Fault Diagnosis Using Test Primitives in Random Access Memories

Zaid Al-Ars Said Hamdioui
Computer Engineering Lab., Faculty of EE, Mathematics and CS
Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
E-mail: z.al-ars@tudelft.nl

Abstract

As diagnostic testing for memory devices increasingly gains in importance, companies are looking for flexible, cost effective methods to perform diagnostics on their failing devices. This paper proposes the new concept of test primitives as a method to diagnose memory faults. Test primitives provide an easy-to-use, extensible, low-cost memory fault diagnosis method that is universally applicable, since it uses simple platform-independent test sequences. The paper defines the concept of test primitives, shows their importance and gives examples to the way they are derived and used in a memory test environment.

Keywords: *test primitives, fault primitives, memory testing, fault diagnosis, test optimization*

1 Introduction

Diagnostic testing is becoming an important issue for memory devices. For manufacturers at many different levels of the memory supply chain, be it for large commodity memory producers, for memory IP design houses or for companies using memory devices in their systems, it is not enough to run memory tests designed only to detect memory faults, but it is also necessary to run tests for memory fault localization and diagnosis. On the one hand, this is driven by the need to curb the continued increase in test cost of newer ICs, by diagnosing the faults actually taking place in the memory and limiting the used memory tests set to detect these faults only. On the other hand, the on-going fragmentation of the IC production process forces memory designers, memory manufacturers and test-service providers to use standardized, easy-to-implement test methods that enables effective transfer of test information between different companies [Vermeulen04].

Diagnostic testing for memory devices has been studied by many researchers in the past, and has seen more active investigation in recent years. David[90] proposed a fault diagnosis method based on running pseudo-random test experiments and comparing pass/fail data with statistically generated fault probabilities. This method is not deterministic and is rather time consuming in terms of test time. Yarmolik[96] presented a diagnostic memory test that is able to distinguish between a number of specific fault models by recording the read operation that resulted in first memory fail. Niggemeyer[00] and then Li[01], on the other hand, introduced the idea of fault diagnosis using output tracing, which involves keeping track of the pass/fail information of every read operation in the diagnostic test, thereby generating a signature for each fault. These tests are designed to diagnose a limited number of memory faults.

More recently, the concept of fault primitives [Al-Ars03] has been used as a basis for diagnosis activities. Fault primitives describe the whole space of memory faults, and have been shown by researchers to correspond to memory faulty behavior observed in practice [Huott99, Borri03]. Recent fault diagnosis work proposes a specific march-like test in combination with a signature for every targeted fault primitive [Harutunyan06, Al-Harbi07]. Clearly, these methods are hardwired to a specific predefined diagnostic test and any modifications to the set of targeted faults needs a new diagnostic test along with a new set of fault signatures.

In this paper, we propose the concept of *test primitives (TP)*, which is a new more flexible approach to fault diagnosis, that is both extensible and implementation independent. Extensible in the sense that new diagnosis capabilities for new fault primitives can be added easily without the need to modify existing tests or signatures. Implementation independence means that the tests do not require any specific test implementation techniques, other than running a test and identifying the pass/fail status of the test. Diagnostic tests proposed so far require keeping track of fail signatures (i.e., pass/fail status of specific read operations within the test), something that is not generally possible on every memory test platform.

This paper is organized as follows. Section 2 defines the concept of test primitives and discusses its advantages and limitations. Section 3 outlines a procedure to be used to

1081-7735/09 $26.00 © 2009 IEEE
DOI 10.1109/ATS.2009.79

create a diagnostic dictionary needed to perform TP-based fault diagnosis. Section 4 presents the way to apply the concept of TPs to diagnose single-cell static FPs. Section 5 ends with the conclusions.

2 Concept of test primitives

This section presents the concept of TPs, and discusses their advantages and limitations. The section starts by defining the space of faults to be diagnosed using the TPs in this paper. Then the test notation used for TPs is presented. Finally, the concept of TPs is introduced and discussed.

2.1 Fault primitives

In order to specify a certain memory fault, one has to represent it in the form of a *fault primitive (FP)*, denoted as $<S/F/R>$. S describes the operation sequence that sensitizes the fault, F describes the logic level in the faulty cell ($F \in \{0, 1\}$), and R describes the logic output level of a read operation ($R \in \{0, 1, -\}$). R has a value of 0 or 1 when the fault is sensitized by a read operation, while the "$-$" is used when a write operation sensitizes the fault. For example, in the FP $<0w1/0/->$, which is the up-transition fault (TF$_1$), $S = 0w1$ means that a $w1$ operation is written to a cell initialized to 0. The fault effect $F = 0$ indicates that after performing $w1$, the cell remains in state 0. The output of the read operation ($R = -$) indicates there is no expected output for the memory.

Functional fault models (FFMs) can be defined as a non-empty set of FPs. This paper targets the class of single-cell static FFMs, which is the most important class of FFMs. Single-cell static FFMs consist of FPs sensitized by performing at most one operation on a faulty cell.

Table 1 lists all single-cell static FFMs and their corresponding FPs. In total, there are 6 different types of FFMs: state fault (SF), transition fault (TF), write destructive fault (WDF), read destructive fault (RDF), incorrect read fault (IRF), and deceptive read destructive fault (DRDF) [Adams96].

Table 1. Single-cell static FFMs and their FPs.

#	Fault	FP	Name
1	SF	$<0/1/->$, $<1/0/->$	State fault
2	TF	$<0w1/0/->$, $<1w0/1/->$	Transition fault
3	WDF	$<0w0/1/->$, $<1w1/0/->$	Write destructive fault
4	RDF	$<0r0/1/1>$, $<1r1/0/0>$	Read destructive fault
5	IRF	$<0r0/0/1>$, $<1r1/1/0>$	Incorrect read fault
6	DRDF	$<0r0/1/0>$, $<1r1/0/1>$	Deceptive RDF

2.2 Test notation

Test primitives use the same notation as march tests, which is defined as a finite sequence of march elements [Suk81, vdGoor98]. A march element is a finite sequence of memory operations applied in order on a specific memory cell before proceeding to the next cell. The way one proceeds from one cell to the next is determined by the address order which can be an increasing address order (e.g., increasing address from the cell 0 to the cell n-1), denoted by the $\Uparrow$ symbol, or a decreasing address order, denoted by the $\Downarrow$ symbol. The $\Uparrow$ and $\Downarrow$ address orders should be the exact opposite of each other. When the address order is irrelevant, the symbol $\Updownarrow$ (i.e., $\Uparrow$ or Dn) will be used. A memory operation can either be $w0$, $w1$, $r0$ or $r1$. A complete march test is delimited by the '{...}' bracket pair, while a march element is delimited by the '(...)' bracket pair. The march elements in the test are separated by semicolons, while the operations within a march element are separated by commas. As an example of a march test, the MATS+ = $\{\Updownarrow(w0); \Uparrow(r0, w1); \Downarrow(r1, w0)\}$ consist of the march elements $\Updownarrow(w0)$, $\Uparrow(r0, w1)$, and $\Downarrow(r1, w0)$. The first march element initiates all the memory cells to 0, the second march element reads 0 followed by writing 1 to each cell, while the third march element works in the opposite way of the second element.

2.3 Test primitives

A *test primitive (TP)* is defined as a march test that satisfies the following two criteria:

1. **Uniqueness**—the test is constructed to detect a specific FP (called the *target FP*) or a set of externally identical FPs
2. **Minimality**—the test should contain the minimum number of operations needed to diagnose the target FP

It is important to note that a TP does not only detect the target FP, but it may also detect other FPs that share (part of) the sensitizing operation sequence of the target FP. For example, the FP $<0w1/0/>$ representing transition fault 1 has the following corresponding TP $\{\Updownarrow(w0); \Updownarrow(w1); \Updownarrow(r1)\}$. This TP detects other FPs such as read destructive fault 1 $<1r1/0/0>$. However, if a TP *does not* result in a fault in the memory under test, this means that the target FP does not take place in the memory. This way, by performing a collection of increasingly more complex test primitives on the memory, it is possible to diagnose the faulty behavior of the memory. This requires a dictionary that maps pass/fail information of different TPs to a specific set of FPs.

There are a number of advantages to test primitives.

- Extensibility: new FPs can be added to the diagnosis process if desired without affecting the diagnosis of already existing FPs. By the same token, a specific set of TPs can be easily reduced to diagnose subsets of the original set of diagnosable FPs, without affecting the diagnosis of other FPs (some conditions apply).

- Platform independence: TPs are applied in the same regular way any detecting memory test is applied to the memory. There is not extra memory needed to store test signatures, or the specific read operation failing. This makes them implementable on *any* memory test platform irrespective of its capabilities. The only thing needed is to store pass/fail information of the whole set of test primitives to perform the diagnosis.

- Unknown fault identification: Some combinations of failing TPs might not be described by any known fault. In this case, TPs make it possible to *empirically* define a fault, and subsequently derive a test for it. This is a very powerful characteristic of TPs that can augment the limited theoretical understanding of new faults.

- Customer returns analysis: A product that functions correctly through production testing in the manufacturing company may still fail in the field at the customer side. These field fails (referred to as *customer returns*) are shipped back to the manufacturer who needs to adjust the test process to prevent delivering components with same fails to the customer. The manufacturer can use TPs to identify what faults are taking place in the products, or even define them in case unknown faults are taking place. Then the test process can be modified to account for these faults.

- Test program optimization: Using TPs to diagnose faults in a product provides information about the probabilities of different faults taking place in the product. This information can be used to optimize the test program by performing trade-off analysis between fault coverage and test cost, depending on the required outgoing product quality.

Fault diagnosis using external memory tests has inherent diagnostic limitations. Since tests are applied externally to the memory, performing a read operation is the only way to identify the content of a memory cell. Therefore, if a read operation fails, the test cannot distinguish between an incorrect value stored in the cell or an incorrect read operation.

In general, two FPs are called *externally identical FPs* if they have the same detection characteristics for any memory test. In other words, there is no test that is able to identify a difference between these faults. One class of externally identical faults is represented by FPs that only differ in their internal behavior. For example, the two faults RDF$_1$ = $<1r1/0/0>$ and IRF$_1$ = $<1r1/1/0>$ in Table 1 have the same behavior externally since both result in a faulty 0 at the output while trying to perform a $r1$ operation. However, *internally* they have a different behavior since RDF$_1$ also results in changing the value of the cell, while IRF$_1$ leaves the cell content intact. More precisely, if FP$_1$ = $<S_1/F_1/R_1>$ and FP$_2$ = $<S_2/F_2/R_2>$, then FP$_1$ and FP$_2$ are called externally identical in case $S_1 = S_2$ and $R_1 = R_2$. These two *different* faults result in the same value on the output of the memory using the same sensitizing operation sequence. Therefore, these two faults are externally identical (i.e., no external test exists that can distinguish between the two). TPs are not required to distinguish between externally identical faults.

3 Generation of TP dictionary

In order to use TPs to diagnose memory faults, we need to generate a TP dictionary that maps the pass/fail characteristics of a set of TPs to their corresponding faults. The TP diagnostic dictionary is produced in four steps.

1. Generate a TP for each targeted FP.
2. Create a table listing the detection capabilities of each TP on the set of targeted FPs.
3. Perform faults classification.
4. Optimize the table to create the TP dictionary.

In the following these steps are discussed in more detail.

3.1 TP generation

First of all, a set of targeted FPs is chosen to be diagnosed. For each targeted FP, the detection condition is derived in order to generate a corresponding march test that detects it. This march test will represent a TP for the targeted FP if it satisfies the uniqueness and minimality criteria for TPs. To meet the uniqueness requirement, it is desired that the march elements of a TP contain the minimum number of operations to sensitize and detect the targeted FP. This is in order to prevent the TP from sensitizing other non-targeted FPs.

To meet the minimality requirement for a single-cell FP, we just need to convert each sensitizing operation in the detection condition into its own march element, and end the TP with a rx. This will result in the shortest TP in most cases. For example, to generate a TP for $<0w1/0/->$, we start by an initializing $w0$, followed by a sensitizing $w1$

and end with a detecting $r1$. This results in the following TP $\{\Updownarrow(w0); \Updownarrow(w1); \Updownarrow(r1)\}$.

When producing TPs for FPs that are *not* externally identical, we might generate a TP that is the same as an existing one. In that case, it is advised to keep this TP rather than generating a new one, since diagnosis between two FPs does not only rely on their own TPs but also depends on all the signatures from all TPs. As a result, it is possible for two FPs with the same TP to have different signatures in the dictionary through other TPs. However, if after constructing the diagnosis dictionary it turns out that two externally distinguishable FPs with the same TP have exactly the same signatures for all TPs, we can add a new TP to identify the required FP.

Example 1: Consider the following 4 FPs we would like to generate a set of TPs for: WDF0 = $<0w0/1/->$, SF0 = $<0/1/->$, IRF0 = $<0r0/0/1>$ and TF0 = $<1w0/1/->$. According to the procedure described above, we now generate the corresponding TPs for each of these FP: TWDF0 = $\{\Updownarrow(w0); \Updownarrow(w0); \Updownarrow\ (r0)\}$, TSF0 = $\{\Updownarrow(w0); \Updownarrow(r0)\}$, TIRF0 = $\{\Updownarrow(w0); \Updownarrow(r0)\}$ and TTF0 = $\{\Updownarrow(w1); \Updownarrow(w0); \Updownarrow(r0)\}$, respectively. Note that two TPs in the TP list are identical (TSF0 = TIRF0), however it might not be necessary to generate a separate TP for either of them.

3.2 TP detection capabilities

The next step in the TP dictionary generation process is to evaluate the detection capabilities of each generated TP in Section 3.1 on the set of targeted FPs. This is done by listing all the TPs in a table and indicating whether they are capable of detecting each targeted FP. In case some TPs are identical, they are listed only once. This makes it possible to identify a so-called *pass/fail signature* for the targeted set of FPs, such that diagnosis can take place.

In some cases, the signatures are not able to distinguish all available FPs in the table, which happens when for example a couple of FPs have the same signatures. In this case, new TP(s) should be added such that each FP has a unique signature (if possible) to facilitate diagnosis.

To illustrate this step, we consider the resulting TPs in Example 1. Since TSF0 and TIRF0 are identical, we select TSF0 to be included in the diagnosis table, as shown in Table 2. Each row in the table represents one of the targeted FPs, while the columns represent the generated TPs. Each entry in the table contains a 1 to indicate that the TP is able to detect the corresponding FP, while a 0 indicates otherwise.

Comparing the signatures of the FPs in the table shows that both SF0 and IRF0 cannot be distinguished using the provided set of TPs. It would be desirable to provide a set of TPs able to tell apart SF0 and IRF0. This is, however, not possible since state faults are purely internal faults that get sensitized without performing any external operations. This makes it impossible diagnose them using only externally applied tests.

Table 2. Pass/fail signatures of targeted FPs in Example 1.

FP name	TSF0	TWDF0	TTF0
SF0	1	1	1
WDF0	0	1	0
TF0	0	0	1
IRF0	1	1	1

3.3 Perform faults classification

FPs in a signature table can be classified into three different classes:

- Externally identical FPs
- Externally distinguishable FPs
- Unknown FPs

Externally identical FPs refer to two FPs that have identical pass/fail signatures for all possible TPs. In other words, there is no test that is able to distinguish between these two faults, since memory tests are based on external observations of the behavior. Taking into consideration that SF0 and IRF0 are externally identical (see Table 2), we can lump these together into one row, as shown in Table 3.

Table 3. Combining identical FPs.

FP name	TSF0	TWDF0	TTF0
SF0, IRF0	1	1	1
WDF0	0	1	0
TF0	0	0	1

Externally distinguishable FPs are those that show a difference in behavior for a given TP, which results in a difference in their signatures in the signature table. For example, WDF0 and TF0 are distinguishable.

Unknown FPs are faults with signatures that do not correspond to the faulty behavior of any known fault model. This can take place, for example, when applying TPs on actual memory devices, where some types of faulty behavior results in unexpected pass/fail information. Such signatures represent faults that are not understood and not yet modeled. This presents a unique opportunity to empirically define and model such faults using their signatures under specific TPs. The number of possible unknown faults can be quantified as follows. Suppose there are m targeted FPs with different signatures that correspond to n different TPs, then there will be $2^n - m - 1$ possibilities for unknown FPs. In case the number of possible unknown signatures is small, these can be included in the signature table, otherwise these could be left out. Table 4 shows the unknown signatures possible for the TPs generated in Example 1.

Table 4. Signatures for unknown FPs.

FP name	TSF0	TWDF0	TTF0
SF0, IRF0	1	1	1
WDF0	0	1	0
TF0	0	0	1
Unknown FP1	0	1	1
Unknown FP2	1	1	0
Unknown FP3	1	0	0
Unknown FP4	1	0	1

3.4 Signature table optimization

The final step in the process of creating the FP dictionary is to optimize the signature table with the objective of minimizing the test cost. For the optimization process, we use a signature table that does not include the unknown FPs shown in Table 4. Next, we remove (if possible) some TPs with two conditions: 1. already distinguishable FPs remain distinguishable after removal, 2. we should guarantee that no FP has a signature with all 0 after this removal. Such removable TPs are called *redundant TPs*. Taking Example 1 for illustration, it is possible to show that TSF0 is a redundant TP and is thus removable. TWDF0 and TTF0 are on the other hand not redundant, since removing either of them will render some FPs undetectable by assigning them an "all 0" signature. Table 5 shows the final signature table for the set of FPs described in Example 1.

Table 5. Signature table for Example 1.

FP name	TWDF0	TTF0
SF0, IRF0	1	1
WDF0	1	0
TF0	0	1

Note that performing Step 3 (Section 3.3) and Step 4 (Section 3.4) on the signature generation process are optional from an industrial point of view. In practice, in the initial yield learning phase of the fabrication process, a test engineer may be more interested in detecting and characterizing new unknown types of faulty behavior in the manufactured chips. In this case, it is more advisable to skip the optimization step (Step 4) and spend more test time attempting to detect unexpected faulty behavior. However, in the later phases of the fabrication process, where the process is well-understood and production goes into high-volume, test time becomes critical. In this case, a test engineer may choose to skip identifying unknown faults (Step 3) and focus on test set optimization in Step 4.

4 TPs for single-cell static FPs

In this section, we apply the concept of TPs as a diagnostic tool for the most well-known memory fault class: the class of single-cell static faults (discussed in Section 2.1). This serves as a case study of the capabilities of TPs in practice. The same approach can be used to diagnose any fault class in the space of fault primitives (for example, two-cell faults, dynamic faults, etc.). In order to generate the diagnostic dictionary for the set of single-cell static faults, the procedure outlined in the previous section will be used. The following discusses the different steps needed to generate the dictionary:

1. The first step is to create a TP for each targeted FP. Table 6 lists the TPs required for detecting individual single-cell static FPs.

Table 6. List of TPs for single-cell static FPs.

FP name	FP	TP	TP name
SF0	$<0/1/->$	$\{\Updownarrow(w0);\Updownarrow(r0)\}$	TSF0
SF1	$<1/0/->$	$\{\Updownarrow(w1);\Updownarrow(r1)\}$	TSF1
TF0	$<1w0/1/->$	$\{\Updownarrow(w1);\Updownarrow(w0);\Updownarrow(r0)\}$	TTF0
TF1	$<0w1/0/->$	$\{\Updownarrow(w0);\Updownarrow(w1);\Updownarrow(r1)\}$	TTF1
WDF0	$<0w0/1/->$	$\{\Updownarrow(w0);\Updownarrow(w0);\Updownarrow(r0)\}$	TWDF0
WDF1	$<1w1/0/->$	$\{\Updownarrow(w1);\Updownarrow(w1);\Updownarrow(r1)\}$	TWDF1
RDF0	$<0r0/1/1>$	$\{\Updownarrow(w0);\Updownarrow(r0)\}$	TRDF0
RDF1	$<1r1/0/0>$	$\{\Updownarrow(w1);\Updownarrow(r1)\}$	TRDF1
IRF0	$<0r0/0/1>$	$\{\Updownarrow(w0);\Updownarrow(r0)\}$	TIRF0
IRF1	$<1r1/1/0>$	$\{\Updownarrow(w1);\Updownarrow(r1)\}$	TIRF1
DRDF0	$<0r0/1/0>$	$\{\Updownarrow(w0);\Updownarrow(r0);\Updownarrow(r0)\}$	TDRDF0
DRDF1	$<1r1/0/1>$	$\{\Updownarrow(w1);\Updownarrow(r1);\Updownarrow(r1)\}$	TDRDF1

2. Then, we incorporate pass/fail information for the TPs and their corresponding FPs. In this step, the pass/fail information shows that TSF0, TRDF0 and TIRF0 have the same signature. The same is true for TSF1, TRDF1 and TIRF1. Therefore, the TRDF0, TIRF0, TRDF1 as well as TIRF1 are removed, leaving only TSF0 and TSF1 in the table.

3. Next, we perform fault classification, which shows that there are $2^8 - 8 - 1 = 247$ unknown faults.

4. Finally, we optimize the signature table in order to reduce the test time by removing redundant TPs. Table 7 shows the optimized TP dictionary for single-cell static FPs (the columns for TSF0 and TSF1 are deleted since they are redundant).

Table 7. TP dictionary for single-cell static FPs.

FP name	TTF0	TTF1	TWDF0	TWDF1	TDRDF0	TDRDF1
SF0, RDF0 & IRF0	1	0	1	0	1	0
SF1, RDF1 & IRF1	0	1	0	1	0	1
TF0	0	1	0	0	0	0
TF1	1	0	0	0	0	0
WDF0	0	0	1	0	0	0
WDF1	0	0	0	1	0	0
DRDF0	0	0	0	0	1	0
DRDF1	0	0	0	0	0	1

Using the TPs listed in Table 6 and the associated dictionary in Table 7, it is possible diagnose single-cell static FPs that are not externally identical. As mentioned in Section 2.3, performing diagnostic testing using this set of TPs provides a number of advantages when compared to well-known approaches that use a single test. For example, this test set is extensible, which means that new classes of FPs (e.g., two-cell faults) can be diagnosed by adding the associated TPs. A second advantage is the platform independence of this test set, meaning that only simple pass/fail information of each TP is needed to perform diagnosis, instead of signature analysis for each read operation in the test, which is not possible for a generic test platform. In addition, the same information provided by this set of TPs can be used to identify unknown faults in the behavior of the failing device, making it possible to perform a much more thorough diagnosis of the faulty behavior of the device under test than with other diagnostic approaches.

5 Conclusions

In this paper, we introduced the concept of test primitives used to diagnose the faulty behavior of memory devices. This new diagnosis method has a number of advantages over conventional diagnostic testing. One advantage is that test primitives are extensible and flexible, such that additional tests can be added to diagnose new faults that are not included originally. Platform independence is another advantage, enabling the application of test primitives on any memory testing system, without the need for specialized changes to the existing test infrastructure. The new diagnosis method is also able to empirically model new, unknown faults in a memory product. Furthermore, the paper presents a procedure to perform fault diagnosis using test primitives for any given set of faults. Finally, the paper presents a set of test primitives and their associated diagnostic dictionary to be used to diagnose the class of single-cell static faults.

References

[Adams96] R.D. Adams and E.S. Cooley, “Analysis of a Deceptive Read Destructive Memory Fault Model and Recommended Testing”, *in Proc. IEEE North Atlantic Test Workshop*, 1996.

[Al-Ars03] Z. Al-Ars and A.J. van de Goor, “Static and Dynamic Behavior of Memory Cell Array Spot Defects in Embedded DRAMs,” *in IEEE Trans. on Computers*, vol. 52, no. 3, 2003, pp. 293-309.

[Al-Harbi07] S.M. Al-Harbi, F. Noor and F.M. Al-Turjman, “March DSS: A New Diagnostic March Test for All Memory Simple Static Faults,” *in IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, 2007, pp. 1713–1720.

[Borri03] S. Borri, M. Hage-Hassan, P. Girard, S. Pravossoudovitch and A. Virazel, “Defect-Oriented Dynamic Fault Models for Embedded-SRAMs,” *in Proc. European Test Workshop*, 2003, pp. 23–28.

[David90] R. David and A. Fuentes, “Fault Diagnosis of RAMs from Random Testing Experiments,”, *in IEEE Trans. on Computers*, vol. 39, no. 2, 1990, pp 220–229.

[Huott99] W. Huott *et al.*, “The Attack of the ‘Holey Shmoos’: A Case of Advanced DFD and Picosecond Imaging Circuit Analysis (PICA),” *in Proc. of IEEE Int’l Test Conference*, 1999, pp. 883–891.

[Li01] J.-F. Li, K.-L. Cheng, C.-T. Huang and C.-W. Wu, “March-Based RAM Diagnosis Algorithms for Stuck-at and Coupling Faults,” *in Proc. IEEE int’l Test Conf.*, 2001, pp. 758–767.

[Niggemeyer00] D. Niggemeyer, M. Redeker, E.M. Rudnick, “Diagnostic testing of embedded memories based on output tracing,” *in Proc. IEEE Int’l Workshop on Memory Technology, Design and Testing*, 2000, pp. 113–118.

[Harutunyan06] G. Harutunyan, V.A. Vardanian and Y. Zorian, “Minimal March-Based Fault Location Algorithm with Partial Diagnosis for All Static Faults in Random Access Memories,” *in Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems*, 2006, pp. 260–265.

[Suk81] D.S. Suk and S.M. Reddy, “A March Test for Functional Faults in Semiconductor Random Access Memories”, *In IEEE Trans. on Comp.*, vol. 30, no. 12, pp. 982–985, 1981.

[vdGoor98] A.J. van de Goor, “Testing Semiconductor Memories, Theory and Practice”, *ComTex Publishing*, Gouda, The Netherlands, 1998.

[Vermeulen04] B. Vermeulen, C. Hora, B. Kruseman, E.J. Marinissen and R. van Rijsinge, “Trends in Testing Integrated Circuits,” *in Proc. IEEE Int’l Test Conf.*, 2004, pp. 688–697.

[Yarmolik96] V.N. Yarmolik, Yu.V. Klimets, A.J. van de Goor and S.N. Demidenko, “RAM Diagnostic Tests,” *in Proc. IEEE Int’l Workshop on Memory Technology, Design and Testing*, 1996, pp. 100–102.