New Algorithms for Address Decoder Delay Faults and Bit Line Imbalance Faults

Ad J. van de Goor
ComTex
Voorwillenseweg 201
2807 CA Gouda, The Netherlands
Ad.vd.Goor@kpnplanet.nl

Said Hamdioui   Georgi N. Gaydadjiev   Zaid Al-Ars
Delft University of Technology
Computer Engineering Laboratory
Mekelweg 4, 2628 CD Delft, The Netherlands
{S.Hamdioui, G.N.Gaydadjiev, Z.Alars}@tudelft.nl

Abstract

Due to the rapid decrease of technology feature size speed related faults, such as Address Decoder Delay Faults (ADDFs), are becoming very important. In addition, increased leakage currents demand for improved tests for Bit Line Imbalance Faults (BLIFs)(caused by memory cell pass transistor leakage). This paper contributes to new and improved algorithms for detecting these faults. First it provides an improved version of existing GalPat algorithm and introduces two new algorithms to detect ADDFs; the paper also shines a new light on the use of the different stress combinations (counting methods, data-backgrounds) and their importance for the detection of ADDFs. Second, it provides an improved algorithm for detecting BLIFs; it increases the defect coverage by being able to detect lower leakage currents.

Keywords: *Memory testing, Address Decoder Delay Faults, Address methods, Data backgrounds, Bit Line Imbalance Faults.*

1 Introduction

The discipline of memory testing has been evolving and maturing for a long time. However, due to the rapid decrease of the technology feature size, faults due to increased leakage currents, such as *Bit Line Imbalance Faults (BLIFs)*, and speed related faults, such as *Address Decoder Delay Faults (ADDFs)*, are becoming important [1]-[4]. In addition, the understanding of the used constructs for composing memory tests, such as the capabilities of the *Counting Methods (CMs)* [5] (i.e., linear, address complement, etc), and the use of Data Backgrounds (DBs), has made progress.

Traditionally BILFs are tested by applying walking 1 using fast-row addressing direction [2, 6]. In addition, most authors have solved the problem of detecting ADDFs by using a test called *Moving Inversion 'MOVI'* [4, 7, 8]. [9] even uses the time consuming GalPat test [5].

This paper contributes to the field of memory testing by introducing new algorithms and improving upon existing algorithms for detecting ADDFs and BLIFs.

For detecting ADDFs, the paper shows that the traditional expensive GalPat [10] algorithm detects such faults. Thereafter, an improved version of GalPat, GalPat-, is introduced; it reduces the test time by 25%, while maintaining the same fault coverage. In addition, two new efficient algorithms for detecting ADDFs are introduced: the Scan+ and the Worst Case Gate Delay (WCGD) algorithm.

For detecting BLIFs, a brief discussion of existing test algorithm is given, followed by an improved version of BLIF, called BLIF+. It puts the memory cell in a worst-case stress condition, hence improving the defect coverage; i.e., it detects lower leakage currents.

The paper is organized as follows. Section 2 introduces the notation used for describing the algorithms. Section 3 describes the ADDFs; Section 4 discusses testing for ADDFs, and introduces two new algorithms. Section 5 addresses testing of BLIFs and proposes an improved version of the traditional BLIF algorithm. Section 6 ends with conclusions.

2 Algorithm and stress notation

The algorithms in this paper consist of linear and non-linear algorithms, described with an extended notation for march algorithms. March algorithms are the most common algorithms used for testing memories. An example of a march algorithm is MATS+ [11], defined as: $\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$; see Table 1. The special symbols $\Updownarrow$, $\Uparrow$, and $\Downarrow$ are the *Address Orders (AOs)*; they determine the way one proceeds from one address to the next address. $\Uparrow$ denotes an ascending AO (e.g., 0,1,2,3,...), $\Downarrow$ denotes a descending AO, while $\Updownarrow$ denotes that the AO can be chosen freely. MATS+ consists of three *March Elements (MEs)*, which are separated by the ';' symbol. The ME '$\Uparrow(r0,w1)$' specifies the $\Uparrow$ AO, while to each address a read operation with expected value '0' will be applied, after

Table 1. Address Complement CM

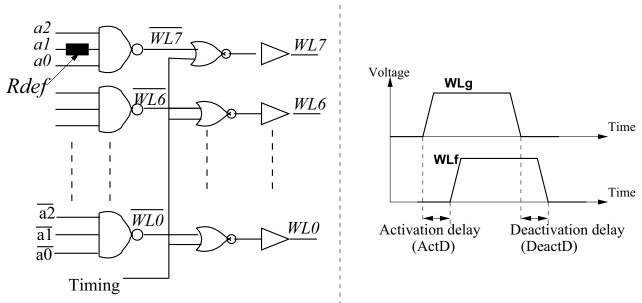
Step	Addr.	$^{AC}\Uparrow$	$^{AC}\Downarrow$
0	000		1,2,3
1	111	22,23,24	
2	001		4,5,6
3	110	19,20,21	
4	010		7,8,9
5	101	16,17,18	
6	011		10,11,12
7	100	13,14,15	

which a '1' will be written.

An algorithm stress specifies the way the algorithm is performed, and therefore it influences the sequence and/or the type of the memory operations. These stresses have shown to be important for the fault coverage of the algorithm [12],[13]. The following algorithm stresses are of interest for this paper:

1. *The Address Direction (AD)*. It is the extension of the one dimensional AO to the two dimensional space of the memory cell array. A real memory consists of a number of rows and columns (and thus also of a number of diagonals). The AD specifies the dimension along which the address sequence has to be applied and consists of three types: *Fast-row*, *Fast-column* and *Fast-diagonal*. Fast-row (Fast-column, Fast-diagonal) increments or decrements the row address (column address, diagonal address) most frequently. To indicate the Fast-row, Fast-column and Fast-diagnoal AD, the subscripts r, c and d are used with the AO respectively; e.g., $_r\Uparrow$ indicates $\Uparrow$ AO with Fast-row AD.
2. *The Counting Method (CM)*. It determines the address sequence. Many CMs exist; e.g., there are 6 ways of address counting for a 3-address memory: 012, 021, 102, 120, 201 and 210. It has been shown that the CM is important for detecting *Address Decoder Delay Faults* (ADDFs) [4, 14]. The most common CM is the *Linear* CM, denoted by the superscript 'L' of the AO (e.g., $^L\Uparrow$), where L specifies the address sequence 0,1,2,3, etc.). Because it is the default CM, the superscript 'L' is often deleted.

   Another CM is the *Address Complement (AC CM)*; it generates all address transitions $x \rightarrow \overline{x}$ by using $\Uparrow$ AO (denoted as $^{AC}\Uparrow$) and all address transitions $\overline{x} \rightarrow x$ using $\Downarrow$ AO (denoted as $^{AC}\Downarrow$). For example for a three-bit address (i.e., N=3), the AC CM specifies the following address sequences, see Table 1: $^{AC}\Uparrow$= 000, **111**, 001, **110**, 010, **101**, 011,**100** and $^{AC}\Downarrow$=**100**, 011, **101**, 010, **110**, 001, **111**, 000 [4, 14]; each bold address is the 1's complement of the preceding address. Note that the '$^{AC}\Downarrow$ starts with address 100 because it has to be the exact reverse of the '$^{AC}\Uparrow$'.



(a) Typical row decoder (b) ActD and DeactD

Figure 1. Typical row decoder and ADDFs

3. *The Data Background (DB)*. It is the data pattern which is actually in the cells of the memory cell array. The four DBs commonly used in industry are:

   Solid (sDB): all 0s (i.e., 0000.../0000...) or all 1s.
   Checkerboard (bDB): 0101.../1010.../0101.../1010...
   Column Stripes (cDB): 0101.../0101.../0101.../0101...
   Row Stripes (rDB): 0000.../1111.../0000.../1111...
   A 'w0' means that the selected DB is applied; a 'w1' means that the inverse of that DB is applied.

Note: the paper assumes that the logical width of the memory is 1 bit; [15] describes how a an algorithm for bit-oriented- can be modified for word-oriented memories.

3 Address decoder delay faults

Memories can exhibit static and dynamic faults. High speed memories, typically operating at speeds of 1 GHz or more, exhibit more dynamic faults, which are speed related. Especially Address Decoder Delay Faults (ADDFs) [4, 14] are increasing in importance, because address decoders typically consist of pre-decoders, connected to local word line decoders and local column decoders via long wires with many via's. Klaus [4] claims that the defect-per-million level decreases by 670 due to tests for ADDFs. Because of production tolerances, the wiring is susceptible to resistance and capacitance variations, while the increasing number of vias experiences extra resistive defects, causing RC delays. In a GHz memory, for example, an extra delay of 0.1 ns is already 10% of the clock frequency; hence, such memories are very sensitive to marginal delay defects [1]-[4], [16], which can be modeled as ADDF.

Figure 1(a) shows an example row decoder with 8 Word Lines (WLs), selected under control of the three address lines a_2,a_1,a_0. In case of a 'defect free' memory, the activation and deactivation of any word line will have no delay; see the timing of the word line WL_g in Figure 1(b). However, in the presence of a defect R_{def} in the path of an input (e.g., a_1), the activation and deactivation of the word line (e.g., WL_7) will be delayed as the timing of the word line

Table 2. Algorithms considered in this paper

#	Name	Test length	Description
1	MATS+	$5n$	$\{\Updownarrow(w0); \Uparrow(r0, w1); \Downarrow(r1, w0)\}$
2	GalPat	$4n^2 + 6n$	$\{\Updownarrow (w0); \Uparrow_v (w1_v, \Uparrow_{-v} (r0, r1_v), w0_v); \Updownarrow (w1); \Uparrow_v (w0_v, \Uparrow_{-v} (r1, r0_v), w1_v)\}$
3	GalRow	$4nR + 6n$	$\{\Updownarrow (w0); \Uparrow_v (w1_v, \Uparrow_{R-v} (r0, r1_v), w0_v); \Updownarrow (w1); \Uparrow_v (w0_v, \Uparrow_{R-v} (r1, r0_v), w1_v)\}$
4	GalCol	$4nC + 6n$	$\{\Updownarrow (w0); \Uparrow_v (w1_v, \Uparrow_{C-v} (r0, r1_v), w0_v); \Updownarrow (w1); \Uparrow_v (w0_v, \Uparrow_{C-v} (r1, r0_v), w1_v)\}$
5	Gal5R	$22n$	$\{\Updownarrow (w0); \Uparrow_v (w1_v, \diamond(r0, r1_v), w0_v); \Updownarrow (w1); \Uparrow_v (w0_v, \diamond(r1, r0_v), w1_v)\}$
6	Gal9R	$38n$	$\{\Updownarrow (w0); \Uparrow_v (w1_v, \square(r0, r1_v), w0_v); \Updownarrow (w1); \Uparrow_v (w0_v, \square(r1, r0_v), w1_v)\}$
7	Scan	$4n$	$\{\Downarrow(w0); \Uparrow(r0); \Uparrow(w1); \Downarrow(r1)\}$
8	BLIF	$8n$	$\{\Updownarrow (w0); {}_r\Uparrow(w1, r1, w0); \Uparrow (w1); {}_r\Uparrow(w0, r0, w1)\}$
New proposed algorithms			
9	Scan+	$6n$	$\{\Downarrow(w0); \Uparrow(r0); \Downarrow(r0); \Uparrow(w1); \Downarrow(r1); \Uparrow(r1)\}$
10	WCGD	$6n(1 + N)$	$\{\Uparrow (w0); \Uparrow_v(w1_v, \Uparrow_{i=0}^{N-1}(r0_{v\oplus 2^i}, r1_v, r0_{v\oplus 2^i}, w0_v)); \Uparrow(w1); \Uparrow_v(w0_v, \Uparrow_{i=0}^{N-1}(r1_{v\oplus 2^i}, r0_v, r1_{v\oplus 2^i}, w1_v))\}$
Improved algorithms			
11	BLIF+	$10n$	$\{\Uparrow (w0); {}_r\Uparrow(w1, w0_{nxt}, r1, w0); \Uparrow (w1); {}_r\Uparrow(w0, w1_{nxt}, r0, w1)\}$
12	GalPat-	$3n^2 + 6n$	$\{\Uparrow (w0); \Uparrow_{v=0}^{n-1} (w1_v, \Uparrow_{a=v+1}^{n-1} (r0_a, r1_v, r0_a), w0_v) \Uparrow (w1); \Uparrow_{v=0}^{n-1} (w0_v, \Uparrow_{a=v+1}^{n-1} (r1_a, r0_v, r1_a), w1_v)\}$
13	GalRow-	$3nR + 6n$	$\{\Uparrow (w0); \Uparrow_{v=0}^{R-1} (w1_v, \Uparrow_{a=v+1}^{R-1} (r0_a, r1_v, r0_a), w0_v) \Uparrow (w1); \Uparrow_{v=0}^{R-1} (w0_v, \Uparrow_{a=v+1}^{R-1} (r1_a, r0_v, r1_a), w1_v)\}$
14	GalCol-	$3nC + 6n$	$\{\Uparrow (w0); \Uparrow_{v=0}^{C-1} (w1_v, \Uparrow_{a=v+1}^{C-1} (r0_a, r1_v, r0_a), w0_v) \Uparrow (w1); \Uparrow_{v=0}^{C-1} (w0_v, \Uparrow_{a=v+1}^{C-1} (r1_a, r0_v, r1_a), w1_v)\}$

Note:

$\diamond$: apply operations to four neighbors of the v-cell: north, west, south and east	$N = R + C$ = the total number of address bits
$\square$: apply operations to eight neighbors of the v-cell	n= the total number of cells
$\Uparrow_{-v}$: apply operations to all cells, except the v-cell	C = the number of Column address bits
$\Uparrow_{R-v}$: apply operations to all cells of same Row as the v-cell, except the v-cell	R = the number of Row address bits
$\Uparrow_{C-v}$: apply operations to all cells of same Column as the v-cell, except the v-cell	${}_r\Uparrow$: denotes Fast-row addressing direction
$r0_{v\oplus 2^i}$: r0 from cell at address $v \oplus 2^i$	$w0_{nxt}$= w0 to next cell in same column

WL_f of Figure 1(b) shows. This results in the two fault models for address decoder delay faults: *Activation Delay (ActD)* and *Deactivation Delay (DeactD)*.

In the presence of R_{def} in the input line a_1, the *Address Transition (AT)* of any word line to WL7 may incur an ActD because of the extra RC time constant of input a_1 to the top NAND gate of Figure 1(a). Similarly, the same RC time constant may cause a DeacD when WL7 is deselected. Given R_{def}, the ActD will only occur upon an AT x**0**y→1**1**1 ($x, y \in \{0, 1\}$); i.e., at least input a_1 makes a 0→ 1 transition; the other inputs may, or may not, make a transition, depending on the values of x and y. Similarly, a DeactD will occur upon the AT 1**1**1→ 1**0**1; i.e., if only input a_1 makes a transition. From the above it can be seen that, given the 3-input top NAND gate of Figure 1(a), 4 ATs may cause an ActD: 000→111, 001→111, 100→111, and 101→111; while only a single AT may cause a DeactD which 111→101.

At this point it may be appropriate to introduce the concept of *Hamming distance (H)* between two addresses: H is defined as the number of bit positions in which two addresses differ; i.e., the detection of DeactD faults requires address pairs with H=1, while the detection of ActD faults requires address pairs with H≥1. Note that an algorithm which generates all address pairs with H=1 detects all ActD and all DeactD faults.

4 Testing of ADDFs

In this section it will be shown that the traditional GalPat algorithm detects ADDFs. Next, the GalPat- algorithm is introduced; it has the same fault coverage as the GalPat algorithm with a 25% reduction in test time. Thereafter two efficient new algorithms to detect ADDFs are introduced.

4.1 Traditional GalPat

Section 2 has shown that the detection of ADDFs requires special *Address Transitions (ATs)*. In addition, algorithms for detecting ActD and DeactD faults have to assume that the defect R_{def} can be located on any gate of the address decoder, and on any input of a gate. This means that they have to generate all ATs covering each possible position of R_{def}.

The most well-known algorithm which generates all address transitions between any pair of addresses is the GalPat algorithm; see Alg#2 in Table 2. This algorithm is non-linear and has a test time complexity of $O(n^2)$ (n is the size of the memory). It has the property that for a given victim cell (v-cell), which has e.g., the value '1', ATs are made from all other memory cells; these other cells are denoted as the aggressor cells (a-cells); see Figure 2(a). GalPat detects all ADDFs, because each cell will become v-cell and ATs are made between all cells. The algorithm applies a *Read-after-Read (RaR)* sequence, which means that a 'rx' operation is applied to the a-cell, followed by a '$r\overline{x}$' op-

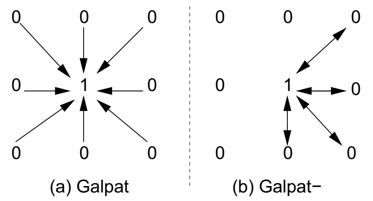


Figure 2. Galpat and Galpat- diagrams

eration to the v-cell [2]. In the march notation for GalPat '$\Uparrow_v(w1_v,\ \Uparrow_{-v}(r0, r1_v),\ w0_v)$' denotes a nested ME. The inner ME $\Uparrow_{-v}(r0, r1_v)$ is applied for each v-cell; the AO '$\Uparrow_{-v}$' means that all addresses are visited, except the v-cell, which has to be skipped; hence the subscript '-v'.

The test length of GalPat is too long for larger and/or slower memories. Subsets of this algorithm are the GalRow and the GalCol algorithms; see Alg#3 and Alg#4 in Table 2. They have the property that for a given v-cell, the a-cells are limited to the row or column of the v-cell. Hence only testing for ADDFs in the row or in the column decoder, but not in both decoders simultaneously. The time complexity of both GalRow and GalCol is of order $O(n^{\frac{3}{2}})$.

Even more simplified versions of GalPat are Gal5r and Gal9r [12]. They perform the same operations as GalPat, but only involving 5, respectively, 9 neighboring cells; thus reducing the time complexity from $O(n^2)$ to $O(n)$; see Alg#5 and Alg#6 in Table 2. Even though Gal5r and Gal9r are highly simplified versions of GalPat, their fault coverage are impressive as compared with regular march algorithms [12].

4.2 Improved GalPat

GalPat algorithm has the property that for a given v-cell, ATs are made from any a-cell to the current v-cell, while all cells become v-cell in turn; see Figure 2(a). Section 3 has shown that in order to detect all ADDFs, an AT has to be made between all pairs of cells, while applying a *Read-after-Read* sequence to the pair of cells. E.g., consider the pair of cells p and q: ATs have to be made from $p{\rightarrow}q$, and from $q{\rightarrow}p$. GalPat does this by making the cells p and q v-cell in turn, and applying two read operations for sensitizing and detecting possible ADDFs occurring during a $p{\rightarrow}q$ AT, and later two read operations for detecting possible ADDFs occurring during a $q{\rightarrow}p$ AT. The result is that two sequences, each with two read operations, are applied.

However, for a given cell q, the following two ATs can be made: from $p{\rightarrow}q$, and back from $q{\rightarrow}p$. In this case the following single sequence with three read operations can detect the possible ADDFs: $rx_p, r\overline{x}_q, rx_p$. The net result is that the number of read operations is reduced from 4 to 3, which is a 25% reduction; see Figure 2(b). The resulting new GalPat- algorithm is given in Table 2. Similar improvements can be made for GalRow and GalCol; see GalRow- (Alg#13) and GalCol- (Alg#14) in Table 2.

Table 3. Address transitions for ActD

WL	Def #	Defective line	Address Transitions
WL0	1	$\overline{a_2}$	$\mathbf{1}xy \rightarrow \mathbf{0}00$
	2	$\overline{a_1}$	$x\mathbf{1}y \rightarrow 0\mathbf{0}0$
	3	$\overline{a_0}$	$xy\mathbf{1} \rightarrow 00\mathbf{0}$
WL1	4	$\overline{a_2}$	$\mathbf{1}xy \rightarrow \mathbf{0}01$
	5	a_1	$x\mathbf{1}y \rightarrow 0\mathbf{0}1$
	6	a_0	$xy\mathbf{0} \rightarrow 00\mathbf{1}$
	..	...	...
WL7	22	a_2	$\mathbf{0}xy \rightarrow \mathbf{1}11$
	23	a_1	$x\mathbf{0}y \rightarrow 1\mathbf{1}1$
	24	a_0	$xy\mathbf{0} \rightarrow 11\mathbf{1}$

4.3 Scan+ algorithm

In addition to the Linear Counting Method (CM), the *Address Complement (AC)* CM [5] has a wide industrial use. The AC counting method has the property that, upon an address transition, N or $N-1$ address bits change; note: the total of address bits is N; see Table 1.

When using the AC CM many gates in the address decoder will switch, causing a maximum of noise and power dissipation (which causes voltage drops, noise spikes, and increases the die temperature). Hence the AC CM is a very effective stress. This paper shows that the AC has another property: it is able to generate all address pairs required for detecting all ActD faults, using only a linear algorithm.

Section 3 has shown that the detection of an ActD fault, for the case of R_{def} at input a_1 of a 3-input NAND gate, requires the AT x**0**y→1**1**1; (with $x, y \in \{0, 1\}$). Table 3 shows the ATs required for detecting ActD faults caused by R_{def} in the inputs of the NAND gates for WL0, WL1 and WL7. The entries for WL7 show that the detection of the ActD fault due to R_{def} in input a_2 (Def#22) requires the AT 0**x**y→1**1**1, R_{def} in input a_1 (Def#23) requires the AT x**0**y→1**1**1, while R_{def} in input a_0 (Def#24) requires the AT xy0→111. Hence, a defect in any of the three gate inputs can be detected with the single AT 000→111. Similarly, R_{def} in any of the inputs of the NAND gate for e.g., WL1 will be detectable with the AT 110→001 , etc.

The column 'Addr' of Table 1 shows the AC counting method; the column ${}^{AC}\Uparrow$ lists the ActD faults detected by the corresponding AT. For example, the AT 000→ 111 detects the defects Def#22, Def#23 and Def#24 of Table 3. The AT 111→000, see column ${}^{AC}\Downarrow$, detects the defects Def#1, Def#2 and Def#3. Hence, by performing the AC counting method for the $\Uparrow$ and the $\Downarrow$ AO, all ActD faults are detected.

Section 3 has already shown that a RaR sequence is required for the detection of the ADDFs. The Scan algorithm [11] is the simplest algorithm which allows for a RaR sequence; see Alg#7 in Table 2. Note that in order to support the RaR sequence, rDB or bDB should be used with Fast-row Addressing Direction (AD); while cDB or bDB should be used with Fast-column AD. In addition, rx-after-$r\overline{x}$ se-

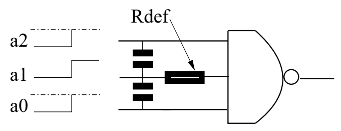


Figure 3. 3NAND gate

quences have to performed for x$\in\{0,1\}$ and for the $\Uparrow$ and $\Downarrow$ AOs, resulting in the modified version of Scan, denoted Scan+ (see Alg#9 in Table 2):
$\{\Downarrow(w0);\ {}^{AC}\Uparrow(r0);\ {}^{AC}\Downarrow(r0);\ \Uparrow(w1);\ {}^{AC}\Downarrow(r1);\ {}^{AC}\Uparrow(r1)\}$

Note that the AO of the March Element (ME) $\Downarrow(w0)$ is $\Downarrow$'; while the AO $\Uparrow$ has been chosen for the ME ' $\Uparrow(w1)$' in order to have write sequences with address transitions from a higher to a lower, and from a lower to a higher address. In addition, the AOs of the read operations have to be the inverse of that of the preceding write operations in order to prevent fault masking.

4.4 WCGD test

This section extends the ADDF model to include the *Worst Case Gate Delay (WCGD)* fault model. It comes from the observation that the switching speed of a gate depends on the number of inputs which change, together with the direction of their change. The worst case situation occurs when only a single input changes. E.g., for the defect R_{def} in Figure 3, only input a_1 needs to make 0→1 transition; the surrounding inputs (denoted by dotted input levels) do not make a transition. In this case, the capacitive load on the switching input is worst case; hence, the gate will exhibit its slowest behavior. When this gate is part of the address decoder like the one of Figure 1(a), then the ATs of Table 4 have to be generated in order to be able to detect the WCGD faults for each gate of the address decoder. E.g., if the R_{def} is in the input line a_1 of the NAND gate of WL$_7$, then the transition $101 \rightarrow 111 \rightarrow 101$ is required for WCGD. Note that the ATs consist of address pairs with hamming distance H=1. Therefore they sensitize both ActD and DeactD faults; hence the WCGD algorithm detects ActD and DeactD faults caused by the WCGD fault. E.g., for a defect in line a_2 of the NAND gate of WL$_7$, the AT $011 \rightarrow 111$ will sensitize the ActD while the AT $111 \rightarrow 011$ will sensitize the DeactD (see Table 4).

The WCGD algorithm (see Alg#10 in Table 2) is as follows:
$\{\Uparrow(w0);\quad \Uparrow_v(w1_v, \Uparrow_{i=0}^{N-1}(r0_{v\oplus 2^i}, r1_v, r0_{v\oplus 2^i}, w0_v));$
$\Uparrow(w1);\quad \Uparrow_v(w0_v, \Uparrow_{i=0}^{N-1}(r1_{v\oplus 2^i}, r0_v, r1_{v\oplus 2^i}, w1_v))\ \}$

The test length of the WCGD algorithm is: $6n(1+N)$. The notation assumes a decoder with N address bits and n addresses, as follows:

- $\Uparrow_{i=0}^{N-1}(r0_{v\oplus 2^i}, r1_v, r0_{v\oplus 2^i}, w0_v)$ contains a nested ME.
- $\Uparrow_{i=0}^{N-1}$: steps the a-cell of the nested ME; the address of the a-cell is the address of v-cell$\oplus 2^i$.

Table 4. Address transitions for WCGD

WL	Defective line	address transition
WL0	$\overline{a_2}$	$\mathbf{1}00 \rightarrow \mathbf{0}00 \rightarrow \mathbf{1}00$
	$\overline{a_1}$	$0\mathbf{1}0 \rightarrow 0\mathbf{0}0 \rightarrow 0\mathbf{1}0$
	$\overline{a_0}$	$00\mathbf{1} \rightarrow 00\mathbf{0} \rightarrow 00\mathbf{1}$
WL1	$\overline{a_2}$	$\mathbf{1}01 \rightarrow \mathbf{0}01 \rightarrow \mathbf{1}01$
	a_1	$0\mathbf{1}1 \rightarrow 0\mathbf{0}1 \rightarrow 0\mathbf{1}1$
	a_0	$00\mathbf{0} \rightarrow 00\mathbf{1} \rightarrow 00\mathbf{0}$
	...	...
WL7	a_2	$\mathbf{0}11 \rightarrow \mathbf{1}11 \rightarrow \mathbf{0}11$
	a_1	$1\mathbf{0}1 \rightarrow 1\mathbf{1}1 \rightarrow 1\mathbf{0}1$
	a_0	$11\mathbf{0} \rightarrow 11\mathbf{1} \rightarrow 11\mathbf{0}$

5 Testing for pass transistor leakage

This section derives the new *Bit Line Imbalance Fault (BLIF)* algorithm, which improves the fault coverage of the existing BLIF algorithm [5, 6]. Before the improved version will be introduced, the BLIF will be described together with it's traditional algorithm.

5.1 Bit Line Imbalance Fault (BLIF)

The BLIF was first introduced by Mazumder [6]. The root cause of the fault is the leakage current of the pass transistors of a cell. Figure 4 shows a v-cell connected to the *True Bit line (TB)* and the *Complement Bit line (CB)* via two pass transistors. During a read operation the v-cell is selected, by enabling the word line WL1, such that it's state is transferred to TB and CB. Note that all other cells in the same column, referred to as the a-cells, are not selected; i.e., their word lines are not enabled.

During a read operation, assuming that the v-cell contains a '1', the TB remains at its pre-charge level, while the CB is discharged. However, because of the leaking pass transistors, the a-cells in the same column also affect the charge level on TB and CB. Assuming that columns consist of C cells, and all a-cells have a '0' value, then TB will be discharged and CB will be charged by C-1 pass transistor leakage currents. The result may be that the read operation applied to the v-cell may fail.

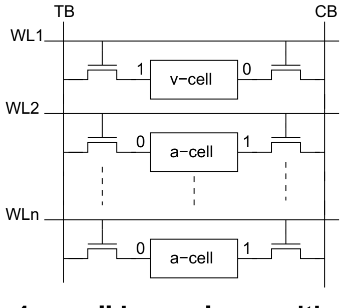


Figure 4. v-cell in a column with a-cells

5.2 Traditional BLIF Test

The following BLIF algorithm has traditionally been used for detecting this fault [5, 6]; see Alg#8 in Table 2: $\{\Uparrow(w0);\ _{r}\Uparrow(w1,r1,w0);\ \Uparrow(w1);\ _{r}\Uparrow(w0,r0,w1)\}$. After initializing the memory to all 0s, a Walking 1 is applied by the '$w1$' operation of the next ME using Fast-row addressing direction. When the '1' has been written, the other $C-1$ cells in the column have a '0' value, such that the result of the following '$r1$' operation is maximally impacted by the leakage currents of the $C-1$ a-cells in the column of the v-cell. The '$r1$' operation therefore should detect the BLIF. Because of the symmetry of the fault model, the algorithm has to apply a Walking 1 and a Walking 0 to each column.

5.3 Improved BLIF Test

The above traditional BLIF algorithm will detect BLIFs, assuming that the precharge operation, applied to the v-cell, after the '$w1$' and before the '$r1$' operation, completely precharges CB. However, during this precharge operation, the voltage on CB effectively behaves as the voltage of a capacitor being charged via a resistor, such that completely charging CB will take a long time. Today's fast memories only can meet their speed requirements by having a reduced precharge time. This means that some remnant charge of the '$w1$' operation is still left; this will oppose the detection of the BLIF by the '$r1$' operation of the ME $_{r}\Uparrow(w1,r1,w0)$. This can be prevented by performing a '$w0$' operation to a cell in the same column as the v-cell, prior to applying the '$r1$' operation to the v-cell. This change results in the improved BLIF+ algorithm (see Alg#11 in Table 2):
$\{\Uparrow(w0);$
$_{r}\Uparrow(w1,w0_{nxt},r1,w0);\ \Uparrow(w1);\ _{r}\Uparrow(w0,w1_{nxt},r0,w1)\}$.

The operation $w1_{nxt}$ means that a $w0$ operation is applied to the next cell in the column of the v-cell, such that the bit lines are pre-biased in the least favorable state for the following '$r1$' operation to pass. That way an imperfect precharge operation increases the likelihood for the '$r1$' operation to fail. Even though the BLIF+ algorithm requires two addresses simultaneously (i.e., the address of the a-cell and the address of the v-cell) the implementation of the address generator for the nxt address can be very simple.

6 Conclusions

In this paper new developments and insight in memory testing are covered. Two aspects are addressed.

Testing Address Decoder Delay Faults (ADDFs). The paper elaborates on the ADDFs which is of increasing importance. The ADDFs are shown to consist of two components: the Activation Delay (ActD) and the Deactivation Delay (DeactD) fault models; their detection requires special address transitions. In addition, it has been shown that the traditional GalPat algorithm detects ADDFs. Next, the GalPat- algorithm, which a reduced version of GalPat algorithm, is introduced. It has the same ADDFs fault coverage, but it's test length is 25% less than that of GalPat. Thereafter, two new algorithms are introduced: the Scan+ and the WCGD algorithms. Scan+ uses the *address complement* counting method and is the shortest algorithm (with a test lenght of $6n$) for detecting ActD faults. WCGD is proposed for detecting the WCGD fault, which occurs when a gate switches due to a change of a single input. The WCGD algorithm is non-linear and has a test length of $6n*(1+N)$; i.e., it depends on the number of address bits N.

Testing Bit Line Imbalance Faults (BLIFs) which are caused by the leakage current of the pass transistors. In order to improve the fault/defect coverage of the traditional BLIF algorithm, a new version is introduced. It puts the prechare of the bit line during the read operation in the worst-case stress combination, hence enabling the detection of lower leakage currents. The new version of the algorithms has a test length of $10n$ while the traditional one has a test length of $8n$.

References

[1] T. Powell, A. Kumar, J. Rayhawk and N.Mukherjee, 'Chasing Subtle Embedded RAM Defects for Nanometer Technologies', *In Proc. of the IEEE Int. Test Conf.*, paper 33.4, 2005.

[2] A. J. van de Goor, S. Hamdioui and R. Wadsworth, 'Detecting Faults in the Peripheral Circuits and an Evaluation of SRAM Tests', *In Proc. of the IEEE Int. Test Conf*, pp. 114-123, 2004.

[3] T.J. Powell, et al., 'BIST for Deep Submicron ASIC Memories with High Performance Application', *In Proc. of the IEEE Int. Test Conf.*, pp. 386-392, 2003.

[4] M. Klaus and A.J. van de Goor, 'Tests for Resistive and Capacitive Defects in Address Decoders', *In Proc. of the Tenth Asian Test Symposium*, pp. 31 - 36, 2001.

[5] A.J. van de Goor, Testing Semiconductor Memories, Theory and Practice, *ComTex Publishing*, Gouda, The Netherlands, 1998. Ad.vd.Goor@kpnplanet.nl

[6] P. Mazumder, 'Parallel Testing of Parametric Faults in a Three Dimensional Random-access Memory', *In Proc. of the IEEE Int. Test Conf.* , pp. 933-941, 1988.

[7] M.T. Fragano, J.H. Oppold, M.R. Ouellette and J.P. Rowland, "Self-Test Pattern to Detect Stuck-Open Faults", US Patent No.: US 6,442,085 B1, Aug. 27, 2002

[8] B. Nadeau-Dostie et al., "Serial Interfacing for Embedded-Memory Testing", *IEEE Design & Test of Comp.*, Vol. 7, No. 2, pp. 52-63, 1990.

[9] S. Nakahara et al., "Built-in self-test for GHz embedded SRAMs using flexible pattern generator and new repair algorithm", *in Proc. IEEE Int. Test Conf.*, pp. 301-310, 1999.

[10] M.A. Breuer and A.D. Friedman, 'Diagnosis and reliable design of digital systems, *Computer Science Press*, Woodland Hills, CA, USA, 1976.

[11] R. Nair, 'An Optimal Algorithm for Testing Stuck-at Faults Random Access Memories', IEEE transactions on Computers, Vol. C-28, No. 3, pp. 258-261, 1979.

[12] A.J. van de Goor and A. Paalvast, 'Industrial Evaluation of DRAM SIMM Tests', *In Proc. IEEE Int. Test Conf.*, pp. 426-435, 2000.

[13] S. Hamdioui, Z. Al-Ars, A.J. van de Goor, R Wadsworth, 'Impact of Stresses on the Fault Coverage of Memory Tests', *Proc. IEEE International Workshop on Memory Technology, Design and Testing*, pp. 103-108, 2005.

[14] S. Hamdioui, Z. Al-Ars, A.J. van de Goor, "Opens and Delay Faults in CMOS RAM Address Decoder", *IEEE Transactions on Computers*, pp. 1630-1639, Vol. 55, Issue 12, December 2006.

[15] A.J. van de Goor, I. B. S. Tlili and S. Hamdioui, "Converting March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories", *International Workshop on Memory Technology, Design and Testing*, pp. 46-52, 1998.

[16] Z. Conroy, G. Richmond, X. Gu and B. Eklow, 'A Practical Perspective on Reducing ASIC NTFs', *In Proc. of the IEEE Int. Test Conf.*, paper 14, 2005.