Tests for Address Decoder Delay Faults in RAMs due to Inter-gate Opens

Ad J. van de Goor[1]   Said Hamdioui[1,2]   Zaid Al-Ars[1,3]

[1]Delft University of Technology
Computer Engineering Laboratory
Mekelweg 4, 2628 CD Delft, The Netherlands
E-mail: A.J.vandeGoor@ewi.tudelft.nl

[2]Philips Semiconductor Crolles R&D
860 Rue Jean Monnet, 38920 Crolles, France
E-mail: S.Hamdioui@ewi.tudelft.nl
E-mail: Said.Hamdioui@philips.com

[3]CatRam Solutions
Reinier de Graafweg 188
2625 DE Delft, The Netherlands
E-mail: Zaid.Al-Ars@catram.com

Abstract

This paper presents an electrical analysis of Address decoder Delay Faults 'AFDs' caused by resistive inter-gate opens in RAMs. It introduces a systematic method to explore the space of possible tests to detect these faults. The method is based on generating appropriate sensitizing address transitions and the corresponding sensitizing operation sequences.

1. Introduction

In [1] Nigh reports 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 [2]. Needham [3] reports that opens were the most likely cause of field returns of Intel microprocessors. Klaus [4] reports 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 [5, 6, 7, 8]. 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 [4, 6, 9, 10, 11, 12]. Most authors have solved the problem of detecting *Delay Faults* in the Address decoders, denoted as '*AFDs*', by using a test called *Moving Inversion 'MOVI'* [4, 6, 11]. [10] even uses the time consuming GalPat test [5]. [9] has solved the problem by adding a decoder specific set of patterns to an existing march test.

This paper presents an analysis, at the electrical level, of AFs caused by resistive opens within the address decoder decoding paths. The paper is organized as follows. Section 2 describes the traditional AFs, together with their detection conditions. Section 3 describes the causes of address decoder delay faults, classifies them and gives a simulation example. In Section 4 the detection conditions for AFD due to intra-gate opens are presented. Section 5 derives the tests. Last, Section 6 ends with the conclusions.

2. Traditional address decoder faults (AFs)

Traditional address decoder faults have been considered for a long time the only class of AFs [5]. They are described below, together with their detection condition. However, first, the notation for 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*, from cell 0 to cell $n-1$, or a *Decreasing ($\Downarrow$) AO*, from cell $n-1$ to cell 0. When the AO is irrelevant the symbol '$\Updownarrow$' is used.

2.2 Traditional AFs and their detection condition

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

- **AFna**: An address does **n**ot **a**ccess its cell.
- **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 $h \geq 1$ [5]. 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 (from **h***ammer*) $r\overline{x}$ (rx) operations; $h \geq 0$.):

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

3. Delays in address decoder paths

Opens are the major cause of delays in the address decoder paths; they can cause *Address decoder Delay Faults 'AFDs'*. Figure 1 shows a sequence of memory accesses, sequentially accessing memory locations with a *good Word Line* 'WLg', a potentially *faulty* WL 'WLf', and last, another 'WLg'. In case of an AFD, the activation and/or the deactivation of WLf will be delayed, causing an *Activation Delay 'ActD'* fault and/or a *Deactivation Delay 'DeactD'* fault. Below, the types of resistive opens in the address decoder paths are classified, the consequences of inter-gate opens are analyzed, and a simulation example is presented.

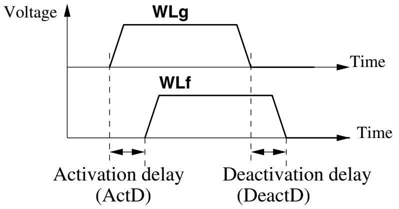

Figure 1. Activation and deactivation delays

3.1 Classification of resistive opens

Figure 2 depicts a part of a CMOS address decoder. The address bits of the three-bit address buffer a_2, a_1, a_0 are routed via the signals $a_2, \overline{a_2}, a_1, \overline{a_1}, a_0$, and $\overline{a_0}$. The decoding of the word lines ($WL0 - WL7$), also referred to as *rows*, is done using 3-input CMOS NAND gates and 2-input NOR gates, together with a buffer circuit. The address of WLx is Ax; it specifies the values of the $N = 3$ address lines 'a_2, a_1, a_0', where each of the address lines 'a_x' may have the true value (denoted as 'a_x') or the complementary value '$\overline{a_x}$'.

$WL0$ is selected if $\{\overline{a_2}, \overline{a_1}, \overline{a_0}\} = \{1, 1, 1\}$, indicating that $A0 = 000$ of $\{a_2, a_1, a_0\}$. Column decoders have a similar structure, and therefore will not be discussed here.

In the decoder of Figure 2 defects can cause opens at the following locations:

- Between logic gates (*inter-gate opens*). Figure 2 shows three inter-gate 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 (*intra-gate opens*). E.g., an open in the drain of the pull-up transistor (for input $\overline{a_1}$) in the NAND gate (not shown in Figure 2).

Klaus [4] states that the probability of inter-gate opens, caused by spot defects in the long global wiring, is at least one order of magnitude larger than that of intra-gate opens; the latter are caused by local spot defects in the short wiring within the decoder gates. Therefore we will focus on inter-gate opens.

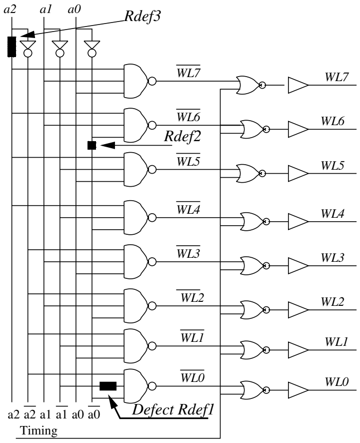

Figure 2. Typical CMOS address decoder

3.2 Inter-gate opens

For sufficiently *high* values of *Rdef1*, see Figure 2, 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 of $WL0$, 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 Figure 1, on the signal $\overline{a_1}$ of the CMOS NAND gate of $WL0$, as explained below.

Activation delay: ($0 \rightarrow 1$ change of $WL0$). The ActD is caused by a $0 \rightarrow 1$ transition of $\overline{a_1}$; see Figure 2. This can be represented by the address transition $x\mathbf{1}y \rightarrow 0\mathbf{0}0$ of $\{a_2, a_1, a_0\}$; $x, y \in \{0, 1\}$. 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.

Deactivation delay: ($WL0$ changes from $1 \rightarrow 0$). The DeactD is caused by a $1 \rightarrow 0$ transition of $\overline{a_1}$; see Figure 2. This occurs upon an address transition $0\mathbf{0}0 \rightarrow 0\mathbf{1}0$ of $\{a_2, a_1, a_0\}$. This is the address transition $WL0 \rightarrow WL2$. The consequence of DeactD will be that $WL0$ will still be active, while the next address, accessing $WLg = WL2$, 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 Figure 1.

3.3 A simulation example for ADFs

Simulations have been performed for Infineon 0.18μm *e*DRAM technology, showing the existence of AFDs for

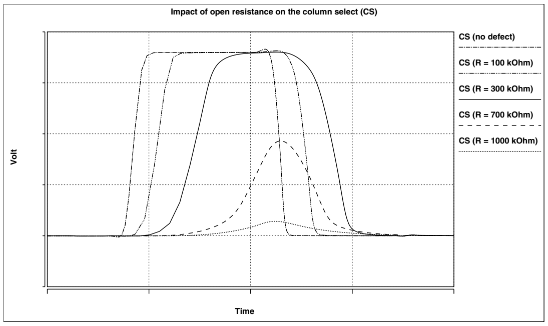

Figure 3. Impact of Rdef on DRAM CS timing

row decoder as well as for column decoders. Due to lack of space, only one example will be given here. The $Rdef$ is injected in the very last stage of the column decoder, impacting the timing of the *Column Select 'CS'* signal.

• *Opens in the column address decoder*: Figure 3 shows five CS waveforms: $Rdef = 0\Omega$, 100KΩ, 300KΩ, 700KΩ, and $Rdef = 1000$KΩ. The result of the defect is that both the ActD and DeactD (i.e., the falling edge) are gradually delayed, and the highest CS voltage reachable gradually decreases.

• *Consequences of Rdef*: For large values of $Rdef$, a read operation produces a fixed value; for intermediate values of $Rdef$ 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 $Rdef$ than read operations.

4. Detection conditions for inter-gate AFDs

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

(a) *Sensitizing address transitions*, and

(b) *Sensitizing operation sequences*.

Sensitizing address transition(s) can be generated 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 Figure 1, which have to be applied in sequence because AFDs are sensitized by *address transitions*. (Note: A_g is the address of WL_g and A_f is the address of WL_f). 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 for a SAT, rather than four addresses when the two SAPs are applied.

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 2 operations for a SAP, and 3 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 AFDs, 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 $\Downarrow^{Bin}= \{7, 6, 5, 4, 3, 2, 1, 0\}$. For the different types of AFDs (i.e., ActD and DeactD), different AMs are required, as described below.

4.1.1 Addressing methods for ActD

Section 3.2 has shown that for *inter-gate opens* 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 $\{a_2, a_1, a_0\}$; this can be represented by the SAP $\{x\mathbf{1}y, 0\mathbf{0}0\}$, with $x, y \in \{0, 1\}$. 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. The Address Complement AM satisfies these requirements.

The *Address Complement 'AC'* AM generates all 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, **111**, 001, **110**, 010, **101**, 011, **100**}, and the inverse sequence $\Downarrow^{AC}=$ {100, **011**, 101, **010**, 110, **001**, 111, **000**}; each address is followed by its one's complement (in bold-face). Note that the '$\Downarrow$' address sequence starts with address '100' because it has to be the *exact inverse* of the $\Uparrow^{AC}$ address sequence.

The AC AM satisfies the SAP requirements for ActD faults because, e.g., in the '$\Uparrow^{AC}$' AO, a_2 makes a $0 \rightarrow 1$ transition in the SAP {**0**00, **1**11}; i.e., for the gate which requires $\{000\}$ on $\{a_2, a_1, a_0\}$ in order to be enabled. The next SAP {**0**01, **1**10} does this for the gate which requires $\{001\}$ in order to be enabled, 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.

4.1.2 Addressing methods for DeactD

Section 3.2 has shown that for 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 $\{a_2, a_1, a_0\}$; this can be represented by the SAP $\{0\mathbf{0}0, 0\mathbf{1}0\}$. This is a SAP with $H = 1$, 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.

In general, for sensitizing any DeactD fault caused by an open in the path of any NAND-gate of an N-bit address decoder, all SAPs with $H = 1$ have to be generated. Only AMs which generate SAPs or SATs with $H = 1$ (i.e., the AMs 2^i and H1 described below) can detect DeactD.

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; see Table 1. 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-addres}. 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, ..., 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, ..., 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. The number of addresses to be generated for the 2^i AM is: $N_{2^i}(N) = N$(for address line 0 through $N-1$)*2^N(the number of addresses in an $\Uparrow^i$ or an $\Downarrow^i$ sequence) $*2$(because the $\Uparrow^i$ the $\Downarrow^i$ sequence have to be applied) $= 2nN$.

The SAPs of the 2^i AM satisfy the requirements for detecting DeactD fault because, e.g., in the '$\Uparrow^0$' address sequence a_0 makes a $0 \rightarrow 1$ transition in SAP {00**0**,00**1**}; i.e., for the gate which requires {000} on $\{a_2, a_1, a_0\}$ in order to be enabled. The next SAP {01**0**,01**1**} does this for the gate which requires {010} on $\{a_2, a_1, a_0\}$ in order to be enabled; etc.

H1 addressing method

This addressing method generates the *minimal* set of SATs with $H = 1$; where H stands for the *Hamming distance* between each address pair $\{Ax, Ay\}$ and $\{Ay, Ax\}$ of the SAT $\{Ax, Ay, Ax\}$.

The H1 addressing method is based on the concept of *constant weight codes* [13]; also referred to as *m-out-of-n codes*. They have the property that each n-bit *Code-Word 'CW'* contains exactly m 1's; 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., $W = 0$, $W = 2$, etc. Then for a given CW, N different SATs are generated by complementing, and thereafter re-complementing, 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/re-complementing 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 1. The 2^i addressing

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

Table 2 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; i.e., 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 $W = 0$, $W = 2$, ..., until $W = 2 * \lfloor N/2 \rfloor$. The total number of CWs, N_{CW}, is the sum of the sets of CWs; i.e., $N_{CW} = \sum_{i=0}^{\lfloor N/2 \rfloor} C_{2i}^N$. For example, for $N = 3$ (see Table 2), N_{CW}=4.

The number of addresses to be generated for the H1 AM, *for the case that SATs are used*, is: $N_{H1}(N) = 3 * N$(for each Code-Word) $*N_{CW} = 3 * N * (\sum_{i=0}^{\lfloor N/2 \rfloor} C_{2i}^N)$. The number of addresses to be generated for the H1 AM using SuperSATs is: $N_{H1_S}(N)$= 3(for the first word) + $2 * (N - 1)$(for additional words) =$(2N + 1) * (\sum_{i=0}^{\lfloor N/2 \rfloor} C_{2i}^N)$ The use of SuperSATs reduces therefore 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 $N_{H1_S}(3) = 28$.

4.2 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 Pairs* for a SAP, and *Sensitizing Operation Triplets* for a SAT;

Table 2. 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				

Table 3. 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	$\Updownarrow (wx); \Updownarrow_f \; (\Updownarrow_g^{AM} \; (rx_g, w\overline{x}_f, wx_g), wx_f)$

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 Figure 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 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 Figure 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 3. 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, '$\Updownarrow^{AM}$ $(r\overline{x}, ..., wx, rx)$' performs the *Read-after-Read 'RaR'* RWS of operations using a certain addressing method AM. 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 Figure 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.)

Depending on the selected operations in $O \in \{r, w\}$, eight RWSs for SOTs are possible (see the bottom block in Table 3): RaRaR, RaRaW, RaWaR, WaRaR, RaWaW, 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 AFD; this violates the WaWaW requirement. The RWSs for SOTs use addresses triplets 'Ag, Af, Ag' in order to allow for SATs; the first and the third address are identical. In the bottom block of Table 3, 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 a '$r\overline{x}_f$' is applied, and last a 'wx_g'.

All RWSs *ending* with an 'R' (i.e., that means that the nested ME *starts* with a 'rx_g' operation and has the form 'XaYa**R**' with X,Y ∈ {R,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 inter-gate AFD

Based on the AM of Section 4.1 and the SOPs/SOTs of Section 4.2, tests for detecting AFDs due to inter-gate opens can be constructed. The results are given in Table 4; entry #1 through #4 list the tests based AC AM, entry #5 through 8 the tests based on 2^i AM, and the entry #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.

Table 4. Tests for address decoder delay faults due to inter-gate opens

#	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	$13n$	$\{\Downarrow (w0); \Uparrow (r0,w1,r1); \Uparrow (r1,w0,r0); \Downarrow (r0,w1,r1); \Downarrow (r1,w0,r0)\}$

A: The superscript 'c' in 'wx_g^c' denotes that the operation is only applied to the first SAT of each CW 'Code Word'
Note: The ';' separates the initializing operations from the operations required by the RWS

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 the AC and the 2^i AM use SAPs and SOPs, and tests based on H1 AM use SATs and SOTs.

Four tests based on the AC AM and four based on the 2^i AM for AFDs can be distinguished, because of the four possible RWSs; see tests #1 through #8 in Table 4. The 2^i AM based tests use the notation '$i_0^{N-1}[\Uparrow^i$ ....]'. 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 [5, 7] are also based on 2^i AM. MOVI tests, included in Table 4 (i.e., test #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 low-cost version of GalPat [5], 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 4. They have been derived from the test structures of the bottom block in Table 3, 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. These operations therefore have been included in the ME which contains the operations of the RWS.

6. Conclusions

In this paper the Address decoder Delay Faults 'AFDs', due to inter-gate opens, have been analyzed. The AFDs have been divided onto *Activation Delay (ActD)* and *Deactivation Delay (DeactD)*. Thereafter a systematic method has been used to develop several tests targeting AFDs. The method is based on generating Addressing Methods 'AMs', together with the required Read-Write-Sequences 'RWSs'. Tests for ActD faults require Address Complement AM, while tests for DeacD faults require the 2^i AM or the more time efficient H1 AM.

References

[1] P. Nigh and Anne Gattiker, 'Test Method Evaluation Experiments & Data'. In Proc. of the IEEE Int. Test Conf. (ITC'00), 2000, pp.454-463

[2] S. Kundu et al., 'Test Challenges in Nanometer Technologies', Journal of Electronic Testing: Theory and Applications, Vol. 17, No. 3/4, 2001, pp. 209-218

[3] W. Needham et al., 'High Volume Microprocessor Test Escapes, an Analysis Our Tests are Missing', In Proc. of the IEEE Int. Test Conf. (ITC'98), 1998, pp.25-34

[4] M. Klaus and Ad J. van de Goor, 'Tests for Resistive and Capacitive Defects in Address Decoders', In Proc. of ATS'01, 2001, pp. 31-36

[5] A.J. van de Goor, 'Testing Semiconductor Memories: Theory and Practice', *ComTex Publishing, Gouda, The Netherlands*, 1998; ISBN 90-804276-1-6.

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

[7] J.H. de Jonge and A.J. Smeulders, 'Moving Inversion Test Pattern is Thorough, Yet Speedy', Computer Design, May 1976, pp. 169-173

[8] S. Hamdioui, 'Testing Static Random Access Memories: Defects, Fault Models, and Test Patterns', *Kluwer Academic Publishers*, ISBN 1-4020-7752-1, 2004.

[9] M. Sachdev, 'Open Defects in CMOS RAM Address Decoders', IEEE Design & Test of Computers, April - June 1997, pp. 26-33

[10] 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., 1999, pp. 301-310

[11] 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

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

[13] T.R.N. Rao and E. Fujiwara, 'Error-Control Coding for Computer Systems', Prentice-Hall International Editions, Englewoods Cliffs, NJ, 1989.