Automatic Analysis of Memory Faulty Behavior in Defective Memories

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

Abstract: *As the complexity of memory faulty behavior increases, it is becoming more difficult to precisely identify the faults the memory exhibits. Knowledge of the precise set of faults is essential for designing an optimal set of memory tests with low test time and high fault coverage. This paper presents an automatic method to analyze the observed faulty behavior and to map it precisely into corresponding faults. The method is unique in its generality, making it possible to identify both static as well as dynamic faults in the behavior. Depending on the complexity of the performed fault analysis, different algorithms may be used, with increasing level of computational complexity.*

Keywords: *functional fault models, systematic fault analysis, precise faults, fault identification, test optimization.*

1 Introduction

In the memory market today, memory tests are required to have an increasingly higher fault coverage to ensure product quality. On the other hand, high test costs and the stiff competition in the memory market make short test times an economic necessity. In order to achieve the short test times, it is important to construct an optimal test set, which requires the *precise* knowledge of the *functional fault models (FFMs)* exhibited by the memory behavior. Once the FFMs in the faulty behavior are precisely identified, a corresponding set of optimal memory tests can be automatically generated [1, 2].

Previous work on FFM identification used functional information to identify a limited number of FFMs, such as some single-cell and two-cell faults [3, 5], while assuming that read operations work correctly. However, the applied algorithms cannot be extended to account for other more complex types of FFMs. More recent fault analysis methods do not apply standard algorithms to identify the observed faulty behavior [7] and usually refer to electrical information (voltages and currents) to identify the presence of a given type of FFM [9, 4, 8]. Since each memory has a distinct electrical design, using electrical information leads to an unclear presentation of fault analysis findings, and makes it difficult to compare published results.

This paper provides a general fault analysis method, applicable to all FFMs. The method only uses functional information (cell contents and memory outputs) and does not refer to electrical data to carry out the identification. Since the general method has an exponential complexity, three algorithms are provided with increasing levels of complexity to be applied, depending on the type of analysis performed. In addition, examples are given for each algorithm to show how to apply it in practice.

Section 2 in this paper describes the basics of functional fault modeling. Section 3 defines the concept of precise fault models and how they lead to designing optimum tests. Then, Section 4 presents the general method used to perform the precise fault analysis. This method employs three different algorithms, discussed in Section 5. Finally, Section 6 ends with the conclusions.

2 Basics of FFMs

Functional faults are informally understood as the deviation of the observed memory behavior from the functionally specified one, under a sequence of performed memory operations. Therefore, two basic ingredients

are needed to define any FFM: (1) a sequence of performed memory operations, and (2) a list of corresponding deviations in the observed behavior from the expected one.

Any sequence of performed memory operations (S) can be represented by the following notation:

$$d_{c_1} \ldots d_{c_i} \ldots d_{c_m}\ Od_{c_1} \ldots Od_{c_j} \ldots Od_{c_n}$$

where c_x: cell address used,
O: type of operation on c, $O \in \{w, r\}$,
d: initialization or written data into c, $d \in \{0, 1\}$,
m: number of initializations, and
n: number of operations.

The initialization part is applied to m cells (denoted as c_i), while the operation part is applied to n cells (denoted as c_j). Note that the value of d in rd_{c_j} of the operation part represents the expected value of the read operation, which may be different from the actual read value detected on the output in case of a faulty memory. As an example of the notation, if an operation sequence is denoted by $0_c w1_c r1_c$ then the sequence starts by accessing cell c (which contains a 0) and writing a 1 into it, then reading the written 1.

Once the operation that causes the fault is known, any difference between the observed and expected memory behavior can be denoted by the following notation $<S/F/R>$, referred to as a *fault primitive (FP)* [6]. S describes the operation sequence that sensitizes the fault, F describes the value of 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 $<0_c w1_c/0/->$, which is a transition fault 1 (TF1), $S = 0_c w1_c$ means that cell c is assumed to have the initial value 0, after which a 1 is written into c. The fault effect $F = 0$ indicates that after performing a $w1$ to c, as indicated by S, c remains in state 0. The output of the read operation $R = -$ indicates that S does not end with a read operation. The notation for the FP $<0_c w1_c/0/->$ can be simplified to $<0w1/0/->_c$.

FPs can be classified into different classes, depending on S. Let $\#C$ be the number of *different* memory cells initialized (c_i) or accessed (c_j) in S, and let $\#O$ be the number of operations (w or r) performed in S. For example, if $S = 0_{c_1} 0_{c_2} w1_{c_2}$ then $\#C = 2$ since two cells (c_1 and c_2) are present in S, while $\#O = 1$ since only one operation is performed ($w1$ to c_2).

Depending on $\#C$, FPs can be divided into the following classes:

- If $\#C = 1$ then the sensitized FP is called a *single-cell FP*.
- If $\#C > 1$ then the sensitized FP is called a *coupling FP*. If $\#C = 2$ then it is described as a *two-coupling FP* or a *two-cell FP*. If $\#C = 3$ then it is described as a *3-coupling FP*, etc.

In case an FP is a coupling FP ($\#C > 1$) then one of the cells in the S should be considered as a *victim* (v) while the other cells are considered as *aggressors* (a). In any FP, the described faulty behavior is related to a victim while the aggressors are considered to contribute to the fault.

Depending on $\#O$, FPs can be divided into the following classes:

- If $\#O \leq 1$ then the sensitized FP is called a *static FP*.
- If $\#O > 1$ then the sensitized FP is called a *dynamic FP*. If $\#O = 2$ then it is described as a *2-operation dynamic FP*. If $\#O = 3$ then it is described as a *3-operation dynamic FP*, etc.

Definition 1 A **functional fault model (FFM)** is a non-empty set of fault primitives (FPs).

For example, the *transition fault (TF)* FFM consists of 2 FPs: TF $= \{<0w1/0/->, <1w0/1/->\}$.

3 Precise FPs and optimal tests

In this section, the concept of precise FPs is defined. Then, imprecise FPs are classified into either overspecified or underspecified FPs.

3.1 Definition of precise FPs

Assume that S has been performed on a defective memory, which results in sensitizing a given FP_k. The resulting FP has the form $<S_k/F_k/R_k>$, where S_k is the sequence that sensitizes the fault. The general representation of S_k has the form:

$$d_{c_1} \ldots d_{c_i} \ldots d_{c_m}\ Od_{c_1} \ldots Od_{c_j} \ldots Od_{c_n}$$

For any given S_k, it is desired that all initializations and operations are *necessary* to sensitize the fault, so that if any of them changes, then a different faulty behavior or no faulty behavior results. On the other hand, it is desired that S_k provides a *sufficient* description of the conditions resulting in the fault, so that if other initializations or operations are performed before S_k then the fault would still take place. Therefore, precise FPs can be defined as follows.

Definition 2 An FP = $<S/F/R>$ is said to be **precise** if

1. The FP is not *overspecified*: this means that all d_{c_i} and Od_{c_j} in S are necessary to sensitize the fault. Overspecification results in an increased test time.
2. The FP is not *underspecified*: this means that the d_{c_i} and Od_{c_j} in S are sufficient to sensitize the fault. Underspecification results in incomplete fault coverage.

3.2 Overspecified FPs

An FP is said to be overspecified if some of the initializations d_{c_i} or the operations Od_{c_j} of S are not necessary to sensitize the fault. Consequently, if FP_o is an overspecified FP then there is always a precise FP (FP_p) with initializations and operations that are all necessary to sensitize the fault. If FP_o only has an overspecified part (and no underspecified part) and it is characterized by $\#O_o$ and $\#C_o$, and if FP_p is characterized by $\#O_p$ and $\#C_p$, then one of the following relations is true:

1. $\#C_p < \#C_o$,
2. $\#O_p < \#O_o$, or
3. $\#C_p < \#C_o$ and $\#O_p < \#O_o$.

The following example shows how FPs can be overspecified according to $\#C$.

Example 1 (overspecified in $\#$C) Assume that a defective memory has a static two-coupling fault such that a logic 1 in the aggressor a forces a logic 1 in the victim v. The S causing this fault is 1_a0_v (aggressor in state 1 and victim in state 0) and therefore this fault can be described as $FP_1 = <1_a0_v/1/->$ (aggressor in state 1 flips the victim from 0 to 1). Now assume that, while performing fault analysis on this defective memory, $S = 1_a1_{a'}0_v$ has been performed where in addition to a a second cell is considered as an aggressor a'. If we assume that a' has no influence on the faulty behavior, S will still fail as a result of the state of a thereby sensitizing $FP_2 = <1_a1_{a'}0_v/1/->$. Yet, this FP does not precisely describe the faulty memory since it sets more conditions than necessary (requiring a' to contain 1) to sensitize the fault. □

3.3 Underspecified FPs

An FP is said to be underspecified if some initializations d_{c_i} or operations Od_{c_j}, necessary to sensitize the fault, are not included in S as conditions to sensitize the fault. Consequently, if FP_u is an underspecified FP then there is always a precise FP (FP_p) with initializations and operations sufficient to sensitize the fault. If FP_u only has an underspecified part (and no overspecified part) and it is characterized by $\#O_u$ and $\#C_u$, and if FP_p is characterized by $\#O_p$ and $\#C_p$, then one of the following relations is true:

1. $\#C_p > \#C_u$,
2. $\#O_p > \#O_u$, or
3. $\#C_p > \#C_u$ and $\#O_p > \#O_u$.

FPs can be underspecified due to insufficient initializations or operations in S. For example, note that all cells in a memory do have a state, whether this state is included in S or not. Therefore, an S that does not include all cell initializations needed to sensitize a fault can still result in a fault, simply because the cells happen to be initialized to the states that sensitize the fault.

Example 2 Assume that a defective memory has a static two-coupling fault such that a logic 1 in the aggressor a forces a logic 1 in the victim v. The S casing this fault is 1_a0_v and therefore this fault can be described as $FP_1 = <1_a0_v/1/->$. Now assume that, while performing fault analysis on this defective memory, $S = 0_v$ has been performed in which a was not considered to influence the faulty behavior. If we assume that c accidentally contains a logic 1, S will still fail thereby sensitizing $FP_2 = <0_v/1/->$. Yet this FP does not precisely describe the faulty memory since it sets fewer conditions than necessary (not requiring a to contain 1) to sensitize the fault. □

4 Fault analysis method

This section discusses a method that enables performing precise fault analysis (i.e., enable identification of precise FPs). Identification of precise FPs is important to generate precise FFMs and, eventually, to derive optimal memory tests. The method, shown in Figure 1, has four steps.

First, all possible combinations of relevant operations sequences (S) should be generated in Step 1. Since it is typical for a memory to have millions of cells and since there are infinitely many possible performed operations, it is not practically feasible, nor realistic, to perform all Ss. Instead, in Step 1, the fault analysis will be performed for a given neighborhood consisting of k *relevant cells* ($\{c_1, ..., c_k\}$) and for a given number of operations ($\#O$). This restriction is realistic because it has been shown [7, 8] that a defect only influences a few

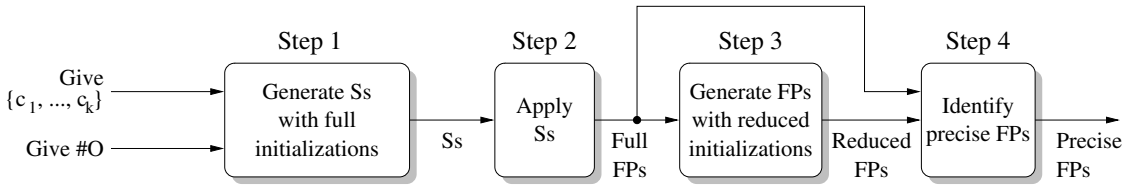

Figure 1. Fault analysis method to generate precise FPs.

cells, and Ss with only a few initializations and operations are needed to sensitize the faults caused by a defect. Ss generated in this step should be fully initialized (i.e., all relevant cells should be initialized to a given value). As an example, assume that only one cell v is considered relevant and that we limit $\#O$ to 1, then the possible Ss are: 0_v, $0_v r 0_v$, $0_v w 0_v$, $0_v w 1_v$, 1_v, $1_v r 1_v$, $1_v w 0_v$ and $1_v w 1_v$.

In Step 2, the total faulty behavior of the memory should be analyzed by applying all Ss generated in Step 1 to the memory. Each failing S is used to identify an FP = $<S/F/R>$, a process that results in a number of FPs where S has a full initialization of all relevant cells (such FPs are referred to as *full FPs*). As an example, applying the fully initialized Ss generated in Step 1 on a defective memory might result in the the following full FPs: $<0_v r 0_v/1/1>$ and $<0_v w 0_v/1/->$. Note that the term full FPs refers to the fact that all FPs resulting from Step 2 have all accessed cells initialized.

The resulting full FPs are taken as input for Step 3 which generates all possible FPs where S has a *reduced initialization part* (referred to as *reduced FPs*). In this step, a new set of FPs is generated with all possible permutations of initializations of S in each full FP. This set of FPs will serve in Step 4 to inspect whether initialization are actually necessary in the FP description. As an example, the full FP = $<0_v w 0_v/1/->$ from Step 2 generates the reduced FP $<w 0_v/1/->$ where the initialization 0_v is removed.

Finally, all FPs (full and reduced) are presented to Step 4, where an algorithm is used to identify the *precise FPs*. In Section 5, algorithms used in Step 4 to identify precise FPs are discussed in detail. The following example shows how to apply Steps 1, 2 and 3 of the fault analysis method shown in Figure 1. Examples of Step 4 are given in Section 5.

Example 3 As input to Step 1, a set of relevant cells ($\{c_1, ..., c_k\}$) and the number of operations ($\#O$) should be given. If the cells are chosen to be $\{a_1, a_2, v\}$ and $\#O = 0$ then Step 1 results in the following 8 Ss: $0_{a_1} 0_{a_2} 0_v$, $0_{a_1} 0_{a_2} 1_v$, $0_{a_1} 1_{a_2} 0_v$, $0_{a_1} 1_{a_2} 1_v$, $1_{a_1} 0_{a_2} 0_v$, $1_{a_1} 0_{a_2} 1_v$, $1_{a_1} 1_{a_2} 0_v$, and $1_{a_1} 1_{a_2} 1_v$.

Step 2 applies these 8 Ss to the memory under investigation and represents the failing Ss of them as FPs. We assume that the following FP$_f$ is sensitized $<0_{a_1} 0_{a_2} 1_v/0/->$, which is called a full FP$_f$ since it contains all initializations of relevant cells. This full FP$_f$ is taken by Step 3 to generate the FP$_r$s with reduced initializations. Step 3 gives the following 3 reduced FP$_r$s: $<1_v/0/->$, $<0_{a_1} 1_v/0/->$, and $<0_{a_2} 1_v/0/->$. All resulting 4 FPs (3 FP$_r$s from Step 3 and the full FP$_f$ from Step 2) are forwarded to Step 4 to inspect which FPs of them are precise. Later in the paper these 4 FPs are used, therefore we denote them here as FP$_1$ = $<1_v/0/->$, FP$_2$ = $<0_{a_1} 1_v/0/->$, FP$_3$ = $<0_{a_2} 1_v/0/->$, and FP$_4$ = $<0_{a_1} 0_{a_2} 1_v/0/->$. □

Depending on the provided neighborhood of relevant cells ($\{c_1, ..., c_k\}$) and the number of operations ($\#O$), the Ss generated in Step 1 of the fault analysis method can be classified into 3 neighborhood types:

1. *Static neighborhoods*: no operations are performed (neither on the aggressors nor on the victim)
2. *Active neighborhoods*: operations are only performed on the aggressors
3. *General neighborhoods*: operations are performed on the victim as well as the aggressors

In the next section, an algorithm is given for the static neighborhood.

5 Precise identification algorithm

This section tackles the precise FP identification problem. Due to the limited space, only the algorithm for static neighborhoods is shown in this paper.

Static neighborhoods mean that the used S has no performed operations ($\#O = 0$); hence, the cells in the performed S are only observed. The general notation of an S of this type can be described as:

$$S_s = d_v \; d_{a_1} \; ... \; d_{a_i} \; ... \; d_{a_{(m-1)}}$$

where one of the cells is considered as a victim (v) while the others ($m-1$) cells are aggressors. This S does not

contain read operations and therefore it results in a fault described by FP$_s$ = $<S_s/F/->$ where $R = -$. The problem is to establish whether FP$_s$ is precise.

According to Definition 2, all initializations of S in a precise FP should be necessary and sufficient to sensitize the fault. For the special case of static neighborhoods, it should be shown that all initializations included in S_s are indeed needed to sensitize the fault and, at the same time, none of the other $(k-m)$ relevant cells in the memory (in which k cells are considered relevant to the faulty behavior; see Figure 1) participate in the fault. In order to give an algorithm to do this, we need the following definition.

Definition 3 Given any general sensitizing operation sequence S_m, with an initialization part involving m cells ($0 \leq k \leq m$), a **memory permutation** of S is defined as an S' with the same operation part as S, and an *extended* initialization part in which the remaining $(k-m)$ *relevant* memory cells are initialized to a given value.
$S' = d_{c_1}...d_{c_i}...d_{c_m}\ d_{c_{m+1}}...d_{c_k}\ Od_{c_1}...\ Od_{c_j}...Od_{c_n}$

In order to show the necessity of a given initialization d_{c_i} in S ($0 \leq i \leq m$), the following procedure should be performed. First, the memory behavior should be inspected when the initialization data d_i for cell c_i is inverted to $\overline{d_i}$. If the data inversion results in proper behavior, then the initialization d_{c_i} is necessary. However, if the fault remains then this does not yet mean that d_{c_i} is unnecessary, since it is possible that d_{c_i} does contribute to the faulty behavior in collaboration with other initializations in S_s. Therefore, d_{c_i} is only considered unnecessary if for all (2^{k-m}) memory permutations of S_s, using $\overline{d}_{c_i}$ still results in a faulty behavior. This means that d_{c_i} is considered necessary if there is at least one memory permutation of S_s that results in no faulty behavior when d_{c_i} is replaced by $\overline{d}_{c_i}$ in S_s. This procedure should be performed $(m-1)$ times for each initialization in S_s.

In order to show that none of the $(k-m)$ relevant cells (not included in S_s) influence the FP, the following should be done. All memory permutations of S_s should be performed and the memory behavior is inspected. If any memory permutation of S_s results in no faulty behavior, then there is a necessary initialization not included in S_s. This means that the initializations in S_s are only considered sufficient if none of the memory permutations of S_s changes the faulty behavior.

The following algorithm gives two conditions which the faulty behavior of the memory should satisfy in order for FP$_s$ to be precise.

Algorithm 1 If performing S_s results in sensitizing FP$_s$ = $<S_s/F/->$, then FP$_s$ is precise if:

1. **Check initializations are necessary:** For each of the $(m-1)$ aggressors in S_s the following should be inspected. If the initialization value is inverted then at least one of the 2^{k-m} memory permutations given by $S' = d_v d_{a_1}\ ...\ \overline{d}_{a_i}\ ...\ d_{a_{(m-1)}} d_{a_m}\ ...\ d_{a_{(k-1)}}$, where $(d_m, ..., d_{k-1}) \in \{0,1\}^{k-m}$, does not result in a fault (i.e., results in *proper* memory behavior).

2. **Check initializations are sufficient:** Performing all possible 2^{k-m} memory permutations for S_s given by $S' = d_v d_{a_1}\ ...\ d_{a_{(m-1)}} d_{a_m}\ ...\ d_{a_{(k-1)}}$, where $(d_m, ..., d_{k-1}) \in \{0,1\}^{k-m}$, always results in $<S'/F/->$.

This algorithm is not trivial in the way it assigns fault primitives to the observed faulty behavior. To clarify the algorithm, the following example is given, where the algorithm results in an FP assignment that is intuitively expected.

Example 4 This example is based on the faulty behavior resulting from a defect injected into a DRAM design. The defect is modeled at the electrical level by two resistors from the victim to two aggressors, as shown in Figure 2. All three cells, the victim and the two aggressors, are located on the true bit line (BT). Let us consider the way the state of the victim is affected by the states of the two aggressors. Since we are only considering the states of the cells while no operations are performed, the three cells remain isolated from the rest of the memory. Therefore, it is safe to say that only these three cells are relevant ($k = 3$) and the rest of memory does not influence this faulty behavior. Assume that the victim is initialized to logic 1 and that the state of the victim is inspected after a time period $\tau < T_{prech}$, where T_{prech} is the cell array precharge time. It is possible to choose a defect resistance value (R_{def}), such that the aggressors can only pull the victim down if both of them are set to 0, otherwise the victim remains 1 after τ. An overview of the way the states of the aggressors affect the state of the victim after a time period τ is given in Table 1.

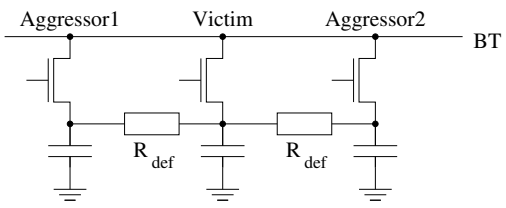

Figure 2. Electrical model of a DRAM with a defect connecting three memory cells together.

Table 1. Effect of a_1 and a_2 on a stored 1 in v as discussed in Example 4.

a_1	a_2	v	v after τ
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	1

Applying Steps 1, 2, and 3 of the analysis method in Figure 1 on this defective memory with the set $\{a_1, a_2, v\}$ and $\#O = 0$ as input, gives the following 4 FPs: $FP_1 = <1_v/0/->$, $FP_2 = <0_{a_1}1_v/0/->$, $FP_3 = <0_{a_2}1_v/0/->$, and $FP_4 = <0_{a_1}0_{a_2}1_v/0/->$. (See Example 3)

One would expect this faulty behavior to be represented by a three-cell FP as in $<0_{a_1}0_{a_2}1_v/0/->$. Applying Condition 2 of Algorithm 1 to FP_1 (with $S = 1_v$) shows that the following memory permutation $S' = 1_{a_1}1_{a_2}1_v$ results in a proper memory behavior. This means that the initializations in FP_1 are insufficient and that FP_1 is underspecified. In the same way, FP_2 and FP_3 are shown to be underspecified. Finally, Algorithm 1 is applied to FP_4. Inverting the initialization of a_1 or a_2 in FP_4 results in a proper memory behavior, thereby validating Condition 1. In addition, all relevant cells are initialized in FP_4 meaning the initializations are sufficient, thereby validating Condition 2. In conclusion, according to Algorithm 1 the faulty behavior observed can be described by the three-cell $FP_4 = <0_{a_1}0_{a_2}1_v/0/->$, as it is intuitively expected. □

6 Conclusions

In this paper, an automatic fault analysis method has been presented that precisely characterizes the faulty behavior of a memory. Two types of imprecision in FPs have been identified—overspecified and underspecified FPs. It has been shown that overspecified FPs result in generating inefficient tests, while underspecified FPs result in tests with incomplete fault coverage. The proposed fault analysis method is general and does not refer to electrical data to specify the type of fault. The method results in a set of precise FPs that are neither overspecified nor underspecified. This enables deriving optimal tests for the memory under analysis, resulting in complete fault coverage with minimal test time.

References

[1] A.J. van de Goor and B. Smit, "Generating March Tests Automatically," *in Proc. IEEE Int'l Test Conf.*, 1994, pp. 870–878.

[2] K. Zarrineh, S.J. Upadhyaya and S. Chakravarty, "A New Framework for Generating Optimal March Tests for Memory Arrays," *in Proc. IEEE Int'l Test Conf.*, 1998, pp. 73–82.

[3] V.N. Yarmolik, Y.V. Kilmets and A.J. van de Goor, "Diagnostic RAM Tests," *in Automatic Control and Computer Science*, vol. 31, no. 2, 1997, pp. 11–16.

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

[5] D. Niggermeyer, M. Redeker and 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.

[6] 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 Comp.*, vol. 52, no. 3, 2003, pp. 293-309.

[7] Z. Al-Ars, *DRAM Fault Analysis and Test Generation*, PhD thesis, Delft Univ. of Technology, Delft, the Netherlands, 2005, http://ce.et.tudelft.nl/~zaid/

[8] S. Hamdioui, *Testing Static Random Access Memories: Defects, Fault Models and Test Patterns*, Kluwer Academic Publishers, Boston, MA, 2004.

[9] S. Naik, F. Agricola and W. Maly, "Failure Analysis of High Density CMOS SRAMs," *in IEEE Design and Test of Computers*, vol. 10, no. 2, 1993, pp. 13–23.

[10] Z. Al-Ars and A.J. van de Goor, "Modeling Techniques and Testing for Partial Faults in Memory Devices," *in Proc. Design, Automation and Test in Europe*, 2002, pp. 89–93.