Precise Identification of Memory Faults Using Electrical Simulation

Zaid Al-Ars Said Hamdioui Georgi Gaydadjiev
Laboratory of Computer Engineering, 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: *Recently, a framework describing the space of all fault models has been established. Subsequently, it has been shown that many new faults of that space do exist. Gradually, The number and complexity of observed memory fault models has been gradually increasing. As a result, it has become increasingly difficult to identify the precise functional fault models that a memory suffers from. This paper shows that there are two types of possible imprecision in describing faults: underspecification, which leads to tests with insufficient fault coverage, and overspecification, which leads to time-inefficient tests. A general method is presented to analyze faulty memory behavior based on electrical simulation and map it precisely onto the corresponding fault models, which makes it possible to generate time-optimal tests with optimal fault coverage.*

Keywords: *memory fault models, precise faults, fault identification, test time, fault coverage.*

1 Introduction

With every generation of memory devices, new more complex fabrication technologies and design techniques are being used. As a result, new types of defect mechanisms are being introduced into the memory, which in turn cause previously unknown types of faulty behavior to take place. Depending on the type of the memory and its field of application, it is commercially desirable to have an extremely low escape rate. This requires memory tests with a high fault coverage to detect all the faults taking place. However, these tests should also be as short as possible to keep the cost of memory testing low.

In order to achieve both memory test requirements of high fault coverage and short test time, there is a need to cover *exactly* the faults taking place in the memory and removing the test parts that detect both unrealistic faults and/or other faults not observed in the memory under test. This requires the *precise* knowledge of the fault models exhibited by the memory faulty behavior. This knowledge can be acquired using defect injection and electrical simulation, then it can be used to generate a corresponding set of optimal memory tests [Zarrineh98].

Presently, simulation-based memory fault analysis methods do not apply standard algorithms to identify the observed faulty behavior [Al-Ars05]. They usually refer to electrical information (voltages and currents) to identify the presence of a given type of fault model in an ad hoc way [Dekker90, Naik93]. This leads to the imprecise specification of the behavior, causing increased test time and/or incomplete fault coverage. Considering the fact that a systematic way of describing all possible memory faults exist [Al-Ars03], together with the fact that many new faults have recently been uncovered, a precise method of identifying the appropriate fault model is essential. This is true since fault models may suffer from underspecification, leading to tests with insufficient fault coverage, or may suffer from overspecification, leading to time-inefficient tests.

Previous work have shown that it is possible to give a precise identification of fault models caused by the initializations of memory cells only, such as state faults and state coupling faults [Al-Ars07]. This paper provides a general fault identification method, applicable to all faults involving both initializations as well as operations. The method only uses *functional information* (cell contents and memory outputs) and does not refer to electrical data to carry out the identification. This makes the method independent of the specific electrical implementation of the memory, making it applicable for both DRAMs and SRAMs.

In Section 2, the paper starts with the definition of fault models in memories. Section 3 identifies overspecification and underspecification as the causes of imprecision in fault modeling. Section 4 presents a method used to perform the precise fault analysis, using the algorithm discussed in Section 5. Section 6 ends with the conclusions.

2 Basics of FFMs

Any memory operation sequence expected to result in a faulty behavior can be represented by the following notation:

$$d_{c_1} \ldots d_{c_i} \ldots d_{c_m}\ Od_{c_1} \ldots Od_{c_j} \ldots Od_{c_n} \qquad (1)$$

where c: 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 value observed at 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.

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)* [Al-Ars03]. 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 $<0w1/0/->_c$ or to $<0w1/0/->$.

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 FP sensitized by the corresponding S is called a *single-cell FP*.

- If $\#C > 1$ then the FP sensitized by the corresponding S 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 FP sensitized by the corresponding S is called a *static FP*.

- If $\#O > 1$ then the FP sensitized by the corresponding S 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.

Clearly, a hierarchy in S results in a hierarchy in FPs. Figures 1(a) and (b) represent the two different types of hierarchies in FPs defined according to $\#C$ and $\#O$, respectively. As shown in the figure, two-coupling FPs are higher in the hierarchy than single-cell FPs, and the larger the number of coupled cells the higher the hierarchical level a fault primitive has. In the same way, a dynamic FP is higher in the hierarchy than a static FP, and the larger the number of operations of a dynamic FP the higher it becomes in the hierarchy. A higher hierarchical level involves more cells and/or more operations, hence has a higher test cost.

The notion of FPs makes it possible to give a precise definition of an FFM as understood for memory devices. This definition is presented next [Al-Ars03].

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

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

3 Imprecision in FPs

As mentioned in Section 1, not all sensitized FPs are necessarily precise in the way they describe the faulty behavior of a memory. In this section, the idea of precise FPs is defined and the reasons that make some sensitized FPs imprecise are analyzed.

3.1 Definition of precise FPs

Assume that S has been performed on a defective memory, which results in sensitizing a given FP. The resulting FP has the form $<S/F/R>$, where S is the sequence that sensitizes the fault. The general representation of S has the form shown in Equation 1.

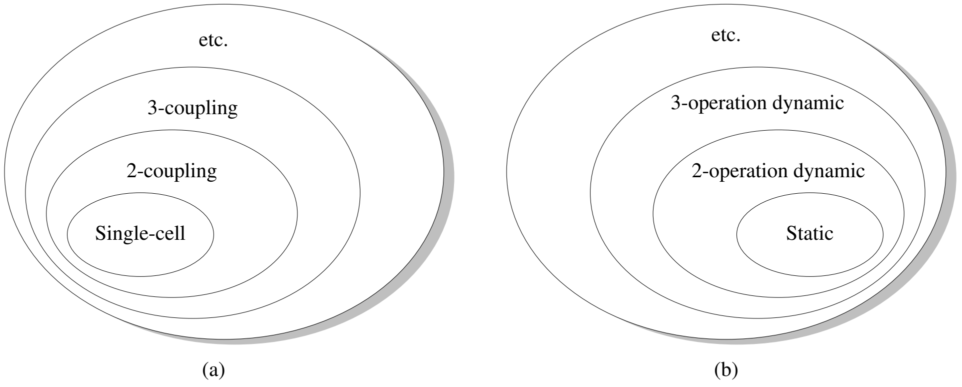

Figure 1. Hierarchical organization of fault primitive classes defined according to (a) $\#C$ and (b) $\#O$.

If a given S is said to sensitize a fault, then it is desired that each component of S is *necessary* and that S as a whole is *sufficient* to sensitize the fault. In other words, it is desired that all m initializations and all n operations contained in S are needed to sensitize the fault, so that if any of them changes, it would result in a different faulty behavior or in no faulty behavior at all. On the other hand, it is desired that S provides a good enough description of the conditions resulting in the fault, so that if other initializations or operations are performed before S then the fault would still take place. Based on the discussion above, 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.

An FP that does not satisfy these conditions is called **imprecise**. FPs can be imprecise as a result of their failure to satisfy Condition 1, Condition 2 or both conditions of Definition 2.

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 $\#O_p < \#O_o$.

These relations show how using overspecified FPs to describe the faulty behavior result in constructing inefficient memory tests. The lower the number of accessed cells ($\#C$) an FP has, the less complex the memory test required to detect it. In the same way, the lower the number of performed operations ($\#O$) an FP has, the shorter the test required to detect it.

Overspecified FPs are a direct consequence of the hierarchical organization of FPs (see Figure 1), since an FP of a given hierarchical level can be covered by other FPs of a higher hierarchical level. The following example shows how FPs can be overspecified based on $\#O$.

Example 1 (overspecified in $\#$O) Similar to the previous example, assume that a defective memory has a static two-coupling fault that can be described as FP$_1$ = $<1_a0_v/1/->$. Now assume that, while performing fault analysis on this defective memory, S = $1_a0_vw1_a$ has been performed where not only the state of a is set to 1, but also a write 1 operation is performed on a. If we assume that the write 1 operation has no effect on the faulty behavior, S will still fail as a result of the state of a, thereby sensitizing FP$_2$ = $<1_a0_vw1_a/1/->$. Again, this FP does not precisely describe the faulty memory since it sets more conditions than necessary (requiring a write 1 operation on a) to sensitize the fault.

From a testing point of view, FP$_1$ can be detected using the march test $\{\Uparrow(w0);\ \Uparrow(r0,w1);\ \Downarrow(w0);\ \Downarrow(r0,w1)\}$. FP$_2$, on the other hand, requires an additional write operation thereby requiring the march test $\{\Uparrow(w0);\ \Uparrow(r0,w1,w1);\ \Downarrow(w0);\ \Downarrow(r0,w1,w1)\}$. Here, the overspecified FP increased the test time by 33%. □

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 in S. This is true since all cells in a memory hold their previous states, 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 fail, simply because the cells happen to be initialized to the states that sensitize the fault as a result of previously performed memory operations.

FPs can also be underspecified due to insufficient operations in S. This is true since the past history of operations on a defective memory leaves a trail of voltages in cells and possibly on bit lines, word lines and data lines. Therefore, an S that does not include all operations needed to sensitize a fault can still result in a fault, simply because the needed operations happen to be performed in past history of the memory. This behavior is strongly related to a special type of faults called partial faults, which is analyzed in more depth in the literature [Al-Ars05].

Underspecified FPs result in constructing memory tests that are, in general, unable to detect the targeted faulty behavior, as can be seen in the following example.

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 causing 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.

From a testing point of view, detecting FP$_1$ requires using the march test $\{\Uparrow(w0); \Uparrow(r0, w1); \Downarrow(w0); \Downarrow(r0, w1)\}$. On the other hand, the underspecified FP$_2$ can be detected using the march test $\{\Uparrow(w0); \Uparrow(r1)\}$, which is clearly unable to detect FP$_1$. □

4 Fault analysis method

This section discusses a method that enables performing precise fault analysis (i.e., enables identification of precise FPs) using defect injection and electrical simulation. Identification of precise FPs is important to generate precise FFMs and, eventually, to derive optimal memory tests. The method, shown in Figure 2, 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 [Al-Ars03] that a defect only influences a few 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_vr0_v, 0_vw0_v, 0_vw1_v, 1_v, 1_vr1_v, 1_vw0_v and 1_vw1_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 following full FPs: $<0_vr0_v/1/1>$ and $<0_vw0_v/1/->$. Note that the term full FPs refers to the fact that all FPs resulting from Step 2 have all accessed cell 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 = $<0_vw0_v/1/->$ from Step 2 generates the reduced FP $<w0_v/1/->$ 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

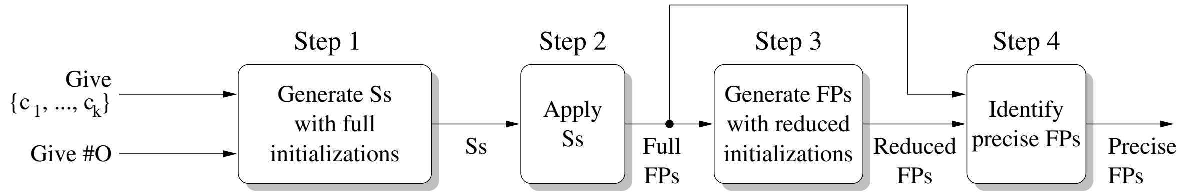

Figure 2. Fault analysis method to generate precise FPs.

how to apply Steps 1, 2 and 3 of the fault analysis method shown in Figure 2. 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$_4$ = $<0_{a_1}0_{a_2}1_v/0/->$. □

5 Precise identification algorithm

The general notation of S in Equation 1 can be reformulated as:

$$S = d_v d_{a_1}...d_{a_i}...d_{a_{(m-1)}}\ Od_{c_1}...Od_{c_j}...Od_{c_n}$$

where the cells (c_j) in the operation part of S can be taken as the victim as well as an aggressor. Since this S may end with a read operation performed on the victim, it results in a fault described by FP = $<S/F/R>$ where $R \in \{0, 1, -\}$. The problem is to establish whether FP is precise. 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 ($0 \le k \le m$), 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 ($0 \le i \le m$), 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. Therefore, d_{c_i} is only considered unnecessary if for all (2^{k-m}) memory permutations of 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 that results in no faulty behavior when d_{c_i} is replaced by $\overline{d}_{c_i}$ in S. This procedure should be performed $(m - 1)$ times for each initialization in S.

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

Showing that all the operations are necessary to sensitize the fault is done as follows. All possible proper subsets of the operations included in S (there are $(2^n - 1)$ of them) should be applied to the memory. If any of these operation subsets shows the same faulty behavior as S under all possible memory permutations, then some operations in S are unnecessary. The operations in S are considered all necessary only if all operation subsets show a different faulty behavior (different F and/or different R) for at least one memory permutation.

The following algorithm gives three conditions which the faulty behavior of the memory should satisfy in order for FP to be precise. In the algorithm, k is the number of relevant cells in the memory, m is the number of initializations in S, while n is the number of operations ($\#O$) in the S.

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

1. **Check operations are necessary:** For each of the $(2^n - 1)$ Ss given by every possible proper subset of the operations performed in S, there is at least one memory permutation that results in FP$'$ = $<S'/F'/R'>$ such that $F' \neq F$ and/or $R' \neq R$.

2. **Check initializations are necessary:** For each initialization in S, the following should be inspected. If the initialization is inverted then at least one of the 2^{k-m} memory permutations given by $S' = d_v d_{a_1} ... \overline{d}_{a_i} ... d_{a_{(m-1)}} d_{a_m} ... d_{a_{(k-1)}} Od_{c_1}$ $...Od_{c_j}...Od_{c_n}$, where $(d_m, ..., d_{k-1}) \in \{0,1\}^{k-m}$, results in FP$'$ = $<S'/F'/R'>$ such that $F' \neq F$ and/or $R' \neq R$.

3. **Check initializations are sufficient:** Performing all possible 2^{k-m} memory permutations for S given by $S' = d_v d_{a_1} ... d_{a_{(m-1)}} d_{a_m} ... d_{a_{(k-1)}}$ $Od_{c_1}...Od_{c_j}...Od_{c_n}$, where $(d_m, ..., d_{k-1}) \in \{0,1\}^{k-m}$, always results in $<S'/F/R>$.

The following example illustrates how Algorithm 1 can be applied to check for precise FPs. The example uses a defect within a single memory cell, which means that the faulty behavior does not depend on other memory cells.

Example 4 The example is based on the faulty behavior resulting from a defect injected within a DRAM cell, as shown in Figure 3. The faulty behavior resulting from this defect can be considered restricted to the defective cell only. Assume that R_{def} has a high enough value to disconnect the cell from the bit line. This means that write operations fail to set the state of the cell, and that read operations detect the precharge state of bit lines rather that the state of the cell.

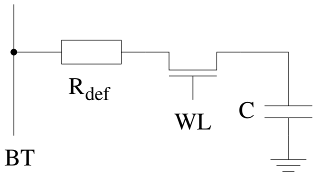

Figure 3. Model of a DRAM cell isolated by a defect from the bit line.

Assume that the cell is initialized to 0, and that it is connected to the true bit line which is precharged to the high voltage level. As a result, the following FPs would be sensitized: FP$_1$ = $<0_v w1_v/0/->$, FP$_2$ = $<0_v r0_v/0/1>$, and FP$_3$ = $<0_v w0_v r0_v/0/1>$. Since the initialization of the cell does not result in a fault, FP$_1$ and FP$_2$ are considered precise. On the other hand, FP$_3$ is not precise because the operation subset $S = 0_v r0_v$ results in the same faulty behavior as FP$_3$. □

6 Conclusions

This paper presented a new fault analysis method to precisely map the faulty behavior of a memory observed during defect injection and electrical simulation to a set of FPs. The method performs all relevant sensitizing operation sequences on the memory and identifies whether the resulting FPs are precise. The resulting precise FPs are not underspecified, which means that they are sufficient to sensitizes the fault. At the same time, the resulting precise FPs are not overspecified, which means that they are necessary to sensitize the fault and contain no redundancies. Identifying the set of precise FPs for a memory leads to deriving memory tests with high fault coverage and optimal test time.

References

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

[Al-Ars05] 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/

[Al-Ars07] Z. Al-Ars and S. Hamdioui, "Automatic Analysis of Memory Faulty Behavior in Defective Memories," *in Proc. IEEE Int'l Conf. on Design and Technology of Integrated Systems in Nanoscale Era*, 2007.

[Dekker90] R. Dekker, F. Beenker and L. Thijssen, "A Realistic Fault Model and Test Algorithms for Static Random Access Memories," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 9, no. 6, 1990, pp. 567–572.

[Naik93] 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.

[Zarrineh98] 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.