Memory Fault Modeling Trends: A Case Study*

SAID HAMDIOUI

Delft University of Technology, Faculty of Electrical Engineering, Mathematic and Computer Science, Computer Engineering Laboratory, Mekelweg 4, 2628 CD Delft, The Netherlands; Intel Corporation, 2200 Mission College Boulevard, Santa Clara, CA 95052, USA

s.hamdioui@ewi.tudelft.nl

ROB WADSWORTH

STMicroelectronics, 850 rue Jean Monnet BP 16, F-38926 CROLLES Cedex, France

JOHN DELOS REYES

Intel Corporation, 2200 Mission College Boulevard, Santa Clara, CA 95052, USA

AD J. VAN DE GOOR

Delft University of Technology, Faculty of Electrical Engineering, Mathematic and Computer Science, Computer Engineering Laboratory, Mekelweg 4, 2628 CD Delft, The Netherlands

Abstract. In recent years, embedded memories are the fastest growing segment of system on chip. They therefore have a major impact on the overall Defect per Million (DPM). Further, the shrinking technologies and processes introduce new defects that cause previously unknown faults; such faults have to be understood and modeled in order to design appropriate test techniques that can reduce the DPM level. This paper discusses a new memory fault class, namely dynamic faults, based on industrial test results; it defines the concept of dynamic faults based on the fault primitive concept. It further shows the importance of dynamic faults for the new memory technologies and introduces a systematic way for modeling them. It concludes that current and future SRAM products need to consider testability for dynamic faults or leave substantial DPM on the table, and sets a direction for further research.

1. Introduction

According to the 2001 International Technology Roadmap for Semiconductor (ITRS 2001), today's system on chip (SOC) are moving from logic dominant

*A preliminary version of this work was presented at the 8th ETW 2003 in Maastricht.

chips to memory dominant chips, since future applications requiring a lot of memory. The memory share on the chip is expected to be about 94% in 2014 [5]! As result the overall DPM level will be determined heavily by the memory DPM level.

To ensure a certain DPM level, memory test engineers must use test algorithms that are able to deal with the new defects that the new memory technologies and

process are introducing. Without such algorithms, the required DPM level, driven by market demands, cannot be achieved. New systematic defects in the new memory technologies often manifest themselves in a different way to the traditional ones. Adequate new fault models and (diagnosis) test algorithms are therefore essential.

Researchers studying the faulty behavior of memory devices have been defining *functional fault models* (*FFMs*) and developing tests to target them [1, 2, 6–10, 13–18]. However, most of the published work is limited to *static faults*, which are faults sensitized by performing *at the most* one operation; e.g., a write operation sensitizes the fault. These models were satisfactory to deal with the defects in the old technologies.

Recent published work shows that another type of faulty behavior can take place in the absence of static faults [3, 10, 12]. This faulty behavior requires *more than one operation sequentially* in order to be sensitized. For example, a write operation, followed *immediately* by a read operation, causes the cell to flip; however, if only a single write or a single read, or a read which does not immediately follow the write is performed, the cell will not flip. Faults requiring *more than one operation sequentially* in order to be sensitized are called *dynamic faults*. The traditional industrial memory tests are designed for static faults, and therefore may not be able to deal with the dynamic faults. Little has been published on dynamic faults. In [3] the existence of dynamic faults has been proven for embedded DRAMs, based on defect injection and SPICE simulation. In [10, 12], the validation of dynamic faults for embedded caches has been proven; in addition, an appropriate test for the observed dynamic faulty behavior has been proposed.

This paper shows the importance of dynamic faults for the new memory (e.g., SRAM) technologies, based on industrial experiments done at ST Microelectronics and at Intel. It gives a systematic way for modeling them; and concludes that current and future (SRAM) memory products need to consider testing dynamic faults, or leave substantial DPM on the table. The paper is organized as follows: Section 2 explains the concept of the dynamic faults; Section 3 presents industrial test results. Section 4 discusses some detected *unique faults*, from which the importance of dynamic faults will follow; it also presents a systematic way for modeling dynamic faults. Section 5 gives a general framework for dynamic faults, with which the space of dynamic faults can be studied and analyzed; while Section 6 ends with conclusions.

2. Dynamic Fault Concept

To mathematically define dynamic faults, the fault primitive concept [20] will be used. The two basic ingredients for any fault model are: (a) A list of performed memory operations, and (b) A list of corresponding deviations in the observed behavior from the expected one. Any list of performed operations on the memory is called an *operation sequence*. An operation sequence that results in a difference between the observed and the expected memory behavior is called a *sensitizing operation sequence* (S). The observed memory behavior that deviates from the expected one is called the *faulty behavior* (F).

In order to specify a certain fault, one has to specify the S together with the corresponding faulty behavior F and the read result (R) of S, in case S is a read operation or a sequence of operations with a read as last one in the sequence. The combination of S, F and R for a given memory failure is called a *Fault Primitive* (*FP*) [20], and is denoted as $\langle S/F/R \rangle$. S describes the sensitizing operation sequence that sensitizes the fault (e.g., a two successive read 0 operations from a cell containing 0 (i.e., 0r0r0)), F describes the value or the behavior of the faulty cell (e.g., the cell flips from 0 to 1), while R describes the logic output level of a read operation (e.g., a wrong value 1) in case S is a read operation, or is a sequence of operations with a read as last one in the sequence. The example given here can be thus denoted as $\langle 0r0r0/1/1 \rangle$.

By inspecting the definition of the FP concept, one can see that the difference between *static* and *dynamic* faults is determined by the number of operations required in S. Let #O be defined as the number of operations performed *sequentially* in S. For example, if a single read operation applied to a certain cell causes that cell to flip, then #$O = 1$. Depending on #O, memory faults can be divided into *static* and *dynamic* faults:

- *Static faults*: These are faults sensitized by performing *at the most* one operation; that is #$O \leq 1$. For example, the state of the cell is always stuck at *one* (#$O = 0$), a read operation to a certain cell causes that cell to flip (#$O = 1$), etc. All published traditional fault models belong to static faults.

- *Dynamic faults*: These are faults that can only be sensitized by performing more than one operation *sequentially*; that is #*O* > 1. For example, two *successive* read operations cause the cell to flip; however, if only one read operation is performed, the cell will not flip. Depending on #*O*, a further classification can be made between *2-operation dynamic* faults whereby #*O* = 2, *3-operation dynamic* faults whereby #*O* = 3, etc. It has been shown that the probability of dynamic faults decreases as the number of operations increases [4].

3. Industrial Validation of SRAM Tests

This section gives an industrial evaluation of the traditional tests as compared with the only test, March RAW [10], specifically designed to target *a few* dynamic faults. March RAW is designed to target dynamic faults sensitized in a *victim cell* by applying a *read-after-write* to the *aggressor cell* or to the victim cell [10]. The test results of DPM screening done at STMicroelectronics and at Intel for advanced SRAMs will be presented; they validate the high fault coverage of March RAW in general, and show the importance of dynamic faults which still need to be worked out.

3.1. Used Tests and Stress Combinations

In the experiment done at STMicroelectronics as well as at Intel, a set of 53 tests has been used. A *test* consists of a *base test* (*BT*) (i.e., test algorithm) applied using a particular *stress combination* (*SC*). A SC specifies the way the test is performed, and therefore influences the sequence and/or the type of memory operations. A SC consists of a combination of values of different *stresses*; e.g., addressing, databackgrounds, etc.

3.1.1. Used Base Tests (BTs). The used BTs are listed in Table 1; the test length of each BT is also included, where n denotes the size of the memory cell array, R denotes the number of rows and C the number of columns. The used march notation is explained as follows: A complete march test is delimited by the '{...}' bracket pair, while a march element is delimited by the '(...)' bracket pair. March elements are separated by semicolons, and the operations within a march element are separated by commas. Note that all operations of a march element are performed at a certain address, before proceeding to the next address. The latter can be done in either an increasing ($\Uparrow$) or a decreasing ($\Downarrow$) address order. When the address order is not relevant, the symbol $\Updownarrow$ is be used.

Table 1. List of the used base tests (BTs).

No.	BT name	Test length	Description
1	SCAN [1]	$4n$	$\{\Uparrow (w0); \Uparrow (r0); \Uparrow (w1); \Uparrow (r0)\}$
2	MATS+ [16]	$5n$	$\{\Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0)\}$
3	MATS++ [6]	$6n$	$\{\Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0, r0)\}$
4	March C− [14, 18]	$10n$	$\{\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\}$
5	PMOVI [8]	$13n$	$\{\Downarrow (w0); \Uparrow (r0, w1, r1); \Uparrow (r1, w0, r0); \Downarrow (r0, w1, r1); \Downarrow (r1, w0, r0)\}$
6	March SR [9]	$14n$	$\{\Downarrow (w0); \Uparrow (r0, w1, r1, w0); \Uparrow (r0, r0); \Uparrow (w1); \Downarrow (r1, w0, r0, w1); \Downarrow (r1, r1)\}$
7	March SS [11]	$22n$	$\{\Updownarrow (w0); \Uparrow (r0, r0, w0, r0, w1); \Uparrow (r1, r1, w1, r1, w0);$ $\Downarrow (r0, r0, w0, r0, w1); \Downarrow (r1, r1, w1, r1, w0); \Updownarrow (r0)\}$
8	March G [17]	$23n$	$\{\Updownarrow (w0); \Uparrow (r0, w1, r1, w0, r0, w1); \Uparrow (r1, w0, w1);$ $\Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0); \Uparrow (r0, w1, r1); \Uparrow (r1, w0, r0)\}$
9	March RAW [10]	$26n$	$\{\Updownarrow (w0); \Uparrow (r0, w0, r0, r0, w1, r1); \Uparrow (r1, w1, r1, r1, w0, r0);$ $\Downarrow (r0, w0, r0, r0, w1, r1); \Downarrow (r1, w1, r1, r1, w0, r0); \Updownarrow (r0)\}$
10	Hammer [19]	$49n$	$\{\Uparrow (w0); \Uparrow (r0, 10 * w1, r1); \Uparrow (r1, 10 * w0, r0); \Downarrow (r0, 10 * w1, r1); \Downarrow (r1, 10 * w0, r0)\}$
11	GalColumn	$6n + 4nR$	$\{\Uparrow (w0); \Uparrow_b (w1_b, col(r0, r1_b), w0_b); \Uparrow (w1); \Uparrow_b (w0_b, col(r1, r0_b), w1_b)\}$
12	GalRow	$6n + 4nC$	$\{\Uparrow (w0); \Uparrow_b (w1_b, row(r0, r1_b), w0_b); \Uparrow (w1); \Uparrow_b (w0_b, row(r1, r0_b), w1_b)\}$
13	WalkColumn	$8n + 2nR$	$\{\Uparrow (w0); \Uparrow_b (w1_b, col(r0), r1_b, w0_b); \Uparrow (w1); \Uparrow_b (w0_b, col(r1), r1b, w0b)\}$
14	WalkRow	$8n + 2nC$	$\{\Uparrow (w0); \Uparrow_b (w1_b, row(r0), r1_b, w0_b); \Uparrow (w1); \Uparrow_b (w0b, row(r1), r1_b, w0_b)\}$

For Hammer, the notation e.g., $10 * w1$ means that the write operation is performed 10 times successively to the same cell. For GalRow and GalColumn, the notation e.g., $row(r0, r1_b)$ means apply a $r0$ operation in an incrementing order to the cells of the row of the *base cell*, and apply $r1$ operation to the base cell *after each* $r0$ operation; a similar explanation applies to $col(r0, r1_b)$. Similarly, for WalkRow and WalkColumn, the notation e.g., $row(r0)$ means apply a $r0$ operation using an incrementing address order to the row of the base cell, and skip the base cell; a similar explanation applies to $col(r0)$.

3.1.2. Used Stresses.

Two types of stresses have been used; namely addressing and data-background stresses. The used addressing stresses consist of two types of addressing:

1. **Fast X (fx)**: Fast X addressing is simply incrementing or decrementing the address in such a way that each step goes to the next row. The sequence proceeds in the adjacent physical order of the word lines.
2. **Fast Y (fy)**: Fast Y addressing is simply incrementing or decrementing the address in such a way that each step goes to the next column. The sequence proceeds in the adjacent physical order of the bit lines.
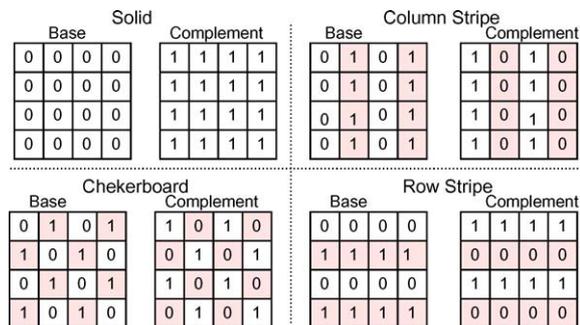
Fig. 1. The common data-backgrounds.

A *data-background* (*DB*) is defined as the pattern of ones and zeros as seen in an array of memory cells. The most common types of data-backgrounds are: Solid (s), Checkerboard (c), Column Stripe (cs), and Row Stripe (rs). Fig. 1 illustrates the four DBs, using a simple 4×4 array. Each DB is shown with the base and the complement values.

Table 2 shows the total number of tests used. The column #SC gives the number of SCs each BT is used with; and the column T.L. gives the test length of each BT. In the table, the solid, the checkerboard, column stripe and row stripe data-background are denoted as 's', 'c', 'cs', and 'rs' respectively; while the fast-X and fast-Y addressings are denoted as 'fx' and 'fy' respectively.

Table 2. List of the used BTs and their stress combinations.

| | | | | Stress combination | | | | | | | |
| | | | | fx | | | | fy | | | |
No.	Base test (BT)	# SC	T.L.	s	c	cs	rs	s	c	cs	rs
1	Scan	4	4n	−	+	+	+	−	+	−	−
2	Mats+	2	5n	+	−	−	−	+	−	−	−
3	Mats++	2	6n	+	−	−	−	+	−	−	−
4	March C−	6	10n	+	−	+	+	+	−	+	+
5	PMOVI	8	13n	+	+	+	+	+	+	+	+
6	March SR	8	14n	+	+	+	+	+	+	+	+
7	March SS	8	22n	+	+	+	+	+	+	+	+
8	March G	2	23n	+	−	−	−	+	−	−	−
9	March RAW	8	26n	+	+	+	+	+	+	+	+
10	Hammer	1	49n	+	−	−	−	−	−	−	−
11	GalColumn	1	6n + 4nR	+	−	−	−	−	−	−	−
12	GalRow	1	6n + 4nC	−	−	−	−	+	−	−	−
13	WalkColumn	1	8n + 2nR	+	−	−	−	−	−	−	−
14	WalkRow	1	8n + 2nC	−	−	−	−	+	−	−	−

A '+' in the table indicates that the corresponding SC is applied, and a '−' denoted that it is not. E.g., Mats+ is implemented two times: one using 'fx' en 's' and one using 'fy' en 's'.

3.2. STMicroelectronics Test Results

All listed BTs and their corresponding SCs in Table 2 have been implemented. From the large number of SRAM chips (with a size of 512 Kbits) tested, 1134 chips fail *all* the tests; while 60 chips fail *only some* tests. We will only concentrate on the 60 chips since they are the most interesting. The data base of the test results has been simplified and filtered for the analysis purposes. Therefore the fault coverage of each BT is considered instead of each test. The FC of a BT is the *union* of the fault coverages of its corresponding SCs. A die is considered detected by a BT if a least one SC of that BT detects the faulty die.

Table 3 gives the *unions* and the *intersections* of the used 14 BTs for *low V_{DD} and low speed* testing. A die belongs to the union of two BTs if at least one of the two BTs found the die to be faulty, and belongs to the intersection of two BTs if both BTs found the die to be faulty. The first column in each table gives the BT number; the second column the name of the BT. The column 'FC' lists the fault coverage of the corresponding BT; the column 'UFs' gives number of *unique faults* (*UFs*) each BT detects. Unique faults are faults that are only detected once with a single test; e.g., March SR detects 2 unique faults that are not detected with any other test. The union and the intersection of each pair of BTs is shown in the rest of the tables. The numbers on the diagonal give the fault coverage (FC) of the BTs, which are also listed in the column 'FC'. The part above the main diagonal lists the union of the corresponding BT pair, while the part below the diagonal lists the intersection of each BT pair. Based on the test result data base and Table 3, one can conclude the following:

1. Total number of faulty chips detected: 60.
2. The best BTs, in terms of FC, are: March G with FC = 49, March SS with FC = 48, and March SR and March C− with FC = 47.
3. There are 4 unique faults, detected by 3 tests; these are listed next together with their FC and the number of unique faults (# UFs) each detects.

BT	FC	# UFs
March SR	47	2
March SS	48	1
March G	49	1

4. The best union pair, in terms of the FC, is March SR and March SS with FC = 55.
5. There are 3 BTs which cover other BTs; e.g. March SS covers all faults found by WalkRow since their intersection is 37, which is the FC achieved with

Table 3. The union and the intersection of BTs (STMicroelectronics).

No.	BT name	FC	UFs	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Scan	34	0	**34**	44	45	51	50	49	50	53	49	43	44	41	44	42
2	Mats+	39	0	29	**39**	40	47	47	52	49	49	47	40	45	43	43	43
3	Mats++	39	0	28	38	**39**	47	47	52	49	49	47	40	46	44	44	44
4	March C−	47	0	30	39	39	**47**	49	54	51	53	50	47	50	49	48	49
5	PMOVI	46	0	30	38	38	44	**46**	54	49	52	49	47	50	49	48	47
6	March SR	47	2	32	34	34	40	39	**47**	55	54	52	52	53	50	52	50
7	March SS	48	1	32	38	38	44	45	40	**48**	53	51	49	51	50	49	48
8	March G	49	1	30	39	39	43	43	42	44	**49**	52	49	52	52	51	50
9	March RAW	46	0	31	38	38	43	43	41	43	43	**46**	47	48	47	47	47
10	Hammer	32	0	23	31	31	32	31	27	31	32	31	**32**	45	42	43	42
11	GalColumn	34	0	24	28	27	31	30	28	31	31	32	21	**34**	41	37	43
12	GalRow	35	0	28	31	30	33	32	32	33	32	34	25	28	**35**	42	39
13	WalkColumn	35	0	25	31	30	34	33	30	34	33	34	24	32	28	**35**	41
14	WalkRow	37	0	29	33	32	35	36	34	37	36	36	27	28	33	31	**37**

WalkRow. The three BTs covering other BTs are given next, with the covered BTs.

BT	Covered BTs
March C−:	Mats+, Mats++, Hammer
March SS:	WalkRow
March G:	Mats+, Mats++, Hammer

3.3. Intel Test Results

This section gives the test results based on the experiment done on Intel embedded caches with a size of 512 Kbytes. All listed SCs in Table 2 have been implemented. From the large number of chips tested, 1343 chips fail *all* the tests; while 204 chips fail *only some* tests. We will only concentrate on the 204 chips since they are the most interesting.

Table 4 gives the union and the intersections of the 14 BTs for a *high* V_{DD} *and high speed* testing; a similar representation is used as in Table 3. Based on the test result data base and Table 4, one can conclude the following:

1. Total number of faulty chips detected: 204.
2. The best BTs in terms of FC are: March G with FC = 185, March SS and March RAW with FC = 184, and March C− with FC = 183.
3. There are 11 unique faults, detected by 3 tests; these are listed next.

BT	FC	# UFs
Scan	168	1
Hammer	171	1
GalRow	176	9

4. The best union pair in terms of the FC is GalRow and March G with FC = 195, followed with GalRow and March RAW with FC = 193.
5. There are 6 BTs covering supersets fails of other BTs; these are given next, together with the covered BTs.

BT	Covered BTs
MATS++	WalkColumn
March C−	Mats++, GalColumn, WalkColumn, WalkRow
PMOVI	WalkRow
March SS	Mats+, Mats++, GalColumn, WalkColumn, WalkRow
March RAW	PMOVI, WalkRow
GalColumn	WalkColumn

It is clear from Tables 3 and 4 that March RAW, designed to target a few of the many possible dynamic faults, generally scores very good as comparison with other BTs. Further, the UFs detected by some empirical tests can not be explained with the well known fault models. Such faults need a detailed analysis in order to be understood and modeled. Some analysis examples of UFs will be given in Section 4; they will show that

Table 4. The union and the intersection of BTs (Intel).

No.	BT name	FC	UFs	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Scan	168	1	**168**	181	181	186	185	178	186	188	188	181	177	190	173	181
2	MATS+	175	0	162	**175**	179	184	183	185	184	186	186	186	180	189	179	181
3	MATS++	177	0	164	173	**177**	183	183	185	184	186	186	184	178	188	177	181
4	March C−	183	0	165	174	177	**183**	185	187	185	186	187	186	183	192	183	183
5	PMOVI	181	0	164	173	175	179	**181**	187	185	186	184	185	183	190	183	181
6	March SR	170	0	160	160	162	166	164	**170**	187	189	188	184	181	192	178	184
7	March SS	184	0	166	175	177	182	180	167	**184**	187	186	186	184	193	184	184
8	March G	185	0	165	174	176	182	180	166	182	**185**	188	188	176	**195**	186	186
9	March RAW	184	0	164	173	175	180	181	166	182	181	**184**	186	186	193	186	184
10	Hammer	171	1	158	160	164	168	167	157	169	168	169	**171**	179	192	178	183
11	GalColumn	164	0	155	159	163	164	162	153	164	163	162	156	**164**	188	164	181
12	GalRow	176	9	154	162	165	167	167	154	167	166	167	155	152	**176**	188	178
13	WalkColumn	173	0	155	156	160	160	158	152	160	159	158	153	160	148	**160**	181
14	WalkRow	181	0	155	162	164	168	168	154	168	167	168	156	151	166	147	**168**

the detected UFs are *dynamic faults*, which indicates their importance for the new memory technologies.

4. Importance of Dynamic Faults

This section analyzes some of detected UFs, and shows that those faults are dynamic faults. It further introduces a systematic way to model the faults. This will be done for ST experiment based UFs as well as for Intel based UFs.

4.1. ST Experiment Based Unique Faults

It has been shown in Section 3.2 that there are three tests which detect UFs; these faults can not be explained using the well-known fault models. This means that additional fault models and/or fault classes exist. By analyzing each test detecting UFs, new fault models will be introduced. Failure analysis can also be used for better understanding of the underlying defect causing such UF behavior. A new test(s) with a shorter test length, targeting the detected UFs, may thereafter be constructed and used for further test purposes.

The detected UFs have been analyzed during DPM screening. The results of the analysis done for March SR and March SS will be presented next; see Table 5. The march elements that were responsible for the detection of UFs, as reported by the simulator/tools during the DPM screening, are given in bold.

The analysis done for March SR points out that one UF, say UF_1 is detected by $\Uparrow$ $(r0, w1, r1, w0)$ and one UF, say UF_2, by $\Downarrow(r1, w0, r0, w1)$. The UF_1 is not detected by $\Uparrow(r0, w1)$ (of March C−), neither by $\Uparrow(r0, w1, r1)$ (of PMOVI) or any other test. The DPM screening analysis reveals that the sensitization and detection of UF_1 require the application of last operation of $\Uparrow(r0, w1, r1, w0)$ (i.e., $w0$) to the *aggressor cell* (*a-cell*) followed *immediately* with the first operation

Table 5. Analysis of UFs (ST).

BT	UFs	Description
March SR	2	$\{\Downarrow (w0); \Uparrow (\mathbf{r0}, \mathbf{w1}, \mathbf{r1}, \mathbf{w0}); \Uparrow (r0, r0);$
		$\Uparrow (w1); \Downarrow (\mathbf{r1}, \mathbf{w0}, \mathbf{r0}, \mathbf{w1}); \Downarrow (r1, r1)\}$
March SS	1	$\{\Updownarrow (w0); \Uparrow (\mathbf{r0}, \mathbf{r0}, \mathbf{w0}, \mathbf{r0}, \mathbf{w1});$
		$\Uparrow (r1, r1, w1, r1, w0); \Downarrow (r0, r0, w0, r0, w1);$
		$\Downarrow (r1, r1, w1, r1, w0); \Updownarrow (r0)\}$

of $\Uparrow(r0, w1, r1, w0)$ (i.e., r0) applied to the *victim cell* (*v-cell*). The a-cell and the *v*-cell are accessed successively; i.e., the address of the *v*-cell is the address of the a-cell plus 1. In a similar way, the analysis showed that the sensitization and detection of UF_2, by $\Downarrow(r1, w0, r0, w1)$, require the application of a w1 operation to the a-cell followed *immediately* with a r1 operation applied to the *v*-cell, where the address of the *v*-cell is the address of the a-cell minus 1. If the operations are not applied sequentially to the a-cell and the *v*-cell, the faults will be not detected, as is the case for example for March C−. It is clear that the detected UFs require the application of a *a read to the v-cell immediately after a write to the a-cell where the v-cell and the a-cell have adjacent addresses.*

A similar conclusion has been drawn based on the analysis done for March SS. The analysis done for March SS reveals that the sensitization and detection of the UF require the application of the last operation of $\Uparrow(r0, r0, w0, r0, \mathbf{w1})$ (i.e., w1) to the a-cell followed immediately with two successive read 0 operations applied to the *v*-cell; where the *v*-cell and the a-cell have adjacent addresses. The fault is only observable at the output of the memory by the second read 0 operation, although the cell may be flipped during the first read 0 operation.

The above discussed UFs are called *dynamic faults* [3, 10], as we have discussed in Section 2; they are faults sensitized by performing *more* than one operation *sequentially* (e.g., read immediately after write). Traditional tests written to cover *static faults* (i.e., faults sensitized by performing at the most one operation) do not necessarily detect dynamic faults. Establishing a complete set of possible dynamic faults together with their appropriate tests still remain to be done.

Next, a systematic way will be given in order to mathematically describe the above discussed UFs. To describe the detected UFs by March SR, the fault primitive notation will be used; see Section 2. Thus a dynamic fault can be denoted as: $\langle S/F/R \rangle = \langle S_a; S_v/F/R \rangle$, where:

$S_a; S_v$ describe *sensitizing operation sequences* applied respectively to the a-cell and to the-cell, which sensitize a fault F in the *v*-cell. Thus for UF_1 observed with March SR, S_a is a w0 operation (applied to a cell containing 1) and S_v is a read 0 operation (i.e., $S_a; S_v = 1w0; 0r0$), while for UF_2 S_a is a w1 operation (applied to a cell containing 0) and S_v is a read 1 operation (i.e., $S_a; S_v = 0w1; 1r1$).

F describes the value of the *faulty* cell (i.e., the *v*-cell); $F \in \{0, 1\}$, *R* describes the logical value which appears at the output of the memory if the sensitizing operation applied to the *v*-cell is a *read* operation: $R \in \{0, 1, -\}$.

Thus UF_1 can be denoted as $\langle 1w0; 0r0/F/1 \rangle$, and UF_2 as $\langle 0w1; 1r1/F/0 \rangle$. Note that in both notations the value of *R* differs from the expected value as described by S_v; for instance in $\langle 1w0; 0r0/F/1 \rangle$, the expected read value is 0 (since $S_v = 0r0$), but the obtained value at the output is $R = 1$; therefore the fault is detected.

In $\langle 1w0; 0r0/F/1 \rangle$, *F* can have two values: (a) $F = 0$; that means the the *v*-cell keeps its state, or (b) $F = 1$, that means that the cell flips to 1. A similar explanation can be given for $\langle 0w1; 1r1/F/0 \rangle$. Therefore there are four possible fault primitives describing the dynamic faults detected with March SR: $\langle 1w0; 0r0/0/1 \rangle$, $\langle 1w0; 0r0/1/1 \rangle$, $\langle 0w1; 1r1/1/0 \rangle$, and $\langle 0w1; 1r1/0/0 \rangle$.

In a similar way, the detected UF by March SS can be also denoted as $\langle 0w1; 0r0/1/0 \rangle$. Note that the cell will flip when the read 0 operation is applied immediately after the write 1 operation. However, the returned data output by the read operation is correct; that is due to the fact the the flipping of the cell occurs slowly. Detection of the fault requires a second read 0 operation as March SS does.

4.2. *Intel Experiment Based Unique Faults*

Section 3.3 showed that there are three tests which detect UFs. As an example, an analysis of GalRow detecting 9 UFs will be done. GalRow is given again in Table 6. The UFs are detected by '$row(r0, r1_b)$' (or by '$row(r1, r0_b)$'), which are given in bold font. The $r1_b$ and the $r0_b$ are the read data observed during the test. Thus the detection of the 9 UFs occurs during $r1_b$ (or $r0_b$).

Let's now consider the WalkRow which is also given in Table 6. The fault detection for this test occurs during '$row(r0), r1_b$' or during '$row(r1), r1_b$' (given in

Table 6. Analysis of UFs (Intel).

BT	UFs	Description
GalRow	9	$\{\Uparrow(w0); \Uparrow(w1_b, \mathbf{row(r0, r1_b)}, w0_b);$
		$\Uparrow(w1); \Uparrow(w0_b, \mathbf{row(r1, r0_b)}, w1_b)\}$
WalkRow	0	$\{\Uparrow(w0); \Uparrow(w1_b, \mathbf{row(r0)}, \mathbf{r1_b}, w0_b);$
		$\Uparrow(w1); \Uparrow(w0_b, \mathbf{row(r1)}, \mathbf{r1_b}, w0_b)\}$

bold font). If we compare the operations responsible for the fault detection for GalRow and for WalkRow, we can see that they are similar; the only difference is that by GalRow a read operation is performed to the *victim cell* (*v-cell*; i.e., base cell), *immediately* after a read operation performed to the *aggressor cell* (*a-cell*) belonging to the row of the victim cell. However for WalkRow, a read operation is applied to the *v*-cell *after* performing read operations to all the cells belonging to the row of the *v*-cell. Thus applying a read operation to the *v*-cell *immediately* after a read operation the the a-cell will sensitize (and detect) the fault, but not if the read to the *v*-cell is *not* applied immediately after a read to the a-cell. In other words, the detected UFs require the application of a *a read to the v-cell immediately after a read tothe a-cell where the v-cell and the a-cell belong to the same row*. These detected faults are called *dynamic faults* [3, 10] as discussed in Section 2.

By using the fault primitive notation, the detected UFs by GalRow can be described as $\langle 0r0; 1r1/F/0 \rangle$ and $\langle 1r1; 0r0/F/1 \rangle$, where $F \in \{0, 1\}$. That means that there are four fault primitives describing the detected UFs: $\langle 0r0; 1r1/1/0 \rangle$, $\langle 0r0; 1r1/0/0 \rangle$, $\langle 1r1; 0r0/0/1 \rangle$ and $\langle 1r1; 0r0/1/1 \rangle$.

The analysis done for some UFs show clearly that dynamic faults, which have been ignored in the past, become a very important fault class for the new memory technologies. A complete analysis of this class, in order to establish the complete fault space, remains still to be done. An experimental/ industrial analysis, using defect injection, SPICE simulation and IFA, in order to establish their probability remain still to be performed; as well as the design of the appropriate test (diagnosis) algorithms targeting such faults.

5. Dynamic Fault Framework

Now that the importance of dynamic faults has been validated experimentally, the general framework has to be determined; with the framework, the space of dynamic faults can be studied and analyzed. This section gives such framework for dynamic faults that still need to be worked out in detail. The framework assume the following two properties:

1. A sensitizing operation with *two* sequential operations; it has been shown based on defect injection and SPICE simulation, that the probability of dynamic faults decreases as the number of operations increases [4]; which means that dynamic faults with

two operations are the most important ones to concentrate on in the first stage.

2. Faults involving of a single of two cells (i.e., coupling faults). Such faults have been shown to be the vast majority of observed faults in practice for static faults [2, 3, 7, 9, 13] and therefore it expected to be also dominant for dynamic faults.

Based on the above, the (two-operations) dynamic faults can be divided into two subclasses: singe-cell and two cell faults.

5.1. Single-Cell Dynamic Faults

Single-cell dynamic faults consist of fault primitives (FPs) sensitized by applying more than one operation to a single cell *sequentially*. If will restrict our analysis to 2- operation dynamic faults (due to the previous mentioned reasons), and by using the FP notation $\langle S/F/R \rangle$, all possible faults can be determined since the space of S, F, and R are numerated. As an example consider S. It can be easy shown that there are 18 possible Ss as given below; $x, y, z \in \{0, 1\}$ and 'r' denotes a read operation and 'w' denotes a write operation.

- 8 Ss have the form '$xwywz$'; e.g., '0w1r1' denotes a write 1 operation applied to a cell whose initial state is 0; the write is followed immediately with a read 1 operation.
- 2 Ss have the form '$xrxrx$'; e.g., '0r0r0' denotes two successive read 0 operations applied to a cell whose initial state is 0.
- 4 Ss have the form '$xrxwy$'; e.g., '0r0w1' denotes a read 0 followed immediately with write 1 applied to a cell whose initial state is 0.
- 4 Ss have the form '$xwyry$'; e.g., '1w1r1 denotes a write 1 followed immediately with read 1 applied to a cell whose initial state is 1.

5.2. Two-Cell Dynamic Faults

Two-cell dynamic faults consist of FPs sensitized by applying more than one operation *sequentially* to two cells: the *aggressor* (*a-cell*) and the *v*-cell. The a-cell is the cell to which the sensitizing operation (or state) should be applied in order to sensitize the fault, while the *v*-cell is the cell where the fault appears. In a similar way as it has been done for single cell faults, and using the FP notation $\langle S/F/R \rangle$, one can determine all possible faults. Depending on how many operations are applied to the a-cell and to the *v*-cell, and on the order in which they are applied, four types of S can be distinguished:

1. S_{aa}: the two sequential operations are applied to the a-cell; while the v-cell is required to be in a certain state.
2. S_{vv}: the two sequential operations are applied to the v-cell; while the a-cell is required to be in a certain state.
3. S_{av}: the first operation is applied to the a-cell, followed immediately with a second one to the v-cell.
4. S_{va}: the first operation is applied to the v-cell, followed immediately with a second one to the a-cell.

All possible single-cell and two-cell dynamic FPs should be thus determined and compiled into functional faults model. The latter will be used to design appropriate test algorithms. These are on going work and will be the subject of upcoming papers.

6. Conclusion

This paper summarizes and analyzes the results of applying up to 17 base tests, each with up to 8 stress combinations, to advanced STMicroelectronics and Intel SRAMs in order to show the importance of dynamic faults of the recent and future memory technologies. The analysis done for detected unique faults, which can not be explained with the well known fault models, shows the existence of new dynamic faults. This indicates the importance of this fault class for new SRAM memory technologies. The dynamic faults observed in STMicroelectronics chips are different from those observed in Intel chips; this reinforces the point that a highly optimized patterns set is only applicable to a particular architecture/technology. The results also show that some tests, designed specifically to target the traditional static faults, also detect some dynamic faults.

The class of dynamic fault has been ignored in the past; it now becomes important and has to be taken into consideration for current and future memory products. This sets a direction for a further research on items like:

1. Establishing the fault space and the fault models for dynamic faults.
2. Validation based on defect injection/ SPICE simulation.
3. Inductive Fault Analysis in order to determine the importance of each introduced fault model and to

better understand the underlying defects causing such dynamic faults.

4. Design of short and high quality tests targeting and diagnosing the considered dynamic faults.
5. Industrial validation.

In fact the definition of dynamic faults presented in this paper is based on the number of successive operations required in order to sensitize a certain fault. In general the algorithms stresses (e.g., data background, addressing directions) as well as the non-algorithms stresses (e.g., timing, voltages, temperature), which have a great impact of the fault coverage, may change the nature of a fault; e.g., from dynamic to static or from three-operation to two-operation dynamic fault, etc. This is because the stress combinations influence the sequence and/or the type of memory operations, and facilitate the detection of the faults by stressing them.

References

1. M.S. Abadir and J.K. Reghbati, "Functional Testing of Semiconductor Random Access Memories," *ACM Computer Surveys*, vol. 15, no. 3, pp. 175–198, 1983.
2. R.D. Adams, "High Performance Memory Testing, Kluwer Academic Publisher, ISBN: 1-4020-7255-4, 2003.
3. Z. Al-Ars and Ad J. van de Goor, "Static and Dynamic Behavior of Memory Cell Array Opens and Shorts in Embedded DRAMs," *Proc. of Design Automation and Test in Europe*, 2001, pp. 496–503.
4. Z. Al-Ars and A.J. van de Goor, "Approximating Infinite Dynamic Behavior for DRAM Cell Defects," *Proc. IEEE VLSI Test Symp.*, 2002, pp. 401–406.
5. Allan et al., "2001 International Technology Roadmap for Semiconductors," *Computer*, vol. 35, no. 1, pp. 42–53, 2002.
6. M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Woodland Hills, CA, USA: Computer Science Press, 1976.
7. R. Dekker et al., "A Realistic Fault Model and Test Algorithms for Static Random Access Memories," *IEEE Trans. on Computers*, vol. C9, no. 6, pp. 567–572, 1990.
8. J.H. De Jonge and A.J. Smeulders, "Moving Inversions Test Pattern is Thorough, Yet Speedy," *Comp. Design*, 1976, pp. 169–173.
9. S. Hamdioui and A.J. van de Goor, "Experimental Analysis of Spot Defects in SRAMs: Realistic Fault Models and Tests," *Proc. of Ninth Asian Test Symposium*, 2000, pp. 131–138.
10. S. Hamdioui, Z. Al-Ars, and A.J. van de Goor, "Testing Static and Dynamic Faults in Random Access Memories," *Proc. of IEEE VLSI Test Symposium*, 2002, pp. 395–400.
11. S. Hamdioui, A.J. van de Goor, and M. Rodgers, " March SS: A Test for All Static Simple RAM Faults," *Proc. IEEE International Workshopon Memory Technology, Design, and Testing*, 2002, pp. 95–100.
12. S. Hamdioui, Z. Al-ars, A.J. van de Goor, and M. Rodgers, "Dynamic Faults in Random Access Memories: Concept, Fault Models and Tests," *Journal of Electronic Testing, Theory and Application*, vol. 19, no. 2, pp. 195–205, 2003.
13. V.K. Kim and T. Chen, "On Comparing Functional Fault Coverage and Defect Coverage for Memory Testing," *IEEE Trans. on CAD*, vol. 18, no. 11, pp. 1676–1683, 1999.
14. M. Marinescu, "Simple and Efficient Algorithms for Functional RAM Testing," in *Proc. of International Test Conference*, 1982, pp. 236–239.
15. S. Naik et al., "Failure Analysis of High Density CMOS SRAMs," *IEEE Design and Test of Computers*, vol. 10, no. 1, pp. 13–23, 1993.
16. R. Nair, "An Optimal Algorithm for Testing Stuck-at Faults Random Access Memories," *IEEE Trans. on Comp.*, vol. C-28, no. 3, pp. 258–261, 1979.
17. D.S. Suk and S.M. Reddy, "A March Test for Functional Faults in Semiconductors Random-Access Memories," *IEEE Trans. on Comp.*, vol. C-30, no. 12, pp. 982–985, 1981.
18. A.J. van de Goor, Testing Semiconductor Memories, Theory and Practice, ComTex Publishing, Gouda, The Netherlands, 1998.
19. A.J. van de Goor and J. de Neef, "Industrial Evaluation of DRAMs Tests," *Proc. of Design Automation and Test in Europe*, 1999, pp. 623–630.
20. A.J. van de Goor and Z. Al-Ars, "Functional Fault Models: A Formal Notation and Taxonomy," *Proc. of IEEE VLSI Test Symposium*, 2000, pp. 281–289.

Said Hamdioui received his MSEE degree with honors form Delft University of Technology, in Delft, The Netherlands, in 1997. Additionally, he received his PhD degree with honors from the same university in 2001. Since then, he has been working at the same University. He interned with Intel Corporation, USA, for about 2.5 year during and after his PhD, and was responsible for developing new low cost and efficient test algorithms for advanced Intel singeport and multi-port cache designs. His research interests concern fault modeling, failure analysis, fault simulation, memory testing, test design, DFT, BIST, BISR, etc. Hamdioui has published numerous papers in the area of testing. He is the winner of European Design Automation Association (EDAA) Doctoral Dissertation Award for 2003, and he is an associate member of the IEEE.

Rob Wadsworth entered the semiconductor industry in 1978 as an apprentice at Ferranti Electronics, England. He subsequently joined Inmos Ltd, Wales, responsible for test development and validation of their parallel processor product line, the Transputer. After Inmos was acquired by ST Microelectronics, formerly SGS-Thomson, he transferred to the ST Central Reasearch and Development Department located in Crolles, France. In this position he is responsible for developing innovative and efficient test techniques for the ST embedded memory products; carrying out this work in Europe, North America and Asia. His research interests include memory test through BIST, BISC and memory fault simulation.

John Eleazar Q. delos Reyes received his MS-ECE (Electronics and Communications Engg) degree from Saint Louis University, in Baguio City, Philippines, in 1996. He joined Intel Philippines shortly after and worked on Cache components as a Product Engr

for Pentium(R) II microprocessors. Since then, he continued to be a Product Engineer focusing on Embedded Cache Testing of Microprocessors where he relocated to Hillsboro, Oregon for Development work.

Ad van de Goor received his MSEE degree from Delft University of Technology, in Delft, The Netherlands. Additionally, he received the MSEE and PhD degrees from Carnegie-Mellon University, Pennsylvania. He worked for the Digital Equipment Corporation in Maynard, Massachusetts, as the chief architect of the PDP-11/45 computer, and for IBM in The Netherlands and in USA, being responsible for architecture of the embedded systems. Currently, he is a professor of computer engineering at Delft University of Technology. His main research interest is testing logic and memories. He has written two books and more than 120 papers in the area of computer architecture and testing. He is a member of the editorial board of the *Journal of ElectronicTesting: Theory and Applications*, and an IEEE fellow.