Opens and Delay Faults in CMOS RAM Address Decoders

Said Hamdioui, *Member*, *IEEE*, Zaid Al-Ars, *Member*, *IEEE*, and Ad J. van de Goor, *Fellow*, *IEEE*

Abstract—This paper presents a complete electrical analysis of Address decoder Delay Faults "ADFs" caused by resistive opens in RAMs. A classification between inter and intragate opens is made. A systematic way is introduced to explore the space of possible tests to detect these faults; it is based on generating appropriate sensitizing address transitions and the corresponding sensitizing operation sequences. DFT features are given to facilitate the BIST implementation of the new tests.

Index Terms—Memory testing, open defects, address decoder delay faults, addressing methods, BIST, DFT.

1 Introduction

Memory technology has been used for a long time to push the state-of-the-art in the semiconductor industry. The consequence of the continuous scaling and high density is the high sensitivity to defects. In addition, memories have signal lines running across the memory area with a very high fan out. Therefore, they have a high load and capacitance. Process variations make them very sensitive to delay type behavior.

In [19], Nigh and Gattiker report that new technology generations will exhibit an increasing sensitivity to, and occurrence of, subtle defect types, many of which will cause additional circuit delays, while the increasing clock speeds will make designs more sensitive to these circuit delays. The increasing use of copper wiring will shift the predominant failure mode from shorts and bridges to opens [13]. Needham et al. [18] report that opens were the most likely cause of field returns of Intel microprocessors. Klaus and van de Goor [12] report that tests for opens in DRAM address decoders reduced the DPM level by as much as 670. In conclusion, faults caused by opens, resulting in delays, are becoming a dominant failure mode.

Much has been published on functional fault models and tests for faults in the memory cell array [1], [2], [4], [9], [15], [16], [24]. However, faults in the address decoders and address decoder paths, denoted as *Address decoder Faults (AFs)* have only gotten limited attention. Several authors have shown the importance of this class of faults [7], [8], [12], [15], [17], [20], [22]. Most authors have solved the problem of detecting *Delay Faults* in the Address decoders, denoted as "*ADFs*," by using a test called *Moving Inversion (MOVI)* [7], [12], [15]. Nakahara et al. [17] even use the time consuming GalPat test [24]. Sacheve [22] has solved the problem by adding a decoder specific set of patterns to an existing march test. Azimane and Majhi [3] reported that the traditional march test may cover the ADFs when varying the duty cycle of the internal clock of the address decoder, while Dilillo et al. [5], [6] simulated some resistive defects to validate the delay faults and modified the known March C- test to target the same faults.

This paper presents an analysis, at the electrical level, of ADFs caused by resistive opens within the address decoder decoding paths, together with detection conditions and tests patterns. The paper is organized as follows: Section 2 discusses the traditional AFs, together with their detection conditions. Section 3 describes the causes of ADFs, classifies them, and gives some simulation examples. Section 4 presents the detection conditions for ADFs; Section 5 derives the tests. Section 6 describes some DFT features to facilitate the implementation of the tests. Last, Section 7 ends with the conclusions.

2 Traditional Address Decoder Faults (AFs)

For a long time, the traditional AFs were considered the only class of AFs [24]. They are described below, together with their detection condition. However, first, the notation of march tests will be given.

2.1 Notation of March Tests

A *march test* is a sequence of march elements. A *march element* consists of a sequence of operations applied to every cell (n is the number of cells in the memory), in either one of two *Address Orders (AOs)*: *Increasing* ($\Uparrow$) *AO* (e.g., from cell 0 to cell $n-1$) or a *Decreasing* ($\Downarrow$) *AO* (e.g., from cell $n-1$ to cell 0). It should be noted that $\Uparrow$ AO is often taken to be any permutation of the available addresses and that $\Downarrow$ AO is the exact reverse of the $\Uparrow$. When the AO is irrelevant, the symbol "$\Updownarrow$" is used.

Example. $\{\Updownarrow(w0); \Uparrow(r0, w1); \Downarrow(r1, w0)\}$ is the MATS+ test [16]. It consists of three march elements: M0, M1, and M2. $\mathrm{M1} = \Uparrow(r0, w1)$ means "for $i = 0$ to $n-1$ do {read $A[i]$ with expected value 0; $A[i] := 1$}."

2.2 Traditional AFs and Their Detection Condition

The following types of AFs have traditionally been the faults considered to occur in address decoders [24]:

- **AFna**: An address does **n**ot **a**ccess its cell.

• *The authors are with the Delft University of Technology, Faculty of Electrical Engineering, Mathematics, and Computer Science, Computer Engineering Laboratory, Mekelweg 4, 2628 CD Delft, The Netherlands. E-mail: {S.Hamdioui, Z.Alars, A.J.vandeGoor}@ewi.tudelft.nl.*

Manuscript received 22 May 2005; revised 11 Feb. 2006; accepted 12 Mar. 2006; published online 20 Oct. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0169-0505.

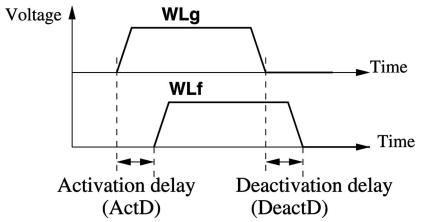


Fig. 1. Activation and deactivation delays.

- **AFmc**: An address uniquely accesses **m**ultiple **c**ells, i.e., this is the *only* address accessing those cells.
- **AFma**: A cell is uniquely accessed by **m**ultiple **a**ddresses, i.e., these addresses *only* access that cell.
- **AFoc**: An address additionally accesses **o**ther **c**ells.

Any march test will detect AFna through AFoc iff it satisfies *Condition AF* for $h \geq 1$ [24] (h from **h***ammer*). It consists of the following two march elements (note: the suffix "u" denotes *up* for the $\Uparrow$ AO, the suffix "d" denotes *down* for the $\Downarrow$ AO; "..." means any number of r (read) or w (write) operations, $\overline{x}$ means NOT x, and $[,r\overline{x}]^h$ ($[,rx]^h$) means h $r\overline{x}$ (rx) operations; $h \geq 1$):

$$\mathbf{AFh{-}u}\text{: } \Uparrow (rx, \ldots, w\overline{x}[,r\overline{x}]^h); x \in \{0,1\},$$
$$\mathbf{AFh{-}d}\text{: } \Downarrow (r\overline{x}, \ldots, wx[,rx]^h); x \in \{0,1\}.$$

3 Address Decoder Delay Faults (ADFs)

Opens are a major cause of delays in the address decoder paths, causing *Address decoder Delay Faults (ADFs)*. Fig. 1 shows a sequence of memory accesses, sequentially accessing memory locations with a *good Word Line* "WLg" (with an address A_g), a potentially *faulty Word Line* "WLf" (with an address A_f). In case of an ADF, the activation and/or the deactivation of WLf will be delayed, causing an *Activation Delay "ActD"* fault and/or a *Deactivation Delay "DeactD"* fault [12], [22], [25].

Next, a classification of resistive opens is given, their consequences are analyzed, and simulation examples showing the existence of the ADFs are presented.

3.1 Classification of Resistive Opens

Fig. 2 depicts a part of a *static* CMOS row address decoder. The decoding of the word lines ($WL0 - WL7$) is done using 3-input CMOS NAND gates and 2-input NOR gates, together with a buffer circuit. The signal "Timing" is used with the NOR gates to generate the timing of the word lines. The address of WLx is Ax; it specifies the values of the $N = 3$ address lines "a_2, a_1, a_0", e.g., $WL0$ is selected if $\overline{a_2}\,\overline{a_1}\,\overline{a_0} = 111$, indicating that the selected address is $A0 = a_2a_1a_0 = 000$.

In the decoder of Fig. 2, defects can cause opens at the following locations:

- Between logic gates (*intergate opens*). Fig. 2 shows three intergate opens (defects $Rdef1$, $Rdef2$, and $Rdef3$). $Rdef1$ is located in the line from $\overline{a_1}$ to the NAND gate decoding $WL0$.
- Inside logic gates (*intragate opens*). Fig. 3 shows a NAND gate with an intragate open (defect $Rdef4$) in the drain of the pull-up transistor for input $\overline{a_1}$.

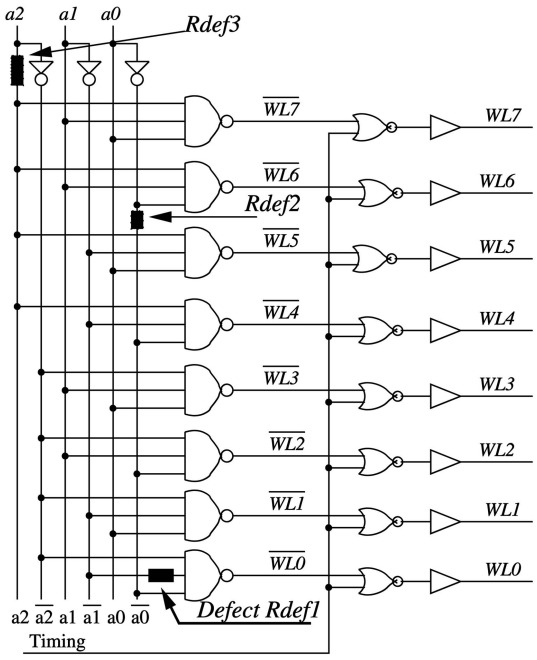


Fig. 2. Typical CMOS address decoder.

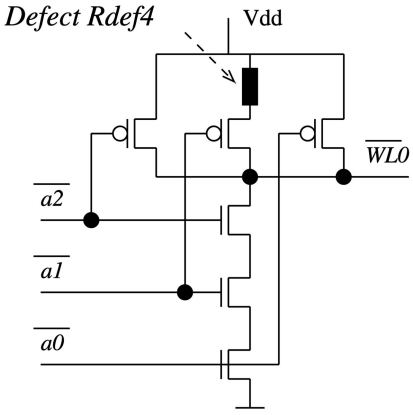


Fig. 3. Example of intragate open.

Klaus and van de Goor [12] state that the probability of intergate opens, caused by spot defects in the long global wiring, is at least one order of magnitude larger than that of intragate opens; the latter are caused by local spot defects in the short wiring within the decoder gates.

It is important to note here that, generally speaking, the column decoders have a similar structure as the row decoders. However, some differences may exist in their implementation such as in the control circuit that deactivates the word lines or column select (see the example in Section 3.4). These differences can impact the timing behavior of the decoders. However, the methodology used in this paper to analyze the row decoders can also be used to analyze the column decoders.

3.2 Intergate Opens

In this section, $Rdef1$ in Fig. 2 will be used to highlight the ActD and the DeactD on WL0. For sufficiently *high* values of $Rdef1$, the corresponding input of the NAND gate will behave as an open connection. Depending on the initial voltage of the floating node and the leakage current in the NAND gate, the input will be pulled low, which means that

$WL0$ will never be selected, causing the AFna of Section 2. Alternatively, the input will be pulled high, which means that $WL0$ will be selected, whenever $\{\overline{a_2}, \overline{a_0}\} = \{1, 1\}$, independent of $\overline{a_1}$, causing AFoc of Section 2.

When $Rdef1$ is *intermediate*, it will cause an ActD and a DeactD fault (see Fig. 1) on $WL0$, as explained below.

Activation delay ($WL0$ changes from $0 \rightarrow 1$). The ActD is caused by a $0 \rightarrow 1$ transition of $\overline{a_1}$ (see Fig. 2). This can be represented by the address transition $x\mathbf{1}y \rightarrow 0\mathbf{0}0$ of $a_2a_1a_0$; $x, y \in \{0, 1\}$. This is an address transition from $WL2 \rightarrow WL0$, from $WL3 \rightarrow WL0$, from $WL6 \rightarrow WL0$, or from $WL7 \rightarrow WL0$. Due to the ActD, the memory cycle involving $WL0$ may only be performed partially, which may lead to an incorrect operation, such as an incorrect read operation or a weakly written voltage.

Deactivation delay ($WL0$ changes from $1 \rightarrow 0$). The DeactD is caused by a $1 \rightarrow 0$ transition of $\overline{a_1}$ (see Fig. 2). This occurs upon an address transition $0\mathbf{0}0 \rightarrow 0\mathbf{1}0$ of $a_2a_1a_0$, which corresponds to the address transition $WL0 \rightarrow WL2$. The consequence of DeactD will be that $WL0$ will still be active, while the next address, accessing, e.g., $WLg = WL2$, is activated such that the operation on $WL0$ may not be completed properly and/or the operation on WLg may not be started properly (see Fig. 1). One example of a faulty behavior as a result of DeactD is the overwriting of the data in the disconnected cell.

3.3 Intragate Opens

Fig. 3 shows a resistive intragate open in the pull-up path of a CMOS NAND gate; this defect will be used to illustrate the DeactD due to the intragate open. The presence of the defect in the pull down path will cause an ActD, as will be explained next. It should be noted that the test consequences for the case that $Rdef4$ is very large (which causes the well-known CMOS Stuck-Open fault [11]) are a subset of the case when $Rdef4$ is intermediate.

Deactivation delay. When $Rdef4$ is intermediate, it will cause a delay on the *falling edge* of $WL0$ (i.e., DeactD) iff $\overline{WL0}$ changes from $0 \rightarrow 1$ due to $\overline{a_1}$ changing from $1 \rightarrow 0$ (see Fig. 3). This will occur with the address transition $0\mathbf{0}0 \rightarrow 0\mathbf{1}0$ of $a_2a_1a_0$.

Activation delay. If $Rdef4$ is in the *pull-down path*, a delay on the *rising edge* of $WL0$ (i.e., ActD) will occur iff $\overline{WL0}$ changes from $1 \rightarrow 0$ due to any input ($\overline{a_2}$, $\overline{a_1}$, or $\overline{a_0}$) changing from $0 \rightarrow 1$. This will occur with the address transition $xyz \rightarrow 000$ of $a_2a_1a_0$, where at least x, y, or z has the value 1.

It should be noted that opens in the pull-up or pull-down paths of CMOS NOR gates set the same addressing requirements on memory tests as those needed for opens in CMOS NAND gates. Therefore, they are not discussed from here on.

Fig. 4 shows a *dynamic address decoder*. Before an address is presented on the inputs of the decoder gate, all decoder gates are precharged using a "Prech" signal line. In case of a defect in the path of one of the pull-down transistors, a DeactD will be present on the output of the decoder. Note that the ActD cannot occur due to the nature of the decoder. The sensitization of the DeactD fault only requires the address of $WL0$ to be present on the inputs of the decoder. Hence, the two-address sequence as required for static address decoders is not required for dynamic address decoders. Therefore, the detection conditions of dynamic address decoders are a subset of those of static ones.

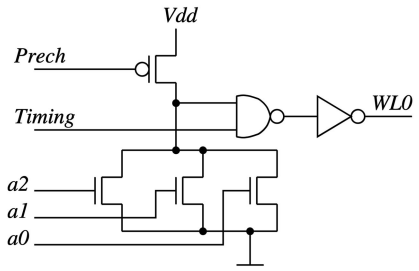


Fig. 4. Dynamic address decoder.

3.4 Simulation Examples for ADFs

Simulations have been performed for Infineon $0.18\mu\text{m}$ *e*DRAM technology for $Rdef$ in the very last stage of the row decoder (i.e., $Rdef1$ in Fig. 2), impacting the timing of the word line "WL" and in the very last stage of the column decoder (i.e., in a similar place as for row decoder), impacting the timing of the *Column Select "CS"* signal.

- *Opens in the row address decoder*. Fig. 5 shows five WL waveforms with gradually increasing open defect values: $Rdef1 = 0\Omega, 10\text{K}\Omega, 30\text{K}\Omega, 70\text{K}\Omega$ and $Rdef1 = 100\text{K}\Omega$. In general, the defect may cause both ActD and DeactD. Sometimes, as shown in the figure, only the ActD is caused but no DeactD because the considered implementation of the row decoder has a special circuit that deactivates the WL at a *fixed moment*.
- *Opens in the column address decoder*. Fig. 6 shows five CS waveforms: $Rdef1 = 0\Omega, 100\text{K}\Omega, 300\text{K}\Omega, 700\text{K}\Omega$ and $Rdef1 = 1000\text{K}\Omega$. The result of the defect is that both the ActD and DeactD are gradually delayed and the highest CS voltage reachable gradually decreases.
- *Consequences of* $Rdef1$. For large values of $Rdef1$, a read operation produces a fixed value. For intermediate values of $Rdef1$ the read operation produces a value which depends on a combination of the stored voltage in the cell and the coupling between the output buffer and other signals. The simulation results also have shown that write operations are less sensitive to $Rdef1$ than read operations.

4 Detection Conditions for ADFs

In case of an AF (see Section 2), it is assumed that the AF is detectable using read and write operations, applied using a particular address order "AO." However, the sensitization of *ADF*s is more complex and has two requirements:

1. *sensitizing address transitions* and
2. *sensitizing operation sequences*.

Sensitizing address transition(s) can be caused by an address pair or an address triplet. A *Sensitizing Address Pair (SAP)* consists of a sequence of *two* addresses $\{Ag, Af\}$ or $\{Af, Ag\}$ of Fig. 1, which have to be applied in sequence because ADFs are sensitized by *address transitions*. (Note: Ag presents the address of WLg and Af that of WLf.) When the two SAPs, $\{Ag, Af\}$ and $\{Af, Ag\}$, are applied in sequence, the *Sensitizing Address Triplet (SAT)* $\{Ag, Af, Ag\}$ can be applied instead. This is more efficient because only three addresses have to be applied, rather than four

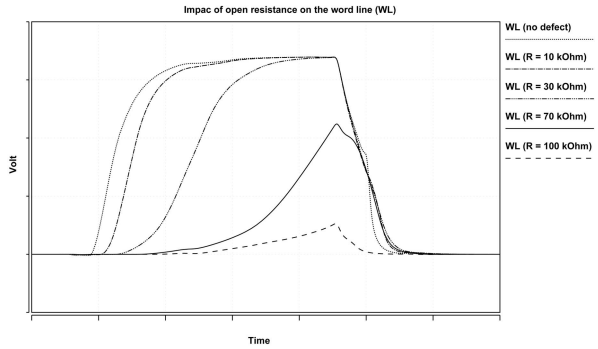


Fig. 5. Impact of Rdef on DRAM WL timing.

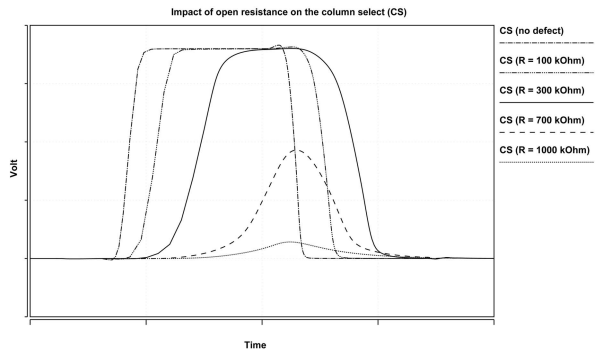


Fig. 6. Impact of Rdef on DRAM CS timing.

addresses when the two SAPs are applied. SAPs/SATs are generated using an *Addressing Method (AM)*.

Sensitizing Operation Sequence. To each address of a SAP or a SAT at least one operation has to be applied, resulting in a *Sensitizing Operation Sequence (SOS)* consisting, respectively, of two operations for a SAP and three operations for a SAT since a least one operation has to be applied to each address of a SAP (SAT).

4.1 Sensitizing Address Transitions

The addresses of the SAPs/SATs, required for sensitizing ADFs, are generated using an *Addressing Method (AM)*. An AM describes the method used for generating the sequence of addresses. A well-known AM is the *Binary AM "Bin"*; for $N = 3$, it consists of the address sequences $\Uparrow^{Bin} = \{0, 1, 2, 3, 4, 5, 6, 7\}$ and $\Downarrow^{Bin} = \{7, 6, 5, 4, 3, 2, 1, 0\}$.

Before discussing the AMs for ADFs, first the required address transitions for ADFs due to intergate and intragate opens will be generalized. The results are summarized in Table 1 for the three-bit address decoder of Fig. 2 and Fig. 3. The table lists the required address transitions for sensitizing the ActD and DeactD faults for intergate and intragate opens together with AMs that can generate such transitions, e.g., an ActD in WL0 due to an open in the line (denoted as "L"). $\overline{a_1}$ of NAND requires the address transition $x\mathbf{1}y \rightarrow 0\mathbf{0}0$, which can be generated either with AC, 2^i, or H1 AM (these AMs are addressed later on in this section). It is important to note that the AMs required for ActD and DeacD due to intergate opens are the same as those required for intragate opens. Therefore, the same AMs can be used to cover all ADFs.

Intergate Opens. Section 3.2 has shown that, for an *intergate open*, the *ActD fault* due to $Rdef1$, in the path of $\overline{a_1}$ of $WL0$, has to be sensitized with the address transition $x\mathbf{1}y \rightarrow 0\mathbf{0}0$ of $a_2a_1a_0$; this can be represented by the SAP $\{x\mathbf{1}y, 0\mathbf{0}0\}$, with $x, y \in \{0, 1\}$. If the defect is in the path of $\overline{a_2}$ or $\overline{a_0}$, then sensitization of the ActD in WL0 will require the address transitions $\mathbf{1}xy \rightarrow \mathbf{0}00$, respectively, $xy\mathbf{1} \rightarrow 00\mathbf{0}$ of $a_2a_1a_0$ (see Table 1). Thus, the only requirement the SAPs have to satisfy for the detection of ActD is that an $x \rightarrow \overline{x}$ transition has to be made for the line containing $Rdef$; other lines also may, or may not, make a transition. Because $Rdef$ can be present in any input of any gate, the set of SAPs has to contain $x \rightarrow \overline{x}$ transitions for each input of each gate, e.g., an ActD in WL7 due to an open in the path of a_2, a_1, or a_0 requires the address transitions $\mathbf{0}xy \rightarrow \mathbf{1}11$, $x\mathbf{0}y \rightarrow 1\mathbf{1}1$, or $xy\mathbf{0} \rightarrow 11\mathbf{1}$, respectively, of $a_2a_1a_0$. *The Address Complement (AC)* AM satisfies these requirements; it is discussed in Section 4.1.1.

TABLE 1
Address Transitions for ActD and DeactD Due to Inter and Intragate Opens

Opens	ADF	WL	L.	address transition	AM
Inter-gate	ActD	WL0	$\overline{a_2}$	$\mathbf{1}xy \rightarrow \mathbf{0}00$	AC, 2^i, H1
			$\overline{a_1}$	$x\mathbf{1}y \rightarrow 0\mathbf{0}0$	
			$\overline{a_0}$	$xy\mathbf{1} \rightarrow 00\mathbf{0}$	
		WL1	$\overline{a_2}$	$\mathbf{1}xy \rightarrow \mathbf{0}01$	
			a_1	$x\mathbf{1}y \rightarrow 0\mathbf{0}1$	
			a_0	$xy\mathbf{0} \rightarrow 00\mathbf{1}$	
		...	...	...	
		WL7	a_0	$\mathbf{0}xy \rightarrow \mathbf{1}11$	
			a_1	$x\mathbf{0}y \rightarrow \mathbf{1}1\mathbf{1}$	
			a_1	$xy\mathbf{0} \rightarrow \mathbf{1}1\mathbf{1}$	
	Deact	WL0	$\overline{a_2}$	$\mathbf{0}00 \rightarrow \mathbf{1}00$	2^i, H1
			$\overline{a_1}$	$0\mathbf{0}0 \rightarrow 0\mathbf{1}0$	
			$\overline{a_0}$	$00\mathbf{0} \rightarrow 00\mathbf{1}$	
		...	...	...	
		WL7	a_2	$\mathbf{1}11 \rightarrow \mathbf{0}11$	
			a_1	$1\mathbf{1}1 \rightarrow 1\mathbf{0}1$	
			a_0	$11\mathbf{1} \rightarrow 11\mathbf{0}$	
Intra-gate	ActD	WL0	$\overline{a_2}$	$\mathbf{x}yz \rightarrow \mathbf{0}00$	AC, 2^i, H1
			$\overline{a_1}$	$x\mathbf{y}z \rightarrow 0\mathbf{0}0$	
			$\overline{a_0}$	$xy\mathbf{z} \rightarrow 00\mathbf{0}$	
		...	...	...	
	DeactD	WL0	$\overline{a_2}$	$\mathbf{0}00 \rightarrow \mathbf{1}00$	2^i, H1
			$\overline{a_1}$	$0\mathbf{0}0 \rightarrow 0\mathbf{1}0$	
			$\overline{a_1}$	$00\mathbf{0} \rightarrow 00\mathbf{1}$	
		...	...	...	

Section 3.2 has also shown that, for an *intergate open*, the *DeactD fault* due to $Rdef1$, in the path of $\overline{a_1}$ of $WL0$ has to be sensitized with the address transition $0\mathbf{0}0 \rightarrow 0\mathbf{1}0$ of $a_2a_1a_0$ (see Table 1); this can be represented by the SAP $\{0\mathbf{0}0, 0\mathbf{1}0\}$. Similarly, if the defect is in the path $\overline{a_2}$ or $\overline{a_0}$ of WL0, then SAPs $\{\mathbf{0}00, \mathbf{1}00\}$, respectively, $\{00\mathbf{0}, 00\mathbf{1}\}$ will be required (see Table 1). In general, for sensitizing any DeactD due to any open in the path of any NAND-gate of an N-bit address decoder, all SAPs with $H = 1$ have to be generated, where H stands for the *Hamming distance* between the two addresses Ax and Ay of the SAP $\{Ax, Ay\}$. H is defined as the number of bit positions in which the addresses Ax and Ay of an address-pair differ. Only AMs which generate SAPs or SATs with $H = 1$ (i.e., the AMs 2^i and H1, which are described in Section 4.1.2 and Section 4.1.3) can cover DeactD's.

It is important to note here that the requirements for detecting the DeactD fault are more severe than those for

the ActD fault and, therefore, the AMs generating SAPs/SATs for DeactD can also be used for ActD (see Table 1).

Intragate Opens. Section 3.3 has shown that, for an *intragate* open in the *pull-up path* of $\overline{a_1}$ of the NAND gate of WL0 (see Fig. 3), the DeactD has to be sensitized with the SAP $\{0\mathbf{0}0, 0\mathbf{1}0\}$ of $a_2a_1a_0$. Similarly, if the open is in the pull-up path of $\overline{a_2}$ of $\overline{a_0}$, then the required SAPs of $a_2a_1a_0$ will be $\{\mathbf{0}00, \mathbf{1}00\}$, respectively, $\{00\mathbf{0}, 00\mathbf{1}\}$ (see Table 1). These are the same requirements as those of DeactD due to *intergate* opens (as discussed above) and, therefore, the AMs 2^i and H1 can be used (see Sections 4.1.2 and 4.1.3).

On the other hand, an ActD fault in WL0 due to a defect in the *pull-down path* of the NAND gate of WL0 (see Fig. 3) requires the SAP $\{xyz, 000\}$ of $a_2a_1a_0$ for its sensitization, where at least x, y, or z has the value 1. This is a less severe requirement than $H = 1$, therefore the ActD will automatically be covered by any test which sensitizes the DeactD.

Note also that, for intragate opens, the requirements for detecting an ActD fault are less severe than those for DeactD. Therefore, for generating SAPs/SATs for an ActD fault, either AC, 2^i, or H1 AM can be used.

4.1.1 Address Complement AM

The *Address Complement (AC)* AM generates all of the required SAPs: those requiring an $x \rightarrow \overline{x}$ transition by using the $\Uparrow$ AO (denoted as $\Uparrow^{AC}$) and those requiring an $\overline{x} \rightarrow x$ transition by using the $\Downarrow$ AO (denoted as $\Downarrow^{AC}$). For $N = 3$, the AC AM generates the following address sequences: $\Uparrow^{AC} = \{000, \mathbf{111}, 001, \mathbf{110}, 010, \mathbf{101}, 011, \mathbf{100}\}$ and the inverse sequence $\Downarrow^{AC} = \{100, \mathbf{011}, 101, \mathbf{010}, 110, \mathbf{001}, 111, \mathbf{000}\}$; each address is followed by its one's complement (in boldface). Note that the "$\Downarrow^{AC}$" starts with address "100" because it has to be the exact reverse of the $\Uparrow^{AC}$.

The AC AM satisfies the SAP requirements for ActD faults due to inter and intragate opens because the "$\Uparrow^{AC}$" AO generates all required SAPs for WL4 through WL7 and the "$\Downarrow^{AC}$" AO generates all required SAPs for WL0 through WL3, e.g., in the "$\Uparrow^{AC}$" AO, the SAP $\{000, \mathbf{111}\}$ generates all required address transitions for any open in any path (i.e., a_2, a_1, or a_0) of WL7 (see Table 1). In the "$\Downarrow^{AC}$" AO, the SAP $\{100, \mathbf{011}\}$ generates all required address transitions for any open in any path (i.e., $\overline{a_2}$, a_1, or a_0) of WL3, and so on.

The number of addresses which have to be written for the AC AM is: $N_{AC}(N) = 2 * 2^N = 2n$, where N is the number of address lines and n is the number of addresses; the factor of "2" is because the $\Uparrow^{AC}$ and the $\Downarrow^{AC}$ AOs are required. For $N = 3$, $N_{AC}(3) = 16$.

4.1.2 The 2^i Addressing Method

The 2^i addressing method consists of two AOs (denoted as $\Uparrow^i$ *and* $\Downarrow^i$), generating all required SAPs for detecting DeactD faults due to inter- and intragate opens (see Table 2). The SAPs for the $\Uparrow^i$ AO consist of the address pairs {even-address, odd-address}, for example, the address pairs {0, 1}, {2, 3}, {4, 5}, and {6, 7}. For the $\Downarrow^i$ AO, the SAPs consist of the address pairs {odd-address, even-address}. The addresses are generated using a binary counting sequence, with increments/decrements of $j = 2^i$, where $0 \leq i \leq N - 1$. The $\Uparrow^i$ sequence, for $j = 2^0 = 1$ (denoted by the "$\Uparrow^0$" symbol) is shown in the second column by the address sequence "$0, 1, 2, 3, \ldots, 6, 7$"; it represents all 3-bit SAPs with $H = 1$ due to $0 \rightarrow 1$ transitions in bit $i = 0$. The $\Downarrow^0$ sequence is represented by the address sequence "$7, 6, 5, \ldots, 2, 1, 0$" of the same column; it represents all SAPs with $H = 1$ due to $1 \rightarrow 0$ transitions in bit $i = 0$. This means that both the $\Uparrow^i$ and $\Downarrow^i$ address sequences have to be used.

TABLE 2
The 2^i Addressing

i; $\mathbf{j} = 2^i$	0; **1**	1; **2**	2; **4**
$\Uparrow^i$	$xy\mathbf{0} \rightarrow xy\mathbf{1}$	$x\mathbf{0}y \rightarrow x\mathbf{1}y$	$\mathbf{0}xy \rightarrow \mathbf{1}xy$
$\Downarrow^i$	$xy\mathbf{1} \rightarrow xy\mathbf{0}$	$x\mathbf{1}y \rightarrow x\mathbf{0}y$	$\mathbf{1}xy \rightarrow \mathbf{0}xy$
Addresses	$0 = 00\mathbf{0}$	$0 = 0\mathbf{0}0$	$0 = \mathbf{0}00$
generated	$1 = 00\mathbf{1}$	$2 = 0\mathbf{1}0$	$4 = \mathbf{1}00$
with	$2 = 01\mathbf{0}$	$4 = 1\mathbf{0}0$	$1 = \mathbf{0}01$
increment	$3 = 01\mathbf{1}$	$6 = 1\mathbf{1}0$	$5 = \mathbf{1}01$
of j,	$4 = 10\mathbf{0}$	$1 = 0\mathbf{0}1$	$2 = \mathbf{0}10$
using end-	$5 = 10\mathbf{1}$	$3 = 0\mathbf{1}1$	$6 = \mathbf{1}10$
around	$6 = 11\mathbf{0}$	$5 = 1\mathbf{0}1$	$3 = \mathbf{0}11$
carry	$7 = 11\mathbf{1}$	$7 = 1\mathbf{1}1$	$7 = \mathbf{1}11$

The SAPs of the 2^i AM satisfy the requirements for detecting DeactD due to inter and intragate opens because, e.g., the required address transitions for DeactD fault in WL0 (see Table 1) will be generated with the first two addresses of $\Uparrow^2$ (i.e., $\mathbf{0}00 \rightarrow \mathbf{1}00$), the first two addresses of $\Uparrow^1$ (i.e., $0\mathbf{0}0 \rightarrow 0\mathbf{1}0$), and the first first addresses of $\Uparrow^0$ (i.e., $00\mathbf{0} \rightarrow 00\mathbf{1}$) (see Table 2).

The number of addresses to be generated for the 2^i AM is: $N_{2^i}(N) = 2 * N * 2^N$, where the factor 2 presents the two AOs $\Uparrow^i$ and $\Downarrow^i$, N presents the number of addresses, and 2^N presents the number of addresses in an $\Uparrow^i$ or an $\Downarrow^i$ sequence. For $N = 3$, this results in $N_{2^i}(3) = 48$.

4.1.3 The H1 Addressing Method

This AM generates the *minimal* set of SATs with $H = 1$, where H stands for the *Hamming distance* between each address pair $\{Ax, Ay\}$ and $\{Ay, Ax\}$. The H1 AM is based on the concept of *constant weight codes*, also referred to as *m-out-of-n codes* [21]. They have the property that each n-bit *Code-Word "CW"* contains exactly m 1s (i.e., has a *weight* of $W = m$). In this application, the CWs are N-bit addresses; the maximum number of different CWs is $C_W^N = \frac{N!}{W!*(N-W)!}$. The idea is to use CWs with an *even* weight (i.e., $W = 0$, $W = 2$, $W = 4$, etc.). Then, for a given CW, N different SATs are generated by complementing and thereafter recomplementing successively one bit of the CW; this guarantees that the three addresses of each SAT have a Hamming distance of $H = 1$. These N complementing/recomplementing operations have to be performed for all CWs. For example, for the 3-bit code-word "000," the following SATs are generated: {000, 001, 000}, {000, 010, 000} and {000, 100, 000}. These three SATs can be combined into the *SuperSAT*: {000, 001, 000, 010, 000, 100, 000}, where the third address of each SAT is also the first address of the next SAT.

Table 3 gives an example of the H1 addressing method for $N = 3$, using SATs as well as SuperSATs. For each CW, three SATs are generated because $N = 3$. The "—" denotes that SuperSATs are used. That means that the last address of the previous SAT is also the first address of the SAT which starts with "—." The SATs, or SuperSATs, are generated for all CWs with *even* weights (i.e., $W = 2i$), which is the *set of CWs* with

TABLE 3
H1 Addressing Method for $N = 3$

#	W	Code-word	SAT	SuperSAT
1	0	000	{000,001,000}	{000,001,000}
2	0	000	{000,010,000}	{—,010,000}
3	0	000	{000,100,000}	{—,100,000}
4	2	011	{011,010,011}	{011,010,011}
5	2	011	{011,001,011}	{—,001,011}
6	2	011	{011,111,011}	{—,111,011}
7	2	110	{110,111,110}	{110,111,110}
8	2	110	{110,100,110}	{—,100,110}
9	2	110	{110,010,110}	{—,010,110}
10	2	101	{101,100,101}	{101,100,101}
11	2	101	{101,111,101}	{—,111,101}
12	2	101	{101,001,101}	{—,001,101}
Note: '—' denotes use of last address of previous SAT				

$W = 0, W = 2, W = 4, \ldots,$ until $W = 2 * \lfloor N/2 \rfloor$. The total number of CWs is $N_{CW} = \sum_{i=0}^{\lfloor N/2 \rfloor} C_{2i}^{N} = 2^{N-1}$. For example, for $N = 3$ (see Table 3), two sets of CWs (i.e., C_0^3 and C_2^3) will be generated. The set of CWs for C_0^3 consists of one CW, which is {000}. The set of CWs for C_2^3 consists of three CWs: {011, 110, 101}. Therefore, N_{CW} for $N = 3$ is 4.

The SATs (and SuperSATs) generated with H1 AM satisfy the requirements for detecting DeactD due to inter and intragate opens, e.g., the required address transitions for DeactD fault on WL0 (see Table 1) will be generated with the first three SATs of Table 3 (i.e., {000, 001, 000}, {000, 010, 000} and {000, 100, 000}). Note that the same address transitions will be generated with the corresponding SuperSAT.

The number of addresses to be generated for the H1 AM, for the case where *SATs* are used, is $N_{H1}(N) = 3 * N * N_{CW}$ (i.e., for each code word, N different SATs are generated). Hence, $N_{H1}(N) = 3 * N * (\sum_{i=0}^{\lfloor N/2 \rfloor} C_{2i}^{N}) = 3 * N * 2^{N-1}$. The resulting number of addresses to be generated for the H1 AM using SuperSATs for *each* CW is:

$$3(\text{for the first word}) + 2(\text{for additional words})$$
$$= 3 + 2 * (N - 1) = 2N + 1.$$

The total number of addresses will then be:

$$N_{H1_S}(N) = (2N + 1) * N_{CW} = (2N + 1) * 2^{N-1}.$$

The use of SuperSATs therefore reduces the number of required addresses by a factor of: $\frac{2N+1}{3N}$. For $N = 3$, $N_{CW} = 4$, and, therefore, $N_{H1}(3) = 36$ and $N_{H1_S}(3) = 28$.

It is interesting to compare the number of required addresses of the optimal SAT-based H1 AMs with that of the corresponding SAP-based 2^i AMs. As has been shown in Section 4.1.2, the $N_{2^i}(N) = 2 * N * 2^N$. Since $N_{H1_S}(N) = (2N + 1) * 2^{N-1}$, the ratio will be:

$$\frac{N_{H1}(N)}{N_{2^i}(N)} = \frac{3 * N * 2^{N-1}}{N * 2^{N+1}} = \frac{3}{4} = 0.75.$$

On the other hand, the ratio of the **S**uperSAT H1 AM over the 2^i AM is: $\frac{N_{H1_S}(N)}{N_{2^i}(N)} = \frac{(2N+1)*2^{N-1}}{N*2^{N+1}} = \frac{2N+1}{4N}$; this ratio is 0.583 for N = 3 and approaches 0.5 for large values of N.

TABLE 4
Read-Write-Sequences (SOP/SOT)

RWS	March element 'ME'
RaW	$\Updownarrow^{AM}(r\overline{x}, ..., wx)$
RaR	$\Updownarrow^{AM}(r\overline{x}, ..., wx, rx)$
WaW	$\Updownarrow^{AM}(w\overline{x}, r\overline{x}, ..., wx)$
WaR	$\Updownarrow^{AM}(w\overline{x}, ..., wx, rx)$
RaRaR	$\Updownarrow(wx); \Updownarrow_f(w\overline{x}_f, \Updownarrow_g^{AM}(rx_g, r\overline{x}_f, rx_g), wx_f)$
RaRaW	$\Updownarrow_f(w\overline{x}_f, \Updownarrow_g^{AM}(wx_g, r\overline{x}_f, rx_g), wx_f)$
RaWaR	$\Updownarrow(wx); \Updownarrow_f(\Updownarrow_g^{AM}(rx_g, w\overline{x}_f, rx_g), wx_f)$
WaRaR	$\Updownarrow(wx); \Updownarrow_f(w\overline{x}_f, \Updownarrow_g^{AM}(rx_g, r\overline{x}_f, wx_g), wx_f)$
RaWaW	$\Updownarrow_f(\Updownarrow_g^{AM}(wx_g, w\overline{x}_f, rx_g), wx_f)$
WaRaW	$\Updownarrow_f(w\overline{x}_f, \Updownarrow_g^{AM}(wx_g, r\overline{x}_f, wx_g), wx_f)$
WaWaR	$\Updownarrow(wx); \Updownarrow_f(\Updownarrow_g^{AM}(rx_g, w\overline{x}_f, wx_g), wx_f)$

4.2 Sensitizing Operation Sequence

To each address of a SAP or a SAT at least one operation has to be applied, resulting in *Sensitizing Operation Pairs* for a SAP and *Sensitizing Operation Triplets* for a SAT. They are discussed next.

4.2.1 Sensitizing Operation Pairs (SOPs)

The required SOP for the SAP $\{Ag, Af\}$ ($\{Af, Ag\}$) consists of two operations "Ox_g, Oy_f" (Oy_f, Ox_g), see Fig. 1: one operation applied to Ag (Af) and the other to Af (Ag). "Ox_g" denotes the operation "O" ($O \in \{r, w\}$), with written or expected data x ($x \in \{0, 1\}$). The subscript g (f) denotes that the operation is applied to Ag (Af). Below, a set of requirements for the two operations of a SOP are given such that they sensitize the ActD and DeactD faults.

- *ActD*: Ox_g, $O\overline{x}_f$; $x \in \{0, 1\}$, $O \in \{r, w\}$. The operation on Af has to be performed with the *complement* of the data value applied to Ag in order for the fault to be detectable. Because of the ActD, $O\overline{x}_f$ may fail (see Fig. 1).
- *DeactD*: Ox_f, $O\overline{x}_g$; $x \in \{0, 1\}$, $O \in \{r, w\}$. Because of the DeactD Ox_f and/or $O\overline{x}_g$ may fail.

Note: x should take on the value $x = 0$ *as well as* the value $x = 1$ because of the likely asymmetric sensitivities to the 0 and 1 state; this is an engineering requirement.

Depending on the selected operations $O \in \{r, w\}$, four *Read-Write-Sequences (RWSs)* are possible (see the top block in Table 4). Note that the RWS of a SOP applied to $\{Ag, Af\}$ ($\{Af, Ag\}$) consists of the following *two* operations: the *last* operation applied to Ag (Af) and the *first* operation applied to Af (Ag). For example, "$\Updownarrow^{AM}(r\overline{x}, \ldots, wx, rx)$" performs the *Read-after-Read (RaR)* RWS of operations using a certain addressing method AM (i.e., AC, 2^i, or H1). The "wx" operation is required in order to change the state to x such that the "rx" operation can follow as the last operation of the *March Element (ME)*.

4.2.2 Sensitizing Operation Triplets (SOTs)

The required SOT for the SAT "$\{Ag, Af, Ag\}$" (see Fig. 1) consists of three operations: Ox_g, $O\overline{x}_f$, Ox_g; $x \in \{0, 1\}$, $O \in \{r, w\}$. (*Note*: x should take on the value $x = 0$ *as well as* the value $x = 1$ for the same reasons as for SOPs.)

TABLE 5
Tests for Address Decoder Delay Faults

#	Prop.	Time	Test description
1	RaW-AC	$9n$	$\{\Updownarrow(w0);\Uparrow^{AC}(r0,w1);\Uparrow^{AC}(r1,w0);\Downarrow^{AC}(r0,w1);\Downarrow^{AC}(r1,w0)\}$
2	RaR-AC	$13n$	$\{\Updownarrow(w0);\Uparrow^{AC}(r0,w1,r1);\Uparrow^{AC}(r1,w0,r0);\Downarrow^{AC}(r0,w1,r1);\Downarrow^{AC}(r1,w0,r0)\}$
3	WaW-AC	$12n$	$\{\Uparrow^{AC}(w0,r0,w1);\Uparrow^{AC}(w1,r1,w0);\Downarrow^{AC}(w0,r0,w1);\Downarrow^{AC}(w1,r1,w0)\}$
4	WaR-AC	$12n$	$\{\Uparrow^{AC}(w0,w1,r1);\Uparrow^{AC}(w1,w0,r0);\Downarrow^{AC}(w0,w1,r1);\Downarrow^{AC}(w1,w0,r0)\}$
5	RaW-2^i	$n+8nN$	$\{\Updownarrow(w0);i_0^{N-1}[\Uparrow^i(r0,w1);\Uparrow^i(r1,w0);\Downarrow^i(r0,w1);\Downarrow^i(r1,w0)]\}$
6	RaR-2^i	$n+12nN$	$\{\Updownarrow(w0);i_0^{N-1}[\Uparrow^i(r0,w1,r1);\Uparrow^i(r1,w0,r0);\Downarrow^i(r0,w1,r1);\Downarrow^i(r1,w0,r0)]\}$
7	WaW-2^i	$12nN$	$\{i_0^{N-1}[\Uparrow^i(w0,r0,w1);\Uparrow^i(w1,r1,w0);\Downarrow^i(w0,r0,w1);\Downarrow^i(w1,r1,w0)]\}$
8	WaR-2^i	$12nN$	$\{i_0^{N-1}[\Uparrow^i(w0,w1,r1);\Uparrow^i(w1,w0,r0);\Downarrow^i(w0,w1,r1);\Downarrow^i(w1,w0,r0)]\}$
9^A	RaRaR-H1	$n+4nN$	$\{\Updownarrow^{H1}(w0_g^c,w1_f;r0_g,r1_f,r0_g);\Updownarrow^{H1}(w1_g^c,w0_f;r1_g,r0_f,r1_g)\}$
10	RaRaW-H1	$4nN$	$\{\Updownarrow^{H1}(w1_f;w0_g,r1_f,r0_g);\Updownarrow^{H1}(w0_f;w1_g,r0_f,r1_g)\}$
11^A	RaWaR-H1	$n+3nN$	$\{\Updownarrow^{H1}(w0_g^c;r0_g,w1_f,r0_g);\Updownarrow^{H1}(w1_g^c;r1_g,w0_f,w1_g)\}$
12^A	WaRaR-H1	$n+4nN$	$\{\Updownarrow^{H1}(w0_g^c,w1_f;r0_g,r1_f,w0_g);\Updownarrow^{H1}(w1_g^c,w0_f;r1_g,r0_f,w1_g)\}$
13	RaWaW-H1	$3nN$	$\{\Updownarrow^{H1}(w0_g,w1_f,r0_g);\Updownarrow^{H1}(w1_g,w0_f,r1_g)\}$
14	WaRaW-H1	$4nN$	$\{\Updownarrow^{H1}(w1_f;w0_g,r1_f,w0_g);\Updownarrow^{H1}(w0_f;w1_g,r0_f,w1_g)\}$
15^A	WaWaR-H1	$n+3nN$	$\{\Updownarrow^{H1}(w0_g^c;r0_g,w1_f,w0_g);\Updownarrow^{H1}(w1_g^c;r1_g,w0_f,w1_g)\}$
16	RaR-MOVI	$13nN$	$\{i_0^{N-1}[\Downarrow^i(w0);\Uparrow^i(r0,w1,r1);\Uparrow^i(r1,w0,r0);\Downarrow^i(r0,w1,r1);\Downarrow^i(r1,w0,r0)]\}$
17	RaR-PMOVI	$13n$	$\{\Downarrow(w0);\Uparrow(r0,w1,r1);\Uparrow(r1,w0,r0);\Downarrow(r0,w1,r1);\Downarrow(r1,w0,r0)\}$
18	RaRaR-Min	$5n$	$\{\Updownarrow^{Min}(w0_g,w1_f;r0_g,r1_f,r0_g);\Updownarrow^{Min}(w1_g,w0_f;r1_g,r0_f,r1_g)\}$
19	RaRaW-Min	$4n$	$\{\Updownarrow^{Min}(w1_f;w0_g,r1_f,r0_g);\Updownarrow^{Min}(w0_f;w1_g,r0_f,r1_g)\}$
20	RaWaR-Min	$4n$	$\{\Updownarrow^{Min}(w0_g;r0_g,w1_f,r0_g);\Updownarrow^{Min}(w1_g;r1_g,w0_f,w1_g)\}$
21	WaRaR-Min	$5n$	$\{\Updownarrow^{Min}(w0_g,w1_f;r0_g,r1_f,w0_g);\Updownarrow^{Min}(w1_g,w0_f;r1_g,r0_f,w1_g)\}$
22	RaWaW-Min	$3n$	$\{\Updownarrow^{Min}(w0_g,w1_f,r0_g);\Updownarrow^{Min}(w1_g,w0_f,r1_g)\}$
23	WaRaW-Min	$4n$	$\{\Updownarrow^{Min}(w1_f;w0_g,r1_f,w0_g);\Updownarrow^{Min}(w0_f;w1_g,r0_f,w1_g)\}$
24	WaWaR-Min	$4n$	$\{\Updownarrow^{Min}(w0_g;r0_g,w1_f,w0_g);\Updownarrow^{Min}(w1_g;r1_g,w0_f,w1_g)\}$

Depending on the selected operations in $O \in \{r, w\}$, eight RWSs for SOTs are possible (see the bottom block in Table 4): RaRaR, RaRaW, RaWaR, RaWaW, WaRaR, WaRaW, WaWaR and WaWaW. The RWS WaWaW will not be considered from here on because at least one "r" operation has to be present in order to detect the ADF. This violates the WaWaW requirement and, therefore, the RWS WaWaW is not included in Table 4. The RWSs for SOTs use addresses triplets "Ag, Af, Ag" in order to allow for SATs; the first and the third addresses are identical. The nested ME "$\Updownarrow_g^{AM}(Ox_g, O\overline{x}_f, Ox_g)$" is performed for each address "Af." The nested ME for the WaRaR RWS has the following form: "$\Updownarrow_g^{AM}(rx_g, r\overline{x}_f, wx_g)$." First, a "$rx_g$" is applied to "$Ag$", next, an "$r\overline{x}_f$" is applied, and, last, a "wx_g.".

All RWSs *ending* with an "R" (i.e., that means that the nested ME *starts* with an "rx_g" operation and has the form "XaYa**R**" with $\mathrm{X}, \mathrm{Y} \in \{\mathrm{R}, \mathrm{W}\}$) require initialization of "Ag"; this is accomplished by the ME "$\Updownarrow(wx)$." Initialization of "Af" is required for the RWSs of the form "Xa**R**aY." This requires *two* extra operations: one to write the value "$\overline{x}$" and one to write back the original value "x." This is performed only *once* for each "Af" address by the ME "$\Updownarrow_f(w\overline{x}_f, \Updownarrow_g^{AM}(Ox_g, r\overline{x}_f, Ox_g), wx_f)$." The RWSs of the form "Xa**W**aY" require the extra "wx_f" operation to write back the original value "x" in "Af."

5 Tests for Address Decoder Delays Faults

Based on the AMs of Section 4.1 and the SOPs/SOTs of Section 4.2, tests for detecting ADFs (due to inter and intragate opens) can be constructed. The results are given in Table 5; entries #1 through #4 list the tests based AC AM, entries #5 through #8, the tests based on 2^i AM, and the entries #9 through #15 list the tests based on H1 AM. The left column shows the test #, the second column lists the test property (this is the RWS together with the AM), the column "Time" lists the test time, in terms of the required number of operations; the last column gives the test. It is important to note here that tests based on AC AM detect ActD faults only, while tests based on 2^i and H1 AMs detect ActD as well as DeactD faults.

Inspecting the AC AM and the 2^i AM reveals that SAPs are generated and, therefore, SOPs are required, while inspecting the H1 AM reveals that SATs are generated instead of SAPs. Therefore, SOTs are required. Consequently, tests based on AC and 2^i AMs use SAPs and SOPs and tests based on H1 AM use SATs and SOTs.

Four tests based on the AC AM as well as based on the 2^i AM for ADFs can be distinguished because of the four possible RWSs (see tests #1 through #8 in Table 5). The 2^i AM-based tests use the notation "$i_0^{N-1}[\Uparrow^i \ldots]$." This means that the part between the square brackets has to be repeated for $i = 0$ through $i = N-1$ such that addresses are incremented/decremented with 2^i.

It is important to note here that the well-known MOVI tests [4], [24] are also based on 2^i AM. MOVI tests are included in Table 5 (i.e., tests #16 and #17). MOVI (test #16) repeats MEs of the *Partial MOVI* (test #17) using 2^i addressing. MOVI is a variant of the RaR-2^i test; the initializing ME "$\Updownarrow(w0)$" is replaced with the ME "$\Downarrow(w0)$" and is repeated for each value of i. MOVI was designed as a

TABLE 6
RaRaR-H1 Operations

#	CW	SAT			Operations
		Ag	Af	Ag	
1	000	000	001	000	$w0_g, w1_f; r0_g, r1_f, r0g$
2	000	000	010	000	$w0_g, w1_f; r0_g, r1_f, r0g$
3	000	000	100	000	$w0_g, w1_f; r0_g, r1_f, r0g$
4	011	011	100	011	$w0_g, w1_f; r0_g, r1_f, r0g$
..	011	...	...	...	...

low-cost version of GalPat [24] and, because of that, it is used frequently in the industry.

Seven tests based on the H1 AM can be distinguished (see tests #9 through #15 in Table 5). They have been derived from the test structures of the bottom block of Table 4 by repeating those for the data values $x = 0$ and $x = 1$ and by adapting the initializing operations to the appropriate AMs. The latter is important because the H1 AM has the property that it does not access all 2^N words. Hence, the initializing operations do not have to be applied to all words. As an example, consider The RaRaR-H1 (test #9). The test has to access the sequence of "H1" addresses (see Table 3). Table 6 will be used to explain the test. Each row in the first block of the table describes a complete 3-address SAT for "CW = 000"; with addresses "Ag," "Af," and "Ag." The last column lists the operations to be applied to the SAT, consisting of *two* initializing operations, separated by ";" from the *three* operations for the RWS. The number of operations required per CW of N SATs consists of $4N + 1$:

- $N + 1$ initializing write operations per CW: two for the first SAT and one for each of the remaining $N - 1$ SATs.
- $3N$ read operations per CW (i.e., three reads per SAT).

Hence, the total test time is

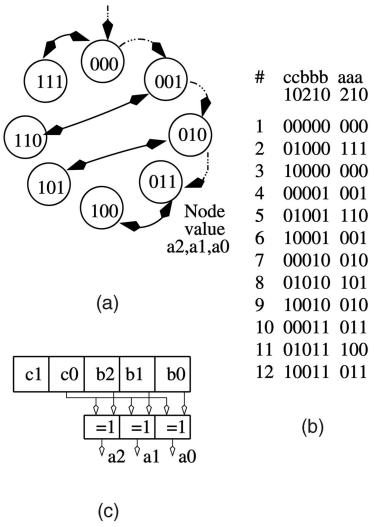
$$(4N + 1) * 2 * N_{CW} = n + 4Nn.$$

(Note: A factor 2 is added since the test has two similar MEs and $N_{CW} = \sum_{i=0}^{\lfloor N/2 \rfloor} C_{2i}^{N} = 2^{N-1} = \frac{n}{2}$.)

6 DFT and BIST Features

As discussed in Section 4, the detection of ActD and DeactD caused by both intergate and intragate opens requires the 2^i or H1 AM (see also Table 1). In addition, it has been shown that the total number of addresses of optimal SAT-based H1 AM is 25 percent less than that of 2^i addressing. This section describes an efficient BIST implementation for generating H1 AM. However, it first gives a minimal AM for detecting ActD faults, together with its BIST implementation.

It is important to note that, for embedded applications, the BIST complexity of generating the various addressing sequences (AC, 2^i, or H1) is also important as the overall number of the required patterns. For example, an AC AM implementation needs two N-bit counters. An efficient implementation for generating 2^i addressing requires a binary up/down N-counter, together with a DFT provision, which allows the least significant address line to be swapped with any of the other address lines [7].



#	ccbbb aaa 10210 210
1	00000 000
2	01000 111
3	10000 000
4	00001 001
5	01001 110
6	10001 001
7	00010 010
8	01010 101
9	10010 010
10	00011 011
11	01011 100
12	10011 011

Fig. 7. Minimal addressing for $N = 3$.

6.1 Minimal Tests and Their BIST for ActD

The "Minimal AM," described next, is the most time-efficient for detecting ActD due to inter and intragate opens; it has a reduction in length by 25 percent as compared with AC AM.

Each gate of the decoder of Fig. 2 is represented by a node in Fig. 7a. The value of the node represents the value of address lines $a_2a_1a_0$. For each node-pair $\{Ax, Ay\}$, two SAPs $\{Ax, Ay\}$ and $\{Ay, Ax\}$ have to be generated in order to guarantee the $x \rightarrow \overline{x}$ and the $\overline{x} \rightarrow x$ transitions on the line with $Rdef$ (see Table 1). These two SAPs can be combined into the *Sensitizing Address Triplet (SAT)* $\{Ax, Ay, Ax\}$ containing both address transitions. The SATs are denoted by the solid bidirectional arrows connecting complementary node-pairs. They constitute a *minimal* set of SATs required to generate all address transitions for detecting ActD faults. Note: The unidirectional dotted arrows are required to connect the node-pairs.

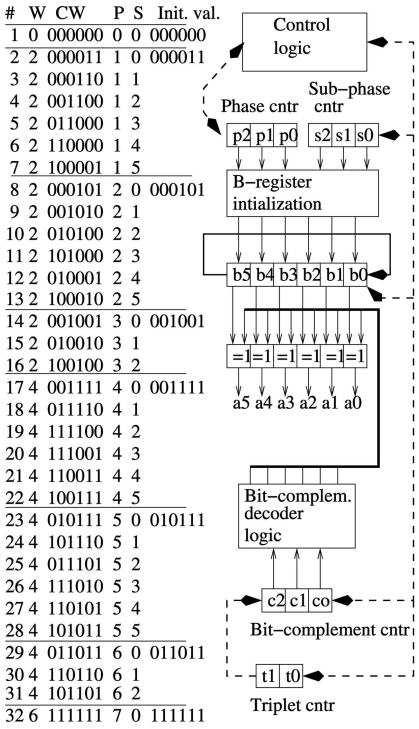
Fig. 7b shows the address sequence for $N = 3$. The column "#" indicates the *step* in the sequence of addresses, the columns "a_2, a_1, a_0" (the "a" is placed *above* the digits "0," "1," and "2") show the value of the address; the columns "$c1c0b2b1b0$" are an implementation detail (explained next). The number of addresses required for an N-bit address is after N-bit address is

$$N_{Min}(N) = 3 * (\text{number of address pairs}) * \frac{2^N}{2}(\text{number of SATs}) = 3 * 2^{N-1},$$

thus $N_{Min}(N) = 3 * \frac{n}{2}$.

Fig. 7c shows a BIST circuit for generating the Minimal address sequence, using a 5-bit counter: $c1c0b2b1b0$. The first 2 bits, $c1c0$, form a modulo-3 counter. The last 3 bits, $b2b1b0$, form a binary counter, which will be incremented upon a {1, 0} to {0, 0} transition of $c1c0$. The outputs of the counter $b2b1b0$ feed XOR-gates, denoted by the blocks with the "=1" symbol, to generate the address signals $a_2a_1a_0$. The resulting counting sequence is shown in Fig. 7b.

The required *Sensitizing Operation Triplet (SOT)* for the SAT "$\{Ag, Af, Ag\}$" consists of three operations: Ox_g, $O\overline{x}_f$, Ox_g; $x \in \{0, 1\}$, $O \in \{r, w\}$, combining the Minimal AM and the RWSs results in tests #18 through #24 of Table 5. They



#	W	CW	P	S	Init. val.
1	0	000000	0	0	000000
2	2	000011	1	0	000011
3	2	000110	1	1	
4	2	001100	1	2	
5	2	011000	1	3	
6	2	110000	1	4	
7	2	100001	1	5	
8	2	000101	2	0	000101
9	2	001010	2	1	
10	2	010100	2	2	
11	2	101000	2	3	
12	2	010001	2	4	
13	2	100010	2	5	
14	2	001001	3	0	001001
15	2	010010	3	1	
16	2	100100	3	2	
17	4	001111	4	0	001111
18	4	011110	4	1	
19	4	111100	4	2	
20	4	111001	4	3	
21	4	110011	4	4	
22	4	100111	4	5	
23	4	010111	5	0	010111
24	4	101110	5	1	
25	4	011101	5	2	
26	4	111010	5	3	
27	4	110101	5	4	
28	4	101011	5	5	
29	4	011011	6	0	011011
30	4	110110	6	1	
31	4	101101	6	2	
32	6	111111	7	0	111111

Fig. 8. H1 addressing for $N = 6$.

have similar structure as the H1-based tests and, therefore, the same explanation applies, e.g., the operations of RaRaR-Min (test #18) consist of *two* initializing operations, separated by ";" from the *three* operations for the RWS. The test time for RaRaR-Min test is 2(for $x = 0$ and $x = 1$) $*5/3$(because every SAT which consists of three addresses requires two additional initializing write operations, "wx_g, $w\overline{x}_f$") $*3n/2$(this is the $N_{Min}(N)) = 2 * \frac{5}{3} * \frac{3n}{2} = 5n$.

6.2 BIST Implementation of H1 AM

Fig. 8 gives an implementation example of the H1 addressing method for $N = 6$. The left part of the figure shows a table with the CWs. The column "#" lists the sequence number of the CW, the column "W" lists the weight of the corresponding CW, the column "CW" lists the CWs. The next three columns are included to explain the hardware implementation: "P" lists the Phase of the CW generator, "S" the Subphase, and "Init.val." lists the value used to (re)initialize the generator. Because $N = 6$, CWs have to be generated for the weights $W = 0$, $W = 2$, $W = 4$, and for $W = 6$ (see column "W"). The sets of CWs for the different weights is $\sum_{i=0}^{3} C_{2i}^{6} = 2^{6-1} = 32$ CWs. The total number of addresses is $N_{H1_S}(6) = (2 * 6 + 1) * 32 = 416$.

A hardware scheme for generating the CWs is shown in the right-hand part of Fig. 8. The "B" register "$b5, \ldots, b0$" contains the generated CWs. For each CW, six SATs with $H = 1$ are generated on the final outputs "$a5, \ldots, a0$," which are the outputs of the XOR gates. The XOR gates are fed by the register "$b5, \ldots, b0$" and the "Bit-complement decoder logic." This logic generates, under the control of the 3-bit "Bit-complement cntr" "$c2, c1, c0$," six outputs. They are all "0" or exactly one output has the value "1." The Super SATs are generated under control of the "Triplet cntr" "$t1$, $t0$."

The register "$b5, \ldots, b0$" is (re)initialized under control of the "Phase counter" "$p2, p1, p0$." The start of each new phase requires a new initialization (see the left part of the figure, column "P," together with the corresponding (re)initialization value "Init.val."). During each phase, the "Subphase counter" "$s2, s1, s0$," column "S" in the table, determines when the current phase has been completed. During a phase, register "$b5, \ldots, b0$" acts as a right-circular shift register. The "Control logic" takes care of the overall control.

7 Conclusions

In this paper, the Address decoder Delay Faults "ADFs" (divided into activation delay "ActD" and deactivation delay "DeactD" faults), due to intergate and intragate opens have been analyzed. Addressing Methods "AMs," together with the required Read-Write-Sequences "RWSs," for detecting such faults have been established. The AMs and the RWSs are combined to explore the space of all possible tests.

Tests for ActD faults require the Address Complement "AC" or the more time-efficient Minimal AM. Several versions of those tests have been designed in a systematic way, based on the possible RWSs. On the other hand, tests for DeactD faults require the 2^i or H1 AMs. Tests based on such AMs also cover the ActD faults. When the address decoder has the property of deactivating the word line at a *fixed* moment, then the DeactD cannot occur and, therefore, using a test based on AC or Minimal AM is the most suitable. However, when both ActD and DeactD faults are possible, a test based on the 2^i or H1 AMs should be used.

Finally, the paper presents BIST circuits for efficient implementation of the Minimal and H1 AMs-based tests.

The question now is: Which ADF test is the best to use when also considering the memory cell array faults (MCAFs) and the peripheral circuit faults (PFs)? Let us assume, for instance, that the address decoder has the property of deactivating the word line at a *fixed* moment and, therefore, only ActD can occur. As mentioned above, either AC or Minimal AM has to be used (see Table 5). Inspecting the tests based on such AMs, while considering tests for MCAFs and PFs, reveals that it is better to use AC-based tests. For example, the RaW-AC test (see Table 5) can be easily combined with March C- [14] (i.e., $\{\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\}$) or with March SS [10], i.e.,

$$\{\Updownarrow (w0); \Uparrow (r0, r0, w0, r0, w1); \Uparrow (r1, r1, w1, r1, w0);$$
$$\Downarrow (r0, r0, w0, r0, w1); \Downarrow (r1, r1, w1, r1, w0); \Updownarrow (r0)\},$$

in order to cover both ADFs and MCAFs. The result will be the same tests, but then used with AC AM rather than binary AM. If, in addition to ADFs and MCAFs, one wants to also cover the PFs [26] (e.g., Slow Sense Amplifier Fault, Slow Precharge Circuit Fault), then the combined tests should be used with appropriate data-background (i.e., checkerboard or row stipe) and address direction (i.e., Fast X).

Based on the above short discussion, one can conclude that, by understanding the faults models (either related to ADFs, to MCAFs or to PFs), their detection conditions, and the freedom the march tests provide, we can combine different march tests to optimize the test time while maintaining a complete fault coverage of the target faults. However, from a practical point of view, an experiment on a large sets of chips, using many of the proposed and old tests, have to be performed in order to establish which test (or set of tests) is the most cost effective. That is the next step we will work on.

REFERENCES

[1] R.D. Adams and E.S. Cooley, "Analysis of Deceptive Destructive Read Memory Fault Model and Recommended Testing," *Records North Atlantic Test Workshop,* pp. 27-32, May 1996.

[2] R.D. Adams and E.S. Cooley, "False Write Through and Un-Restored Write Electrical Level Faults Models for SRAMs," *Proc. IEEE Int'l Workshop Memory Technology, Design, and Testing (MTDT),* pp. 27-32, 1997.

[3] M. Azimane and A.K. Majhi, "New Test Methodology for Resistive Open Defect Detection in Memory Address Decoders," *Proc. IEEE VLSI Test Symp.,* pp. 123-128, 2004.

[4] J.H. de Jonge and A.J. Smeulders, "Moving Inversion Test Pattern Is Thorough, Yet Speedy," *Computer Design,* pp. 169-173, May 1976.

[5] L. Dilillo et al., "Comparison of Open and Resistive-Open Defect Test Conditions in SRAM Address Decoders," *Proc. IEEE Asian Test Symp.,* pp. 250-255, 2003.

[6] L. Dilillo et al., "March iC-: An Improved Version of March C- for ADOFs Detection," *Proc. IEEE VLSI Test Symp.,* pp. 129-134, 2004.

[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. 6,442,085 B1, 27 Aug. 2002.

[8] E. Gizdarsky, "Detection of Delay Faults in Memory Address Decoders," *J. Electronic Testing: Theory and Applications,* vol. 16, pp. 381-387, 2000.

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

[10] S. Hamdioui, A.J. van de Goor, and M. Rodgers, "March SS: A Test for All Static Simple RAM Faults," *Proc. IEEE Int'l Workshop Memory Technology, Design and Testing,* pp. 95-100, 2002.

[11] N. Jha and S. Gupta, *Testing of Digital Systems.* Cambridge Univ. Press, 2003.

[12] M. Klaus and A.J. van de Goor, "Tests for Resistive and Capacitive Defects in Address Decoders," *Proc. Asian Test Symp.,* pp. 31-36, 2001.

[13] S. Kundu et al., "Test Challenges in Nanometer Technologies," *J. Electronic Testing: Theory and Applications,* vol. 17, nos. 3/4, pp. 209-218, 2001.

[14] M. Marinescu, "Simple and Eficient Algorithms for Functional RAM Testing," *Proc. IEEE Int'l Test Conf.,* pp. 236-239, 1982.

[15] B. Nadeau-Dostie et al., "Serial Interfacing for Embedded-Memory Testing," *IEEE Design and Test of Computers,* vol. 7, no. 2, pp. 52-63, 1990.

[16] R. Nair, "An Optimal Algorithm for Testing Stuck-At Faults Random Access Memories," *IEEE Trans. Computers,* vol. 8, no. 3, pp. 258-261, Mar. 1978.

[17] S. Nakahara et al., "Built-In Self-Test for GHz Embedded SRAMs Using Flexible Pattern Generator and New Repair Algorithm," *Proc. IEEE Int'l Test Conf.,* pp. 301-310, 1999.

[18] W. Needham et al., "High Volume Microprocessor Test Escapes, an Analysis Our Tests Are Missing," *Proc. IEEE Int'l Test Conf.,* pp. 25-34, 1998.

[19] P. Nigh and A. Gattiker, "Test Method Evaluation Experiments & Data," *Proc. IEEE Int'l Test Conf.,* pp. 454-463, 2000.

[20] J. Otterstedt et al., "Detection of CMOS Address Decoder Open Faults with March and Pseudo Random Memory Tests," *Proc. IEEE Int'l Test Conf.,* pp. 53-62, 1998.

[21] T.R.N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems.* Prentice-Hall, 1989.

[22] M. Sachdev, "Open Defects in CMOS RAM Address Decoders," *IEEE Design and Test of Computers,* pp. 26-33, Apr.-June 1997.

[23] M. Sachdev, "Test and Testability Techniques for Open Defects in CMOS RAM Address Decoders," *Proc. European Design and Test Conf.,* pp. 428-434, 1996.

[24] A.J. van de Goor, *Testing Semiconductor Memories: Theory and Practice,* ComTex Publishing, 1998.

[25] A.J. van de Goor, S. Hamdioui, and Z. Al-ars, "Tests for Address Decoder Faults in RAMs Due to Inter-Gate Opens," *Proc. European Test Symp.,* pp. 146-151, 2004.

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

Said Hamdioui received the MSEE and PhD degrees (both with honors) from the Delft University of Technology, Delft, The Netherlands. He is currently with the Delft University of Technology. He has more than eight years of experience in industry and academia as a consultant and/or as researcher and developer on test issues in general and memory testing in particular. He spent a couple of years with Intel Corporation in Santa Clara and Folsom, California, where he was responsible for developing new low-cost and efficient test solutions for advanced Intel single-port and multiport embedded cache designs in the new generations of microprocessors. In addition, he spent more than one and half years with Philips Semiconductors in France and The Netherlands, where he was responsible for driving advanced product debug and yield ramp activities and for developing systematic ways of reducing the time in yield improvement for advanced semiconductor memories. Dr. Hamdioui is the author of a book and about 40 conference and journal papers in the area of testing; many of them are the result of cooperation with industrial partners (e.g., Intel, ST, Infineon, etc). His research interests include VLSI test and reliability, deep-submicron CMOS IC design and test, systematic fault modeling, test generation and optimization for semiconductor memories, design for testability, BIST, yield enhancement, product engineering, etc. He was the recipient of the European Design Automation Association (EDAA) Award for 2001. He is a member of the IEEE.

Zaid Al-Ars received the MSc and PhD degrees in electrical engineering, both with honors, from the Delft University of Technology in Delft, The Netherlands. He is currently working at the Delft University of Technology. He has more than six years of experience with industry and academia as a consultant and a researcher on test issues in general and memory testing in particular. He spent a number of years with Infineon Technologies (former Siemens Semiconductors) in Munich, Germany, where he was responsible for constructing new test methodologies to reduce the overall costs of their test flow. He is the author of more than 30 papers in the field of electrical defect simulation, fault modeling, and test generation in memory devices, most of which are based on cooperation with international industrial partners in the semiconductor industry, such as Infineon, Intel, etc. He is a member of the reviewing committees of various international conferences and journals. His research interests include systematic fault analysis and test generation and optimization for semiconductor memory products, design for testability, built-in self-testing and repair, logic testing, diagnosis, and IC reliability, and inductive fault analysis (IFA) and yield improvement of defective ICs. Dr. Al-Ars is a member of the IEEE.

Ad J. van de Goor received the MSEE degree from the Delft University of Technology, Delft, The Netherlands, in 1965, and the MSEE and PhD degrees from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 1970. He worked with Digital Equipment Corporation, Maynard, Masachusetts, as the chief architect of the PDP-11/45 computer. He also worked for IBM in The Netherlands and in the United States, being responsible for the architecture of embedded systems. He was a professor of computer engineering at the Delft University of Technology and is currently retired. He has written two books and more than 150 papers in the areas of computer architecture and testing. He is on the editorial board of the *Journal of Electronic Testing: Theory and Applications*. His main research interests are in testing memories and logic. Professor van de Goor is fellow of the IEEE.