin, CFst, CFtr, CFwd, CFrd, CFir | March SC15x |
| SFT\_SR | SAF, TF, RDF, IRF, DRDF CFin, CFid, CFst, CFtr, CFrd, CFir | March SR [10] |
| SFT\_7 | SAF, TF, RDF, IRF, CFin, CFid, CFst, CFtr, CFrd, CFir | March C- [2][3] |
| SFT\_6 | SAF, TF, WDF, RDF, IRF, DRDF, CFin, CFst, CFtr, CFrd, CFir | March SC14x |
| SFT\_5 | SAFs, TFs, RDFs, IRFs, DRDFs, Cfin, CFst, CFtr, CFrd, CFir | March 1/0 [8] |
| SFT\_4 | SAF, TF, WDFs, RDF, IRF, DRDF, CFin, CFst, CFrd, Cfir | March SC10x |
| SFT\_3 | SAF, TF, RDF, IRF, CFin, CFid, CFst | March A [11] |
| SFT\_2 | SAF, TF, RDF, IRF, Cfin, | March X [11] |
| SFT\_AB1 | SAF, TF, WDF, RDF, IRF, DRDF | March AB1 [4] |
| SFT\_1 | SAF, RDF, IRF | MATS\_OR [11] |

## **Experimental Results:** Our collateral fault coverage study



Consider a user who would like to cover a specific set of fault types but there exists no march test that provides 100% coverage and meets the given constraint on test length.

The goal of this study was to determine if well-known march tests with the given length would provide sufficiently high fault coverage. Our broader goal was to characterize how gracefully does the fault coverage for the given set of fault types, say SFTx, decrease as we move from an march test that provides 100% coverage for SFTx, to well-known tests with lengths , , and so on.

The study was straightforward to perform but the results were extremely surprising. Figure 1 shows the fault coverage for the well-known march tests shown in Table 2, for SFT10, the most comprehensive set of fault types shown in Table 3.

*All coverage values reported in this paper are for* ***a memory with locations****.* The key observations from our study follow.

1) The shortest march test that provides 100% coverage for SFT10 must have length , hence all well-known tests with shorter lengths provide < 100% coverage for this SFT.

2) While some of these tests, e.g., March C-, provide high collateral coverage given their lengths (10N for C-), other tests provide much lower levels of collateral coverage. Also, while 100% coverage of all one-cell fault types in these SFTs can be achieved by tests, several of these tests, all of which are longer than , do not provide 100% one-cell fault coverage.

One clear observation is that well-known tests designed to cover dynamic faults (see green bars in Figure 1) and double faults (see yellow bars), provide very low collateral coverage for faults in our SFTs, which only include single static faults. This is due to the major differences in the conditions for detection of single static vs dynamic or double static faults.

Next consider the well-known tests generated to target single static faults. Our hypothesis is that tests that provide high collateral coverage have targeted SFTs that contain fault types with diverse conditions for detection while the opposite may be the case for the SFTs used to generate tests that provide low collateral coverage. Interestingly, higher collateral coverage was provided by older tests. A systematic investigation of this is one of the tasks for our future research.

3) We noticed the dramatic drop in coverage when test length decreases from (March MSS) to (March B).

a) At first, this seemed to suggest that we may not be able to reduce test length unless we are willing to accept a large

decrease in fault coverage. Fortunately, the bars in the graph show that the coverage provided by these well-known tests does

not increase monotonically with the lengths of the tests. For example, for STF10, two tests provide lower coverage than a test and equal coverage as a test. Further, if we ignore the tests generated for dynamic or double faults (the green and yellow bars) we are left with no tests for many lengths. This observation also holds for other SFTs we studied.

b) A logical implication of this non-monotonicity and these gaps is that ***there must exist other march tests that provide higher fault coverage for SFT10 (as well as other STFs) than many of these well-known tests.***

This put us on a quest to generate new march tests that provide higher collateral coverage, especially for SFT9 and SFT10.

Figure 1: Fault coverage for well-known march tests for SFT10

# Generating march tests to maximize fault coverage

We first present our method for generating march tests that maximize fault coverage under tight constraints on test length. Then we present our new march tests, the coverage these tests provide, and analysis of their key properties and benefits.

## New March Test Generator

***Problem statement:*** For a given set of fault types, say SFTx, and *a constraint on test length*, say , generate march tests that ***maximize the coverage*** of the fault instances in SFTx.

As a starting point, our method uses the set of all possible march tests of a short length, , for some low value of , as the *seed* set of tests. Then, in a step-by-step manner our method *grows* these tests to successively obtain march tests, march tests, …, , and selects only the highest and second-highest fault coverage providing march tests in each iteration. As we will see ahead, our method extensively uses our above fault simulator thereby heavily exploits its complete scalability. Next, we describe the key steps of our method.

*Step-1: Create a seed set of sets*

Our method starts by enumerating the set of all possible march tests. We can start with , or a higher value of .

To reduce its run-time complexity significantly, our method for enumeration of tests uses several properties of march tests. Specifically, it uses necessary conditions that any march test must satisfy: e.g., a operation cannot be followed by a operation; every march test must start with a write operation ( or ); and so on. We further reduce the run-time by using additional conditions to capture properties that make two march tests equivalent in terms of test length and fault coverage. For example, for every fault type in the set of fault types in Table 3, a march test MTx that starts with has a dual march test MTy where the value for every operation in MTx is complemented, such that MTx and MTy are guaranteed to provide equal fault coverage. Also, a march test MTx where the first march element has the direction has a dual march test MTy where the direction of every march element in test MTx is reversed, such that MTx and MTy are guaranteed to provide equal fault coverage. Hence, we only generate march tests that start with and where the first march element has direction .

At the end of this step, we have a comprehensive set of high-coverage march tests.

*Step 2: Select only those march tests that provide the highest and second highest fault coverage*

This requires fault simulation for each march test. When we grow the march tests in the next step, the number of tests would grow dramatically, hence this step is necessary.

At the end of this step, we have a small number of the highest and second-highest fault coverage providing march tests.

*Step 3: If , output the set of highest fault coverage providing march tests and exit; otherwise continue to the next step.*

*Step 4: Grow each of the tests selected above to obtain a comprehensive set of march tests.*

We perform three tasks to grow each of the above tests.

a) From each of the above march tests, say MTj, *extract the sequence of operations*. E.g., if we start with a march test , then the sequence of operations obtained is .

b) *Grow each of the above sequence of operations* by inserting one instance of every possible operation (i.e., , , , and ) into every location in the sequence. As in Step-1, we only insert the operations that satisfy the necessary conditions that all march tests must satisfy.

E.g., if we start with the above sequence , then we obtain the sequences: , , , and . Note that we avoid sequences that cannot provide meaningful march tests or create equivalent march tests.

c) *Create all possible march tests for each of the above sequence of operations*. We first partition, in all possible ways, a given sequence of operations into partitions, where is either or +1, where is the number of march elements in MTj. Finally, we add the direction to the first partition and every combination of directions ( and ) to each of the other partitions to obtain a set of march tests.

E.g., for the grown sequence we create the following sets of options: ; ); and . (The original test has only one march element, hence we only consider 1 or 2 partitions.) Finally, for the option , we add all combinations of directions to obtain the following march tests: ; . We repeat this for all the other options above.

At the end of this step, we have a large set of march tests. We then repeat the process, starting at Step-2.

For example, when we want to generate 12N march tests for SFT10, then we may start by generating all possible 2N march tests and repeat the above steps to create a large number of 3N march tests and select a small number of 3N tests that provide maximum and second maximimum coverage for SFT10. We repeat the entire process to generate a small number of highest coverage providing 4N tests, 5N tests, …, 12N tests.

## **Results:** New march tests that maximize fault coverage

We use our above fault-coverage maximizing march test generator for the most comprehensive set of fault types in Table 3, namely, SFT10. Recall that test is necessary to achieve 100% coverage for this SFT [13], and that we are interested in fault coverage maximized under tight constrains on test length. Hence, we use our new test generator to create new march tests that maximize coverage for each of the test lengths in the range , , , and . Our new tests are shown in Table 4 and the fault coverage (*for* ***a memory with locations***) provided by our new tests are shown in Figure 2. Our new fault-coverage maximizing march tests are called SC##x, where ## denotes the length of the march test. Also, the coverages for our new tests are shown by blue bars while those for well-known tests generated for single static faults, dynamic faults, and double faults are shown by orange, green, and yellow bars, respectively.

* + 1. A comparison of equal length tests shows that our method provides much higher coverage of faults in the target SFT, namely SFT10. *This comparison is* ***not*** *a criticism of the well-known tests since these tests were* ***not*** *generated to maximize coverage for SFT10.* In fact, fault simulations show that, while our new test SC13x provides higher coverage than PMOVI for STF10 (83% vs 76%); for SFT5, which was used to generate PMOVI, SC13x provides slightly lower coverage than PMOVI (94% vs 100%). This is true for all other (except one)

well-known march tests that we studied.

***Hence, instead of being a criticism of well-known tests, these results are a demonstration of the benefits of viewing memory testing as a fault-coverage maximization problem.***

2) These results also confirm that (see Section II) there exist march tests that provide higher coverage for these test lengths. ***Importantly, our method is able to generate tests for certain lengths for which few well known tests exist.***

3) More important is the comparison of coverage for March C- and our tests SC10x, SC11x, …, SC17x with March MSS, the test that provides 100% coverage for this SFT. The figure shows that if tight test cost constraints limit us to tests with lengths , , , and so on, then our new method provides a way to ensure that the fault coverage decreases gracefully with increasingly tight constraints on test length. Hence, ***these results demonstrate the practical viability of the paradigm based on fault-coverage maximization (instead of focusing on 100% coverage).***

Finally, the only method that solves a similar problem is [17]. Their method and ours both develop tests by inserting operations into tests in a step-by-step manner. Analysis of the two methods shows that, in absence of run-time constraints, our method will generate all possible tests while theirs may miss some. However, practical run-time constraints require both methods to perform the insertion step only for selected tests. A comparison shows that for many of our SFTs both methods provide similar coverage while for SFT8 our method provides higher coverage for most test lengths.

Table 4：Our new coverage-maximizing march tests for SFT10

|  |  |  |
| --- | --- | --- |
| **Name** | **Test length** | **Test Patterns** |
| March SC10x | 10N |  |
| March SC11x | 11N |  |
| March SC12x | 12N |  |
| March SC13x | 13N |  |
| March SC14x | 14N |  |
| March SC15x | 15N |  |
| March SC16x | 16N |  |
| March SC17x | 17N |  |
| March SC18x | 18N |  |



Figure 2：Fault coverage for our new coverage-maximizing march tests for SFT10, along with well-known march tests

# Conclusion and future research

Our goal was to undertake the first fault-coverage oriented analysis of existing march tests to determine whether the fault coverage decreases gracefully as we apply increasingly tight constraints on test length. The purpose of this was to determine if it is possible to reduce memory test costs by trading-off a small decrease in coverage.

To enable this, we developed a fault simulator for memory cell faults that is completely scalable, i.e., independent of memory size, yet guarantees correctness for new types of memory faults.

Our first study of fault coverage of well-known march tests for comprehensive sets of fault types (SFT9 and SFT10) provided a major surprise: many of the march tests provide low collateral fault coverage. This motivated us to develop a march test generator that maximizes fault coverage for a given constraint on test length. We used this to generate a family of march tests that provide significantly higher fault coverage than well-known tests. Our new tests provide graceful reduction in fault coverage as we impose increasingly tighter constraints on test lengths.

Our ongoing and future research include basic questions about fault coverage-oriented testing, such as how do we compute fault coverage for fault types (one-/two-cell; single/double) that differ dramatically in the numbers of faults and related issues in computing probabilities of occurrence of various faults. We are also developing systematic algorithms for fault coverage maximizing memory test generation.

# Acknowldegement

This research was funded by National Science Foundation.

##### References

1. J. Van de Goor and B. Smit, "Automating the verification of memory tests," Proceedings of IEEE VLSI Test Symposium, Cherry Hill, NJ, USA, 1994, pp. 312-318.
2. M. Marinescu, “Simple and Efficient Algorithms for Functional RAM Testing”, In Proc. of IEEE International Test Conference, 1982.
3. A.J. van de Goor, “Testing Semiconductor Memories, Theory and Practice”, ComTex Publishing, Gouda, The Netherlands, 1998.
4. A. Benso, A. Bosio, S. Di Carlo, G. Di Natale and P. Prinetto, "March AB, March AB1: new March tests for unlinked dynamic memory faults," IEEE International Conference on Test, 2005., 2005, pp. 8 pp.-841.
5. J.H. De Jonge and A.J. Smeulders, “Moving Inversions Test Pattern is Thorough, Yet Speedy”, In Comp. Design, pp. 169-173, 1976.
6. A.J. van de Goor and G.N. Gayadadjiev, “March U: A Test for All Unlinked Memory Faults, IEE Proc. of Circuits Devices and Systems, V. 144, No. 3, pp. 155-160, 1997.
7. S. Hamdioui, Z. Al-Ars and A. J. van de Goor, "Testing static and dynamic faults in random access memories," Proceeding’s 20th IEEE VLSI Test Symposium (VTS 2002), 2002, pp. 395-400.
8. M.A. Breuer and A.D. Friedman, Diagnosis and Reliable Design of Digital Systems, Computer Science Press, Woodland Hills, Calif., 1976.
9. A.J. van de Goor and G. Gaydadjiev, “March LR: A Memory Test for Realistic Linked Faults”, In Proc. IEEE VLSI Test Symposium, 1996.
10. S. Hamdioui, A.J. van de Goor, “Experimental Analysis of Spot Defects in SRAMs: Realistic Fault Models and Tests”, In Records of ATS, pp. 131-138, Taipei, Taiwan, 2000.
11. RAM Fault models and Memory Testing, Cheng-Wen Wu, Lab for realiable computing (LaRC), NTHU.
12. D.S. Suk and S.M. Reddy, “A March Test for Functional Faults in Semiconductor Random-Access Memories,” lEEE Trans. Computers, Vol. C-30, No. 12, 1981, pp. 982-985.
13. S. Hamdioui, Ad J. van de Goor, M. Rodgers, “March SS: A Test for All Static Simple RAM Faults”, MTDT 2002: IEEE International Workshop on Memory Technology, Design and Testing, 2002 pp. 95 – 100.
14. A. Benso, S. Di Carlo, G. Di Natale and P. Prinetto, "Specification and design of a new memory fault simulator," Proceedings of the 11th Asian Test Symposium, 2002. (ATS '02)., 2002, pp. 92-97.
15. S. Hamdioui, Z. Al-Ars, A. J. van de Goor and M. Rodgers, "Linked faults in random access memories: concept, fault models, test algorithms, and industrial results," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, no. 5, pp. 737-757, May 2004.
16. Kuo-Liang Cheng, Jen-Chieh Yeh, Chih-Wea Wang, Chih-Tsun Huang and Cheng-Wen Wu, "RAMSES-FT: a fault simulator for flash memory testing and diagnostics," Proceedings 20th IEEE VLSI Test Symposium (VTS 2002), 2002, pp. 281-286.
17. Chi-Feng Wu, Chih-Tsun Huang, Kuo-Liang Cheng and Cheng-Wen Wu, "Fault simulation and test algorithm generation for random access memories," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 4, pp. 480-490, April 2002.