Using Linear Tests for Transient Faults in DRAMs

Zaid Al-Ars Said Hamdioui Georgi Gaydadjiev

Delft University of Technology, Faculty of EE, Mathematics and CS

Laboratory of Computer Engineering, Mekelweg 4, 2628 CD Delft, The Netherlands

E-mail: z.e.al-ars@ewi.tudelft.nl

Abstract: *Recent developments in the theory of DRAM fault modeling have identified the space of tests needed to detect all DRAM faults. Tests developed to target transient DRAM faults are very time consuming, as they have a quadratic dependence on the number of tested memory cells. This paper presents techniques to reduce the complexity of these tests. The paper also introduces a reduced test, with linear dependency on the number of cells, that detect all realistic transient faults in DRAMs.*

1 Introduction

DRAM test development has commonly been an ad hoc activity, where large numbers of tests are performed on a representative sample and those tests are chosen that result in the best fault coverage [10]. This approach results in a long and costly test development time, and eventually leads up with a non-optimal test flow [6]. Recently, a theoretical framework of DRAM tests was proposed [5], based on Spice simulations of DRAM models, that is able to detect all possible DRAM faults [4]. Some of the proposed tests, however, are rather complex, such that they scale super-linearly with the number of cells in the memory.

Transient faults are DRAM faults that get temporarily sensitized, and subsequently correct themselves after a limited amount of time [2]. In order to detect these faults properly, the detecting operations should follow the sensitizing operations directly without any delay, before the fault gets corrected. This detection requirement results in a quadratic $O(n^2)$ dependency of test complexity on the number of memory cells. Such test complexity requires a rather long test application time on current day high-density DRAMs, that render them impractical for a high-volume manufacturing test environment.

This paper tackles the high complexity issue of DRAM tests generated to detect transient faults. The paper suggests a number of optimization techniques to reduce test complexity and limit test application time. The concept of physical locality of defects is used, which states that a given defect can only cause a fault in a number of adjacent cells, rather than cells located far apart. The used optimization techniques reduce the test complexity from quadratic to linear, making it suitable for industrial application.

This paper is organized as follows. Section 2 lists the different fault primitives to be used in this paper, followed by a description of the classes of DRAM faults in Section 3. Section 4 derives the general tests needed to detect all transient faults and identifies their complexity, while Section 5 optimizes these tests to limit the needed test time. Finally, Section 6 ends with the conclusions.

2 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. The most important FFM classes are single-cell static FFMs and two-cell static 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), deceptive read destructive fault (DRDF) [1].

Table 1. Single-cell static FFMs and their corresponding 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

Two-cell static FFMs consist of FPs sensitized by performing at most one operation while considering the faulty effect of two cells. Such FPs can be represented as $<S_a; S_v/F/R>$, where S_a is the sequence performed on the aggressor (a) and S_v is the sequence performed on the victim (v). Table 2 lists all two-cell static FFMs and their corresponding FPs. In total, there are 7 different types of two-cell static FFMs: state coupling fault (CFst), disturb coupling fault (CFds), transition coupling fault (CFtr), write destructive coupling fault (CFwd), read destructive coupling fault (CFrd), incorrect read coupling fault (CFir), and deceptive read destructive coupling fault (CFdrd).

Table 2. Two-cell static FFMs and their FPs ($x, y \in \{0, 1\}$).

#	Fault	FP	Name
1	CFst	$<0; 0/1/->$, $<0; 1/0/->$	State coupling
		$<1; 1/0/->$, $<1; 0/1/->$	fault
2	CFds	$<xwy; 0/1/->$, $<xwy; 1/0/->$	Disturb coupling
		$<xrx; 0/1/->$, $<xrx; 1/0/->$	fault
3	CFtr	$<0; 0w1/0/->$, $<0; 1w0/1/->$	Transition coupling
		$<1; 0w1/0/->$, $<1; 1w0/1/->$	fault
4	CFwd	$<0; 0w0/1/->$, $<0; 1w1/0/->$	Write destructive
		$<1; 0w0/1/->$, $<1; 1w1/0/->$	coupling fault
5	CFrd	$<0; 0r0/1/1>$, $<0; 1r1/0/0>$	Read destructive
		$<1; 0r0/1/1>$, $<1; 1r1/0/0>$	coupling fault
6	CFir	$<0; 0r0/0/1>$, $<0; 1r1/1/0>$	Incorrect read
		$<1; 0r0/0/1>$, $<1; 1r1/1/0>$	coupling fault
7	CFdrd	$<0; 0r0/1/0>$, $<0; 1r1/0/1>$	Deceptive read
		$<1; 0r0/1/0>$, $<1; 1r1/0/1>$	destructive CF

3 DRAM-specific faults

DRAM faults can either be attributed to leakage currents (resulting in time dependent faults), or to improperly set voltages (resulting in voltage dependent faults) [4]. Figure 1 shows a summary of DRAM-specific faults.

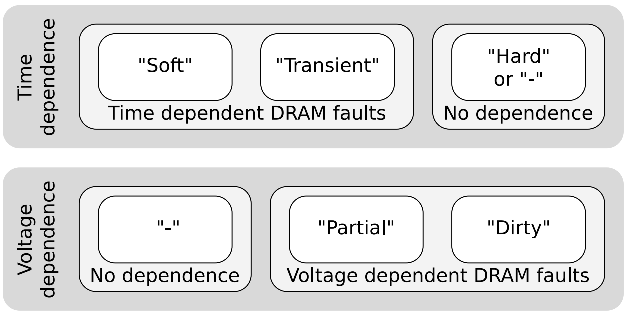

Figure 1. Summary of the space of DRAM-specific faults.

3.1 Time dependent faults

Time dependent faults are caused by leakage currents in faulty cells [8]. Time dependence divides all faults into three classes: soft, transient and hard.

Soft faults—Soft faults (s) only become detectable after some time from their sensitization. These faults can be tested for by adding a *delay* within the test, as it is the case for the *data retention fault*, for example [7]. Soft faults are caused by writing weak voltages into memory cells, that soon get depleted by naturally occurring leakage currents. Soft faults are represented as sFP $=<S_T/F/R>$, where S has an added time parameter T to indicate that some time should elapse before full sensitization. The open defect in Figure 2(a) shows an open that may cause soft faults in a DRAM cell. If the open defect has an intermediate resistance value that is not too high (causing hard faults) and not too low (not causing a fault at all), write operations store a *weak voltage* into the cell. If leakage opposes the weak voltage, the stored information gets lost over time.

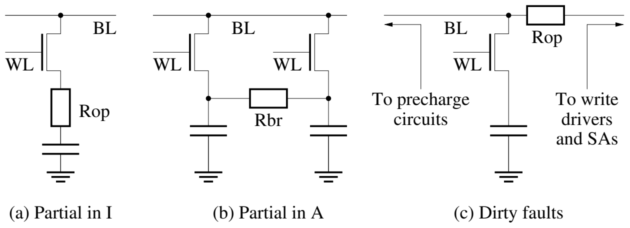

Figure 2. Defects causing (a) $\mathrm{p_i}$, (b) $\mathrm{p_a}$, and (c) causing dirty faults.

Transient faults—Transient faults (t) are memory faults that do not remain sensitized indefinitely, but tend to correct themselves after a period of time. Transient faults are tested for by performing all the operations in the fault in *back-to-back mode* directly after each other, and directly following them by a read. The DRAM open in Figure 2(a) may cause transient faults. For a specific range of R_{op} values, write operations set a faulty voltage within the cell that is not strong enough to qualify as a hard fault. If leakage tends to correct the weak faulty voltage, the stored voltage gets corrected over time. Transient faults are represented as tFP $=<\underline{S}/F_L/R>$, where the underline below S means that the operations in S should be performed in back-to-back mode. Furthermore, F has an added time parameter L (*life time*) to indicate that these faults are time limited. An underline below operations implies that the operations have to be performed after each other within one march element. For example, if $S = w1\underline{w0}$ then the detection condition should be $\Updownarrow$(..., $w1$, $\underline{w0}$, ...).

Hard faults—Identifying a fault as being hard ("h" or "-") indicates that it is neither soft nor transient (i.e., it is insensitive to time). All the generic faults described in Section 2 are hard faults.

3.2 Voltage dependent faults

Operations performed on a defective DRAM may set improper voltage levels on memory nodes, thereby causing two types of DRAM faults: partial faults and dirty faults.

Partial faults—Partial faults (p) are faults that can only be sensitized when a specific memory operation is successively repeated a number of times, either to properly initialize the faulty cell (*partial faults during initialization* p_{i}), or to properly sensitize the fault in the cell (*partial faults during fault sensitization or activation* p_{a}). Figure 2(a) shows an example of an open (R_{op}) in the cell, causing $\mathrm{p_i}$. R_{op} prevents fully initializing the cell to the required voltage with only one operation, which means that full initialization requires repeating the operation a number of times. Figure 2(b) shows an example of a bridge (R_{br}) between two cells, causing $\mathrm{p_a}$. These faults are modeled by performing an operation Ox an h (or *hammer*) number of times. For example, if $<xOy/F/R>$ becomes partial during initialization $\mathrm{p_i}$, it should be modeled as $\mathrm{p_i}$FP $=<x^hOy/F/R>$.

Dirty faults—Dirty faults (d) assume that after proper initialization or sensitization, the state of the memory (voltages on the BLs, the WLs, or in data buffers) is corrupted, such that subsequent detection is prevented. In order to ensure detectability, additional operations (so called *completing operations*) must be performed to correct the corrupted state of the memory. Figure 2(c) shows an example of an open defect (R_{op}) on the BL that causes dirty faults. This defect disconnects memory cells from the write drivers, which prevents the memory from writing the cells. This defect also prevents properly precharging the BL. As a result, a $w0$ operation that fails to write 0 in the cell ends up preconditioning the BL to properly sense a 0, thereby causing a dirty fault. These fault are modeled by the introduction of completing operations to the FP. Detectability of all known dirty faults can be ensured using a completing write operation with data opposite to the data in the victim, performed to a cell different from the victim but positioned on the same BL pair (i.e., dFP $=<xO_vy[w_b\overline{y}]/\overline{y}/->_{b,v\in \mathrm{BL}}$).

3.3 Realistic space of DRAM faults

Any generic memory fault, described in Section 2, can represent a DRAM-specific fault by adding DRAM-specific fault attributes to it. First, there are voltage dependent attributes: partial (p), dirty (d), or neither (-). Second, there are time dependent attributes: hard (h or -), soft (s) and transient (t). Furthermore, the partial attribute can either be initialization related ($\mathrm{p_i}$), or activation (or sensitization) related ($\mathrm{p_a}$).

Based on a detailed analysis of the characteristics of these faults, the full realistic space of DRAM faults can be constructed for singe-cell faults, as well as two-cell faults [3, 4].

$$\begin{array}{c}\text{Single-cell}\\\text{fault}\end{array} = \left\{\begin{array}{c}\text{-}\\ \mathrm{p_i}\\ \mathrm{d}\\ \mathrm{p_id}\end{array}\right\}\left\{\begin{array}{c}\text{h or -}\\ \mathrm{s}\\ \mathrm{t}\end{array}\right\}\mathrm{FP} \qquad (1)$$

$$\begin{array}{c}\text{Two-cell}\\\text{fault}\end{array} = \left\{\begin{array}{c}\text{-}\\ \mathrm{p}\end{array}\right\}\left\{\begin{array}{c}\text{h or -}\\ \mathrm{s}\\ \mathrm{t}\end{array}\right\}\mathrm{FP} \qquad (2)$$

These expressions indicate that any generic single-cell fault can either be regular (-), initialization partial ($\mathrm{p_i}$), dirty (d) or partial dirty ($\mathrm{p_i}$d), while being hard (h or -), soft (s) or transient (t) at the same time. Two-cell faults can regular (-) or partial (p), while being hard, soft or transient. Note that some faults classes are considered unrealistic, such as activation partial ($\mathrm{p_a}$) single-cell faults, and therefore they are not included in the space.

For example, a transition 0 fault can be hard ($\mathrm{hTF_0}$), which is the same as the generic $\mathrm{TF_0}$. It can also be partial hard ($\mathrm{p_ihTF_0}$), dirty hard ($\mathrm{dhTF_0}$) and partial dirty hard fault ($\mathrm{p_idhTF_0}$) The same combinations apply for soft $\mathrm{TF_0}$ and transient $\mathrm{TF_0}$.

4 Tests for transient faults

Transient FPs mean that, after a fault is sensitized, leakage results in correcting the fault before it gets detected. This section derives the tests needed to detect single-cell and two-cell transient faults.

Detection conditions

An FP has two components to describe a fault: F (the value of the faulty cell) and R (the output on a read operation). Only F can be transient (get corrected by leakage), whereas R cannot, since it gets sensitized *and* detected on the output at the same time.

Table 3 lists all single-cell transient faults, along with their detection conditions. For example, the (partial, dirty and transient) transition 0 fault ($\mathrm{p_idt\ TF_0}$), must first be initialized a multiple number of times ($w1^h$). Then, the sensitizing write 0 operation can be performed ($w0$), before a completing operation with data 1 must be applied to a different cell along the same BL as v ($[O1_b]$). The detection condition starts with multiple $w1$ operations to initialize the cell to 1, directly followed by the sensitizing $w0$, the completing $O1_b$, and a detecting $r0$. Note that this detection condition is not a regular one, since it requires operations to be performed on two different cells (b and

Table 3. List of single-cell, transient FPs and their detection conditions. The underlined operations must be performed back-to-back.

#	Fault	$<S/F_L/R>, O \in \{w, r\}$	Detection cond., $O \in \{w, r\}$
1	dt SF$_0$	$<0_v[\underline{O1_b}]/1_L/->$	$\Updownarrow(..., w0, \underline{O1_b}, \underline{r0}, ...)$
2	dt SF$_1$	$<1_v[\underline{O0_b}]/0_L/->$	$\Updownarrow(..., w1, \underline{O0_b}, \underline{r1}, ...)$
3	p$_\text{i}$dt WDF$_0$	$<\underline{w0^h_v}[\underline{O1_b}]/1_L/->$	$\Updownarrow(..., \underline{w0^h}, \underline{O1_b}, \underline{r0}, ...)$
4	p$_\text{i}$dt WDF$_1$	$<\underline{w1^h_v}[\underline{O0_b}]/0_L/->$	$\Updownarrow(..., \underline{w1^h}, \underline{O0_b}, \underline{r1}, ...)$
5	p$_\text{i}$dt TF$_1$	$<\underline{w0^h_v}\underline{w1_v}[\underline{O0_b}]/0_L/->$	$\Updownarrow(..., \underline{w0^h}, \underline{w1}, \underline{O0_b}, \underline{r1}, ...)$
6	p$_\text{i}$dt TF$_0$	$<\underline{w1^h_v}\underline{w0_v}[\underline{O1_b}]/1_L/->$	$\Updownarrow(..., \underline{w1^h}, \underline{w0}, \underline{O1_b}, \underline{r0}, ...)$
7	p$_\text{i}$dt IRF$_0$	$<\underline{w0^h_v}[\underline{O1_b}]\underline{r0_v}/0_L/1>$	$\Updownarrow(..., \underline{w0^h}, \underline{O1_b}, \underline{r0}, ...)$
8	p$_\text{i}$dt IRF$_1$	$<\underline{w1^h_v}[\underline{O0_b}]\underline{r1_v}/1_L/0>$	$\Updownarrow(..., \underline{w1^h}, \underline{O0_b}, \underline{r1}, ...)$
9	p$_\text{i}$dt DRDF$_0$	$<\underline{w0^h_v}\underline{r0_v}[\underline{O1_b}]/1_L/0>$	$\Updownarrow(..., \underline{w0^h}, \underline{r0}, \underline{O1_b}, \underline{r0}, ...)$
10	p$_\text{i}$dt DRDF$_1$	$<\underline{w1^h_v}\underline{r1_v}[\underline{O0_b}]/0_L/1>$	$\Updownarrow(..., \underline{w1^h}, \underline{r1}, \underline{O0_b}, \underline{r1}, ...)$
11	p$_\text{i}$dt RDF$_0$	$<\underline{w0^h_v}[\underline{O1_b}]\underline{r0_v}/1_L/1>$	$\Updownarrow(..., \underline{w0^h}, \underline{O1_b}, \underline{r0}, ...)$
12	p$_\text{i}$dt RDF$_1$	$<\underline{w1^h_v}[\underline{O0_b}]\underline{r1_v}/0_L/0>$	$\Updownarrow(..., \underline{w1^h}, \underline{O0_b}, \underline{r1}, ...)$

v) within only one march element. The fact that the operations in these detection conditions need to be performed back-to-back is indicated by the underline below the corresponding operations.

In the same way, one may derive the detection conditions corresponding to all two-cell, hard faults.

Test algorithms

Based on the detection conditions of single-cell and two-cell transient faults, it is possible to derive memory tests that detect all these faults. A march test that detects all single-cell transient faults can be represented by March T1C (for transient, 1-cell) below.

March T1C = {

$\Updownarrow(\underline{w0^h}, \underline{w1_b}, \underline{r0})$;	$\Updownarrow(\underline{w1^h}, \underline{w0_b}, \underline{r1})$;
ME0	ME1
$\Updownarrow(\underline{w0^h}, \underline{w1}, \underline{w0_b}, \underline{r1})$;	$\Updownarrow(\underline{w1^h}, \underline{w0}, \underline{w1_b}, \underline{r0})$;
ME2	ME3
$\Updownarrow(\underline{w0^h}, \underline{r0}, \underline{w1_b}, \underline{r0})$;	$\Updownarrow(\underline{w1^h}, \underline{r1}, \underline{w0_b}, \underline{r1})\}$
ME4	ME5

This march test has six march elements (ME0 through ME5), each of which begins with a hammer write operation and ends with a detecting read operation. This test has a complexity of $(16 \cdot n + 6 \cdot h \cdot n)$. The march elements have special operations (such as $w1_b$) to be performed on a different cell along the same BL as the current cell of the march element. The operations in each march element must be performed back-to-back directly after each other (hence the underline below the operations in the test).

A march test that detects all two-cell, transient faults can be represented by March T2C below.

March T2C = {

$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w0^h_j}, \underline{r0_i}, \underline{r0_i}))$;	$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w1^h_j}, \underline{r0_i}, \underline{r0_i}))$;
ME0	ME1
$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0^h_j}, \underline{r1_i}, \underline{r1_i}))$;	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w1^h_j}, \underline{r1_i}, \underline{r1_i}))$;
ME2	ME3
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w0_j}, \underline{w1^h_j}, \underline{r0_i}))$;	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0_j}, \underline{w1^h_j}, \underline{r1_i}))$;
ME4	ME5
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w1_j}, \underline{w0^h_j}, \underline{r0_i}))$;	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w1_j}, \underline{w0^h_j}, \underline{r1_i}))$;
ME6	ME7
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w0_j}, \underline{r0^h_j}, \underline{r0_i}))$;	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0_j}, \underline{r0^h_j}, \underline{r1_i}))$;
ME8	ME9
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w1_j}, \underline{r1^h_j}, \underline{r0_i}))$;	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w1_j}, \underline{r1^h_j}, \underline{r1_i}))$;
ME10	ME11
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w0^h_j}, \underline{w0_i}, \underline{r0_i}))$;	$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w1^h_j}, \underline{w0_i}, \underline{r0_i}))$;
ME12	ME13
$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0^h_j}, \underline{w1_i}, \underline{r1_i}))$;	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w1^h_j}, \underline{w1_i}, \underline{r1_i}))$;
ME14	ME15
$\Updownarrow_i(\Updownarrow_j(\underline{w1^h_i}, \underline{w0^h_j}, \underline{w0_i}, \underline{r0_i}))$;	$\Updownarrow_i(\Updownarrow_j(\underline{w1^h_i}, \underline{w1^h_j}, \underline{w0_i}, \underline{r0_i}))$;
ME16	ME17
$\Updownarrow_i(\Updownarrow_j(\underline{w0^h_i}, \underline{w1^h_j}, \underline{w1_i}, \underline{r1_i}))$;	$\Updownarrow_i(\Updownarrow_j(\underline{w0^h_i}, \underline{w0^h_j}, \underline{w1_i}, \underline{r1_i}))\}$
ME18	ME19

This test has 20 march elements (ME0 through ME19), each of which contains a *nested march element*. This test has a complexity of $(56 \cdot n^2 + 24 \cdot h \cdot n^2)$. The reason behind the high computational complexity is the assumption that an aggressor can cause a fault in any victim anywhere in the memory. This assumption is, however, unrealistic. The impact of an aggressor is almost always limited to the adjacent neighboring cells. This observation can significantly simplify March T2C, by limiting the value of j to a limited number of adjacent cells. This reduces the complexity of the test from quadratic to linear with the number of cells, as discussed below.

5 Optimizing transient tests

The high time complexity of the two-cell march test (March T2C) for transient faults presented above stems from the assumption that each cell can be coupled to all other cells in the memory. This assumption is unnecessary, since practically a cell can only be coupled to the closest physically neighboring cells on the layout. Once the layout of the memory is known, it is possible to significantly reduce the time complexity of these tests.

The most widely used DRAM layout today is shown in Figure 3 [9]. In this figure, the circles represent memory cells, the horizontal lines represent word lines (WLs), while the vertical lines represent bit lines (BLs). BLs are organized in pairs of true (BT) and complementary (BC) bit lines. Note that the order of the WLs is scrambled (so-called *reflected WL organization*), such that WL3 follows WL1 instead of WL2.

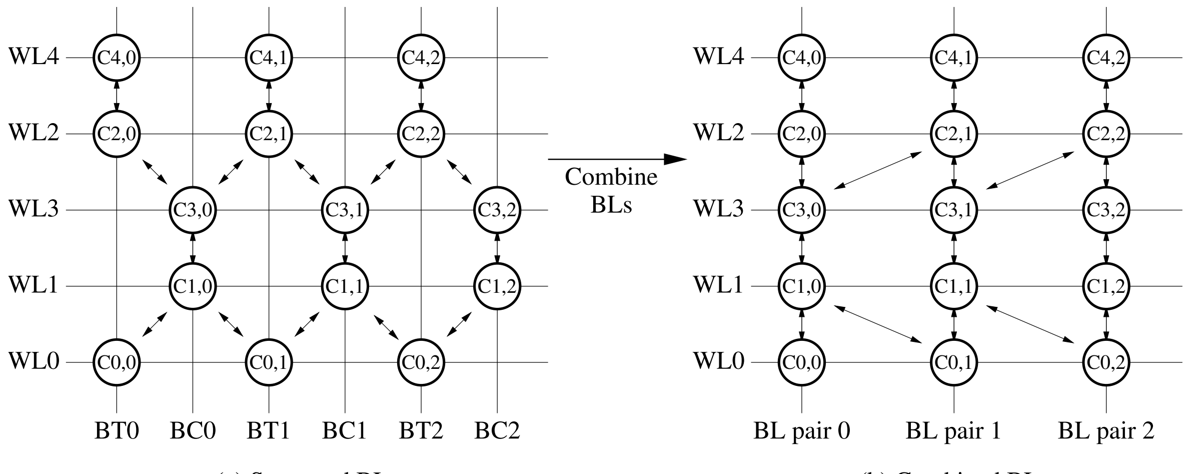

Figure 3. Physically neighboring cells (a) on the layout, and (b) with combined BLs (i.e., by combining BT and BC into a single BL pair).

Each memory cell in the figure is indicated by the letter C and a couple of numbers that refer to the WL and BL the cell is connected to. For example, the cell C3,2 is the memory cell connected to WL3 and BL2. All even numbered WLs (such as WL0, WL2, etc.) access cells connected to BT, while all odd numbered WLs access cells connected to BC. Considering C1,1, for example, the three closest neighboring cells on the layout are C0,1, C0,2 and C3,1.

According to this layout, each cell has three closest physical neighbors. This situation is shown in Figure 3(a), where the closest neighbors are highlighted by arrows that connect between them.

When march tests are applied to the memory under test, memory cells are accessed in an increasing, or a decreasing logical address order. In the memory shown in Figure 3(a), an increasing logical cell address corresponds for example to the following cell sequence C0,0, C1,0, C2,0, C3,0, C4,0, (then the rest of WLs are accessed), then C1,0, C1,1, etc. In this example, a march test accesses a cell on BT first, then a cell on BC, then again on BT, then BC, etc., according to their logical address and not to their physical position. From a march test point of view (i.e., using logical addressing), there is no difference between a cell connected to BT or to BC. Therefore, it is possible to combine each BT and BC of a given BL pair to inspect the way cell neighbors are organized from a march test point of view. This is done in Figure 3(b), which combines each BTx and BCx in Figure 3(a) into a single BL pair x line.

In order to identify all logically neighboring cells from Figure 3(b), it is only necessary to swap WL2 and WL3, so that a logically ordered WL sequence can be obtained. This is done in Figure 4. This figure shows clearly each cell and its closest neighbors, derived from the physical cell proximity in Figure 3(a).

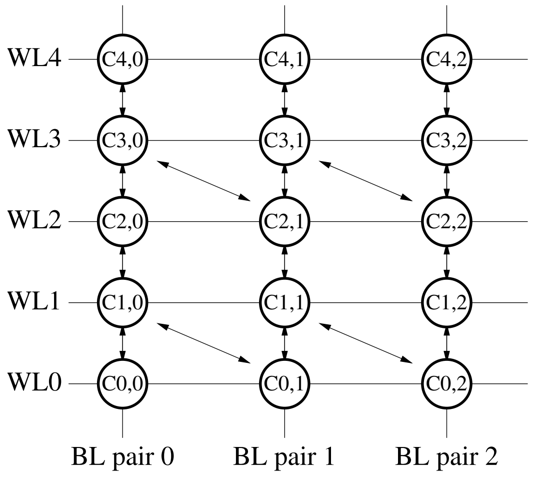

Figure 4. Logically neighboring cells.

Using this layout information, it is possible to reduce the complexity of March T2C to a more optimized test (called March T2C$_{layout}$), which has a linear complexity rather than quadratic, by limiting the possible cells that could act as aggressors to only the three cells connected by an arrow in Figure 4. The localized version of the test is given below.

March T2C$_{layout}$ = {

$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w0}_j^h, \underline{r0}_i, \underline{r0}_i));$ ME0	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w1}_j^h, \underline{r0}_i, \underline{r0}_i));$ ME1
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w0}_j^h, \underline{r1}_i, \underline{r1}_i));$ ME2	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w1}_j^h, \underline{r1}_i, \underline{r1}_i));$ ME3
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w0}_j, \underline{w1}_j^h, \underline{r0}_i));$ ME4	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w0}_j, \underline{w1}_j^h, \underline{r1}_i));$ ME5
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w1}_j, \underline{w0}_j^h, \underline{r0}_i));$ ME6	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w1}_j, \underline{w0}_j^h, \underline{r1}_i));$ ME7
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w0}_j, \underline{r0}_j^h, \underline{r0}_i));$ ME8	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w0}_j, \underline{r0}_j^h, \underline{r1}_i));$ ME9
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w1}_j, \underline{r1}_j^h, \underline{r0}_i));$ ME10	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w1}_j, \underline{r1}_j^h, \underline{r1}_i));$ ME11
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w0}_j^h, \underline{w0}_i, \underline{r0}_i));$ ME12	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w1}_j^h, \underline{w0}_i, \underline{r0}_i));$ ME13
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w0}_j^h, \underline{w1}_i, \underline{r1}_i));$ ME14	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w1}_j^h, \underline{w1}_i, \underline{r1}_i));$ ME15
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(\underline{w1}_i^h, \underline{w0}_j^h, \underline{w0}_i, \underline{r0}_i));$ ME16	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(\underline{w1}_i^h, \underline{w1}_j^h, \underline{w0}_i, \underline{r0}_i));$ ME17
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(\underline{w0}_i^h, \underline{w1}_j^h, \underline{w1}_i, \underline{r1}_i));$ ME18	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(\underline{w0}_i^h, \underline{w0}_j^h, \underline{w1}_i, \underline{r1}_i))\}$ ME19

The Δ_i in the test refers to the three cells in the neighborhood of the cell i, as shown by the cells connected with an arrow in Figure 3(a). This test is similar to March T2C, with the exception that the parameter j does not run through all memory cells, but only the neighboring cells to i. As an example of the test, march element ME0 can be understood as follows.

- For every the cell i in the memory start by initializing the cell to 0.
- Directly afterwards, write 0 an h number of times into every cell in the neighborhood of i.
- Directly afterwards, read 0 twice from cell i.

This test has 20 march elements, each of which has a nested march element. This test has a complexity of $(168 \cdot n + 72 \cdot h \cdot n)$, which is of the order $O(n)$. As an example, the complexity of ME0 can be calculated as follows: $3 \cdot n(3 + h)$, or $\#$ of cells $\times$ $\#$ of neighbors per cell $\times$ ($\#$ of single operations $+$ $\#$ of h repeated operations). Comparing this test with March T2C shows that the complexity of the test was significantly reduced, which makes much more suitable for industrial application.

6 Conclusions

This paper discussed the memory tests needed to detect transient faults in DRAM devices. The operations in these tests have to be performed back-to-back directly after each other. The assumption that all memory cells can be coupled to each, no matter where these cells are located, significantly increases the complexity of the tests. The paper showed how to use topological layout information to limit the length of these tests, reducing their complexity from quadratic $O(n^2)$ to linear $O(n)$. This makes them more practical candidates for implementation in the industry.

References

[1] 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.

[2] Z. Al-Ars and A.J. van de Goor, "Transient Faults in DRAMs: Concept, Analysis and Impact on Tests," *in Proc. IEEE Int'l Workshop on Memory Technology, Design and Testing*, 2001, pp. 59–64.

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

[4] Z. Al-Ars, A.J. van de Goor and S. Hamdioui, "Space of DRAM Fault Models and Corresponding Testing," *in Proc. Design, Automation and Test in Europe*, 2006, pp. 1–6.

[5] Z. Al-Ars *et al.*, "DRAM-Specific Space of Memory Tests," *in Proc. IEEE Int'l Test Conference*, 2006.

[6] G. Antonin, H.-D. Oberle and J. Kolzer, "Electrical Characterization of Megabit DRAMs, 1. External Testing," *in IEEE Design & Test of Computers*, vol. 8 , no. 3, 1991, pp. 36–43.

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

[8] A. Keshavarzi, K. Roy and C.F. Hawkins, "Intrinsic Leakage in Low Power Deep Submicron CMOS ICs," *in Proc. IEEE Int'l Test Conf.*, 1997, pp. 146–155.

[9] H.-D. Oberle and P. Muhmenthaler, "Test Pattern Development and Evaluation for DRAMs with Fault Simulator RAMSIM," *in Proc. IEEE Int'l Test Conf.*, 1991, pp. 548–555.

[10] A.J. van de Goor and A. Paalvast, "Industrial Evaluation of DRAM SIMM Tests," *in Proc. IEEE Int'l Test Conf.*, 2000, pp. 426–435.