Address and Data Scrambling: Causes and Impact on Memory Tests

Ad. J. van de Goor
Delft University of Technology
Department of Information Technology and Systems
Section Computer Engineering
Mekelweg 4, 2628 CD Delft, The Netherlands
A.J.vandeGoor@ITS.TUDelft.nl

Ivo Schanstra
Infineon Technologies AG
Balanstrasse 73, D-81541 Munich, Germany
Ivo.Schanstra@Infineon.com

Abstract: *The way address sequences and data patterns appear on the outside of a memory may differ from their internal appearance; this effect is referred to as* scrambling, *which has a large impact on the effectiveness of the used tests. This paper presents an analysis of address and data scrambling for memory chips, at the layout and at the electrical level. A method is presented to determine the data backgrounds to be used for the different memory tests. It will be shown that the required data backgrounds are fault model, and hence, also test specific. Industrial results will show the influence of the used data backgrounds on the fault coverage of the tests.*

Keywords: *Address-scrambling, data-scrambling, data backgrounds, fault models, memory tests*

1 Introduction

The problems with testing memories are very different from testing logic. The main reason is that the fault behavior of memories is inherently analog; while the used fault models, such as stuck-at faults, typically have a digital (logical) nature. In addition, because of the internal analog operation of memories, their faulty behavior can be very complex, resulting in large classes of logical fault models [1]. Much has been published on modeling memory faults and tests [2, 3]. Most of the published tests have been designed for bit-oriented memories, which have a word-width of a single bit; i.e., $B = 1$.

However, a systematic method has been designed to cover word-oriented memories, which are more than one bit wide [4, 5]. This means that any of the published tests can be modified to be applicable to a memory with B-bit words ($B \geq 2$). The published method is based on the use of *Data Backgrounds* (*DB*s) [6], which are B-bit binary patterns written into a word. It has been shown [4, 5] that coupling faults within a word, which can occur in word-oriented memories, are detectable by march tests using the appropriate DBs. The to-be-used set of DBs is shown to depend on the assumed coupling fault model between cells within the same word of the word-oriented memory.

In addition to the modification of memory tests for covering word-oriented memories, tests also have to be modified in order to take *scrambling* into account. Scrambling means that the *logical* structure, as seen by the user from the outside of the chip, differs from the *physical* or *topological* internal structure of the chip. The consequence is that logically adjacent addresses may not be physically adjacent (this is called *address scrambling*) and that logically adjacent data bits are not physically adjacent (this is called *data scrambling*). It has been demonstrated experimentally [7, 8] that the fault coverage of a test varies by about 35% by using different addressing sequences and/or by using different DBs, while it has become an industrial practice to apply a set of test algorithms several times, every time using a different pair of DBs [9] in order to compensate for the effects of scrambling. In spite of the large impact of scrambling on the effectiveness of memory tests, no paper has been written in which the causes of scrambling, the different forms it can have, together with the consequences for the used tests, have been analyzed. These are the subject of this paper.

Section 2 describes the reasons and different forms of scrambling. Section 3 shows the consequences of scrambling for the to-be-used data backgrounds, and Section 4 for the to-be-used addressing. Section 5 shows the impact scrambling has on the fault coverage of memory tests. Section 6 ends with conclusions.

2 Reasons for scrambling

This section describes the industrial memory design and layout practices, which are the main causes for scrambling. These causes are:

1. Geometry optimisation introducing folding;
2. Address decoder optimisation;

Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA'02)
0-7695-1453-7/02 $17.00 © 2002 IEEE

3. Cell area optimisation by sharing contacts and well areas;
4. Speed and robustness optimisation based on bitline twisting;
5. Yield optimisation by introducing redundancy;
6. Achieving I/O pin compatibility utilising address or data line swap.

Strictly taken, the sharing of contacts and well areas by neighboring cells does not automatically lead to scrambling; but like scrambling, it impacts test implementation.

2.1 Folding

For a memory of a given size, the *memory cell array* (*MCA*) area has to be laid out such that it approaches a square in order to balance the lengths (and therefore the capacitances and delays) of the bitlines and the wordlines. However, this requires the number of words and the number of bits per word (referred to as B) to be about the same, which will not always be the case. For example, consider the 64*4 logical organization; the topology of such a memory would not be acceptable, because the bitlines would be too long. A much better topology would be 16*16; i.e., 16 rows, each containing 16 bits. The translation of the logical 64*4 organization to the topological 16*16 organization (indicating 16 rows with 16 bits each) is called *folding*. Several folding schemes exist, depending on the layout of the bits in a row. We will discuss the adjacent and the distributed folding schemes.

In the *adjacent* folding scheme, the B bits of the logical word are still adjacent topologically in the row; i.e., the row is filled with a set of W words, W = (Number of bits in a row)/B. For our example this means that each row contains $W = 4$ logical $B = 4$-bit words, whereby the bits of each word are physically adjacent; see Figure 1.

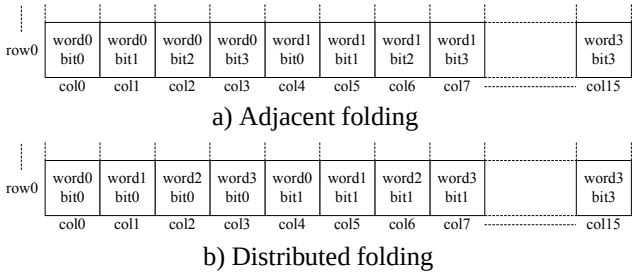

Figure 1: Adjacent and distributed folding

In the *distributed* folding scheme the B bits of the logical word are distributed over the entire row in such a way that the W bits numbered '0' of the W logical words in a row are physically adjacent, etc. The result is that two logically adjacent bits are separated by $W - 1$ bits of the other words in that row, see Figure 1.

2.2 Address decoder optimisation

Depending on their implementation, the row- and column decoders of a memory can introduce scrambling. Implementing a separate row/column decoder with all row/column addresses as imputs for every row and every column of an MCA would be very impractical for large memories; the required silicon area would be prohibitive, and it would be very difficult to make the row decoder height and column decoder width less than the height and width of a single memory cell. Fortunately, it is not necessary to spend all this silicon area, since the address decoders for individual rows and columns are for a large part identical and can thus be shared. This sharing of circuitry, in combination with further area optimisation, can lead to scrambling.

An example of this principle is shown in Figure 2. On the left-hand side of the figure, the two logical address inputs $AL1$ and $AL0$ are pre-decoded into the signals $AL1$, $\overline{AL1}$, $AL0$ and $\overline{AL0}$ by two inverters. These signals are then further decoded by local decoders, in this case simply AND gates. The local decoders on the left show a situation without scrambling; in this case, it is not possible to share any interconnect, especially contacts between different metal layers, between them. Since the physical address bits $AP1$ and $AP0$ are always identical to the logical address bits $AL1$ and $AL0$ there is no scrambling.

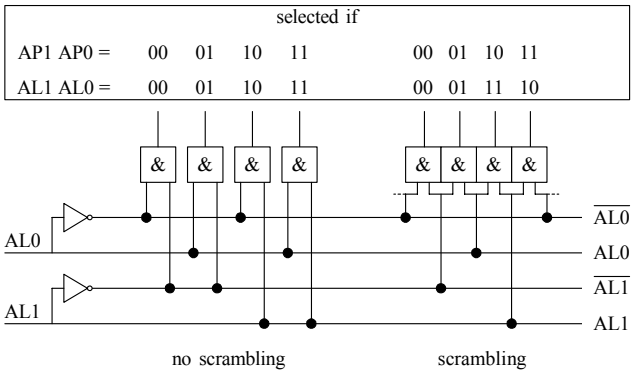

Figure 2: Address decoder scrambling

The local decoders on the right, however, have been reorganised to be able to share as many contacts as possible; this way, a smaller decoder can be built. As a side effect, scrambling is introduced. This can be seen by inspecting the differences between the logical and physical addresses, that obey the following scrambling equations, which are typical for many industrial products [10]:

$$AP0 = AL0 \oplus AL1$$

$$AP1 = AL1$$

Note: $\oplus$ stands for the XOR function and $\overline{X}$ for the negation of X. The example of Figure 2 is of course very simple; real-life address decoders usually will be much more complex.

2.3 Contact and well sharing

For all but the smallest RAMs the cell area plays an important role. In addition to using aggressive design rules, some layout measures can be taken to reduce the cell area. These measures include the sharing of contacts (such as power, ground and bitline contacts) and P- and N-well areas between groups of cells. Both contacts and well boundaries take silicon area; contacts also form a yield and reliability exposure, such that reducing their number is very desirable.

Figure 3 shows an example layout of a 6-T SRAM cell in a standard CMOS process. Only the active area, polysilicon and first metal layers are drawn; the N-well and P-well implantation areas are indicated. A transistor is formed where the poly runs over active area. This leads to two PMOS pull-up transistors ($P1$ between VDD and the true node, and $P2$ between VDD and the complement node, both in the N-well area) and four NMOS transistors; two pull-down transistors ($N1$ between VSS and the true node, and $N2$ between VSS and the complement node), and two access transistors ($N3$ between BL and the true node, and $N4$ between $\overline{BL}$ and the complement node), all in the P-well area. The true node (T) is formed in metal to the left and at the top of the figure, and in polysilicon as the gate of $P2$ and $N2$; the complement node (C) is shown in metal to the right and bottom of the figure, connecting to the poly gate of $N1$ and $P1$.

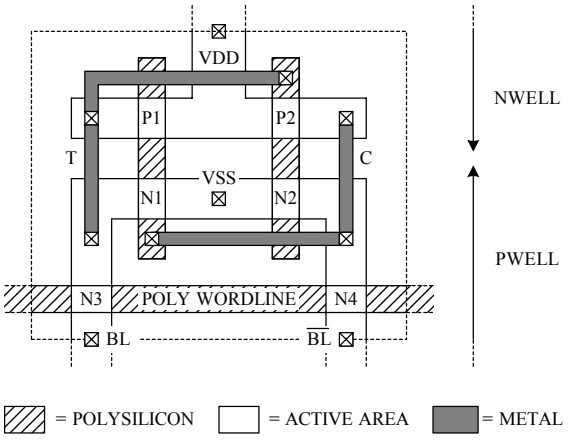

Figure 3: Example layout of a single SRAM cell

Figure 4 shows how this cell layout can be used in an MCA, saving silicon area by introducing cell *mirroring*. As can be seen from this figure, the layout of the single SRAM cell has been mirrored across the X-axis as well as across the Y-axis. In this case, the X-axis mirroring is absolutely necessary; it allows for sharing of the BL, $\overline{BL}$ and VDD contacts by two neighboring cells in the same column. In addition, the P- and N-wells can now be shared between adjacent rows. Without this sharing, twice as many P-well and N-well rows would be needed at the expense of extra silicon area, mainly due to minimum design rule distances for the P-well to N-well boundary, and to a lesser extent because of well contacts.

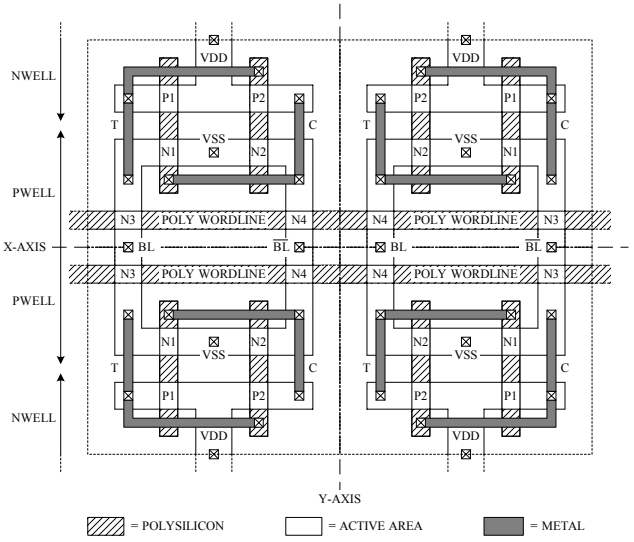

Figure 4: A group of 4 SRAM cells showing mirroring with respect to X-axis and Y-axis

In Figure 4, also Y-axis mirroring is indicated. For the layout of Figure 3, however, this is not necessary for sharing contacts or wells; it is just shown as an example. For some real-life SRAM cell layouts, however, this mirroring may be required; e.g., to share VDD or GND contacts or to optimise for certain lithography effects. Note that, although the cell layout has been mirrored accross the Y-axis, the BL is still to the left and the $\overline{BL}$ is still to the right; for a symmetric SRAM cell design, BL and $\overline{BL}$ can be chosen arbitrarily.

The layout of Figures 3 and 4 shows just one example of area optimisation by sharing contacts and well areas; many other configurations are possible, such as sharing of GND instead of VDD contacts and/or between row instead of column neighbours. This type of contact sharing has an impact on testing word-oriented memories, as will be shown in Section 3.

2.4 Bitline twisting

Bitlines in memory arrays are very sensitive to disturbs, because during read operations only a small differential voltage (in the order of 100 mV) between BL (the true bitline)

and $\overline{BL}$ (the complement bitline) is used by the sense amplifiers; a larger differential voltage would increase the read time. Because of the fact that BLs of neighboring columns are physically close together, they tend to have a large mutual capacitance and therefore a large coupling effect; see Figure 5a. It shows that $BL1$ and $\overline{BL0}$, as well as $\overline{BL1}$ and $BL2$, have a large mutual capacitance (the capacitance between $BL1$ and $\overline{BL1}$ is not drawn). Speed and/or design robustness can be improved by reducing this capacitive coupling.

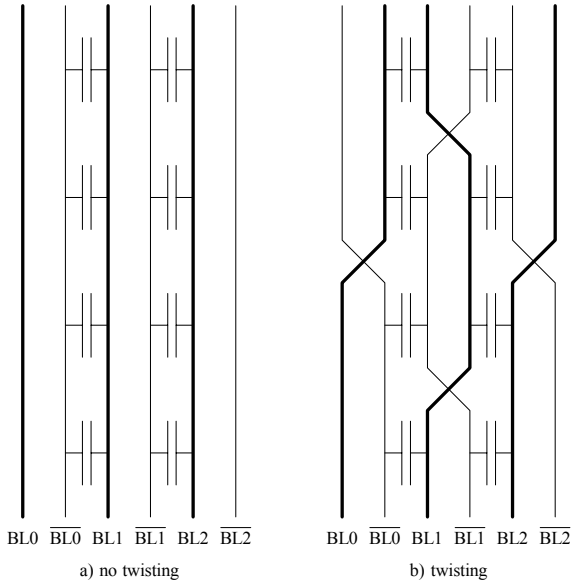

Figure 5: Bitline twisting

By *twisting* $BL/\overline{BL}$ of the even BL pairs once in the middle, and $BL/\overline{BL}$ of the odd BL pairs twice, at $1/4$ and $3/4$ of their lengths, the capacitive coupling is reduced to $1/4^{th}$, because any two BLs/$\overline{BL}$s not belonging to the same pair are only adjacent for $1/4^{th}$ of their total length. In addition, because differential sense amplifiers use the $BL/\overline{BL}$ pairs for sensing differential voltages, the effective coupling capacitance is reduced to zero, because a given BLx is coupled to the BLy and to the $\overline{BLy}$ of the neighboring BL pair; see Figure 5b. For example, $BL1$ is coupled to $BL2$ and to $\overline{BL2}$ such that a disturbing signal on $BL1$ impacts $BL2$ and $\overline{BL2}$ in the same way; i.e. the difference between $BL2$ and $\overline{BL2}$ does not change.

Twisting $BL/\overline{BL}$ means that the left-to-right order of BL and $\overline{BL}$ are switched, which does not change the array operations; i.e., it is transparent to the user of the memory. However, it changes the *adjacency* of BLs; for example $\overline{BL1}$ in the untwisted design was only neighboring $BL2$, while $BL1$ was not neighboring neither $BL2$ nor $\overline{BL2}$. In the twisted design $\overline{BL1}$ is adjacent to $BL2$ *and* $\overline{BL2}$, as well as to $BL0$ and $\overline{BL0}$, which means that a resistive defect between $\overline{BL1}$ and $BL0$, $\overline{BL0}$, $BL2$ or $\overline{BL2}$ can cause a coupling fault.

The effectiveness of twisting depends on other design properties. If, for example, a VDD or VSS line runs between adjacent BL pairs in the same metal layer, twisting does not make sense. For small memories, which typically have short BLs and strong signal levels, twisting is not used, while more complex BL twisting schemes are currently used for large memories, whereby groups of 3 or more BL pairs are twisted.

2.5 Redundancy

Chips with a larger area usually have a lower yield than smaller chips, since the larger ones are more likely to 'catch' defects. For large *memories*, the situation is even more critical, since especially for these memories the cell area has been minimized using aggressive design rules; consequentially, even smaller defects can cause opens and shorts. Given the fact that many current ASICs (and microprocessors) contain several large memory arrays, those arrays effectively determine the yield of such ASICs.

Because of the regular structure of the memory array, redundancy can be introduced in order to achieve an acceptable manufacturing yield. For example, spare rows and columns can be added, so that a limited number of defective rows and columns can be repaired [11]; any access to these defective rows/columns is rerouted to the spare ones. Typically, about 2% of the array area is allocated to spare rows/columns.

The repair can be performed statically, i.e., as part of the manufacturing process, or dynamically, every time upon power-on of the ASIC [12]. It consists of detecting and locating any defective rows/columns, calculating a repair solution if it exists and, if so, replacing them using the spares; the latter can be done by programming the repair solution into fuses (static repair) or latches (dynamic repair) that control multiplexing circuitry in the memory.

Since the spares only logically replace the defective rows/columns, using redundancy changes the relation between logical and topological neighborhoods for subsequent tests. One would expect that high quality tests would compensate for this; however, this is not the case. First of all, such tests would depend on the redundancy-concept of a certain memory type, requiring a large test development effort. Much worse, such tests would depend on the specific repair solution, which varies from chip to chip; this would require reading the repair solution from the chip and then selecting or calculating the required test, a very impractical if not impossible task. Finally, quite often many chips are tested in parallel (current DRAM chip testers allow for testing 256 chips in parallel!) by applying the same algorithm to all chips, which clearly makes a chip-specific test

impossible. Therefore, production tests typically ignore the changed topology due to repairs.

2.6 I/O pin compatibility

Commodity memory chips are specified such that they are footprint and pin compatible with similar parts of competitors. This facilitates the replacement of a given memory chip of one manufacturer by that of another. It allows for easy entry in an established market and for the availability of second suppliers (as required by many system houses).

Naturally, the internal designs and layouts of compatible chips may differ radically. The address and data pads of the chip may be connected to different internal address and data signals, since a memory remains logically equivalent if address(data) lines are swapped with other address(data) lines. The same holds for inverting the logical value of individual address(data) lines. This swapping and/or inverting, however, does influence the relation between the logical and physical organisation.

For example, in case of a chip with $B = 4$, the first manufacturer offers a chip whereby the data I/O pins of the package are numbered $D0$, $D1$, $D2$ and $D3$, bonded to the data I/O chip pads $P0$, $P1$, $P2$ and $P3$, respectively. Next, another manufacturer produces a compatible version whereby the data I/O chip pads are in the sequence: $P3$, $P0$, $P2$ and $P1$. Because of bonding restrictions, these I/O pads have to be swapped as follows: $P3 \rightarrow D0$, $P0 \rightarrow D1$, $P2 \rightarrow D2$ and $P1 \rightarrow D3$, causing data scrambling. Similarly, pin compatibility may also cause address scrambling.

3 Consequences of scrambling for data backgrounds

Some topological DBs, like *checkerboard*, are very well known; however some other DBs and the exact definition of a DB in an SRAM are less obvious. These and other issues, when using data backgrounds with scrambling, are discussed in the next sections.

3.1 Common data backgrounds

Using the proper topological DBs is important for detecting the coupling effects between cells and between BLs, and for detecting weaknesses in memory periphery [13]. Commonly the following topological data backgrounds are distinguished, see Figure 6:

- *Solid* DB (so): all memory cells are filled with '0's; and its inverse ($\overline{so}$): all memory cells are filled with '1's;
- *Checkerboard* DB (cb): an alternating pattern of '0's and '1's in both row as well as column direction (its inverse $\overline{cb}$ is not shown);
- *Row stripe* DB (rs): rows filled with '0's alternating with rows filled with '1's ($\overline{rs}$ not shown);
- *Column stripe* DB (cs): columns filled with '0's alternating with columns filled with '1's ($\overline{cs}$ not shown);
- Other DBs, such as *double row stripe* (drs), etc.

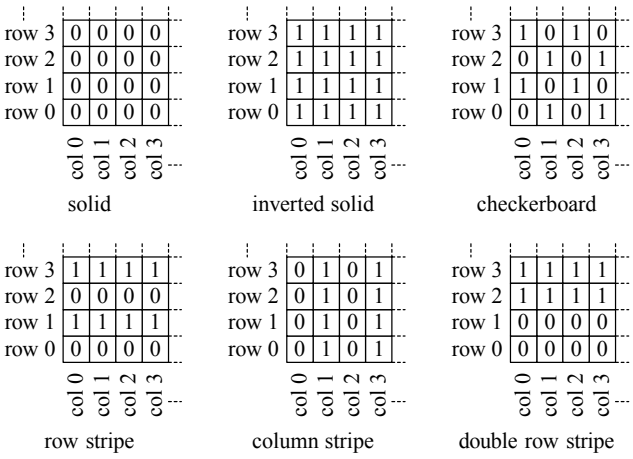

Figure 6: Some common data background types

The proper DBs to use depend on the type of fault or defect to be detected. In case of cell leakage in a DRAM, for example, it may be desirable to use the cb and $\overline{cb}$ DBs. To sensitize leakage between adjacent BLs from different columns, the so or $\overline{so}$ DB might be better suited. If the memory periphery is to be stressed, a $cb/\overline{cb}$ or $rs/\overline{rs}$ could be best, provided fast-x addressing is used (see Section 4).

Note: The usage of all of these DBs not only depends on the specific memory architecture and scrambling, but also on the used algorithms.

3.2 DBs in SRAMs with bitline twisting

For DRAMs these DBs are quite clear, but for SRAMs the situation is more complex, since each SRAM cell has two nodes storing opposite logic values, and the bitlines also run in pairs. For SRAMs, the DBs of Figure 6 define the value for the *left node* (L) (the one closest to the adjacent column with lower number); the value for the *right node* (R) is then automatically the opposite. The result is shown in Figure 7. As can be seen from this figure, leakage or coupling effects between adjacent cells in the same row are better sensitized using the rs than the cb topological DB.

The importance of defining *topological* DBs based upon node location, using L and R nodes, instead of on stored

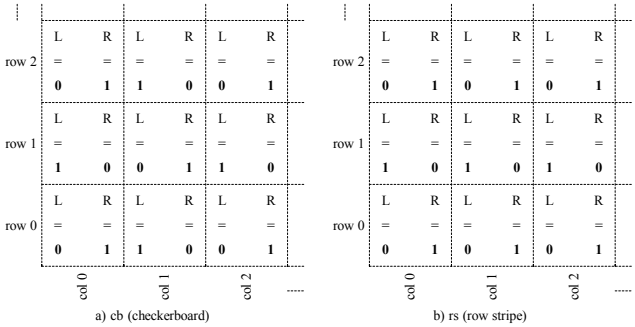

Figure 7: The cb and rs DBs at SRAM cell node level

bit value, using T and C nodes, can be seen in Figure 8. It shows the situation for a rs DB depending on bitline twisting; the stored bit value is indicated as the large digit in the center of each cell. As can be seen in part b of this figure, the stored bit value at a certain row and column has to be modulated with the BL twisting information to obtain the desired rs DB; e.g., to sensitize leakage paths. If in an SRAM a DB is used without correcting the logic bit value for twisting, the DB is called *pseudo-topological*.

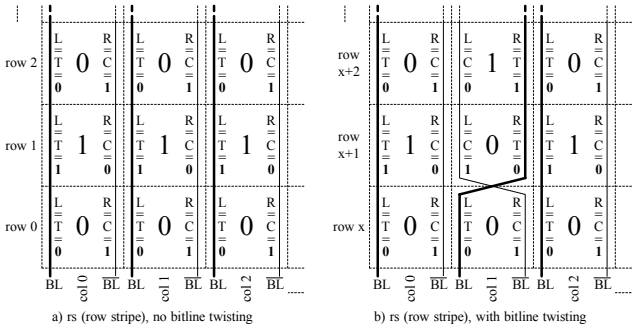

Figure 8: Stored bit value and node values for the rs DB at depending on BL twisting

3.3 Which DBs to use with bitline twisting

If all scrambling information is available, the test can simply compensate for this by modifying logical data values to obtain the desired topological DBs. Note that some tests and DBs, however, are not aimed at the topological cell values. A test for SRAM access transistor leakage applied to an SRAM with BL twisting will require the same logical value on all $T(C)$ nodes in a column, not on all $L(R)$ nodes! Similarly, a test for shorts between data lines applied to a memory with distributed folding, needs to take the logical values of the data lines of the memory into account, regardless of topological DBs.

Taking bitline twisting into account in a test implementation can be easy with suitable memory ATE. When using BIST or CPU-based memory test, however, this can be much more difficult. Instead, a set of pseudo-topological DBs (see Section 3.2) can be used to replace a topological DB. For example, in case of the twisting scheme of Figure 5, the rs topological DB that covers cell neighborhood effects can be replaced as follows:

- two adjacent cells in the same column, with no BL twist in between, need the rs or cb pseudo-topological DB;
- two adjacent cells in the same column, with a BL twist in between, need the so or cs DB;
- two adjacent cells in the same row, with the same T and F node orientation, need the so or rs DB;
- two adjacent cells in the same row, with opposite T and F node orientation, need the cs or cb DB.

Therefore, by using either the so and cb, or the rs and cs pseudo-topological DBs, any form of twisting is accounted for, including no twisting! Then the test programmer does not have to take the details of twisting into account.

To check whether any coupling, resistive as well as capacitive, between any pair of BLs exists (in case of adjacent folding), the so or rs, and the cs or cb pseudo-topological DBs have to be used, because even in the case of no twisting, they generate all 4 states on any two neighboring BLs. Therefore, no additional DBs are needed.

3.4 Effects of contact sharing and folding

In case VDD and/or GND contacts are shared between cell pairs, coupling via a resistive VDD or GND contact is possible and special attention, in terms of to-be-used DBs, is required. Figure 9 shows two examples of cells sharing the GND contacts.

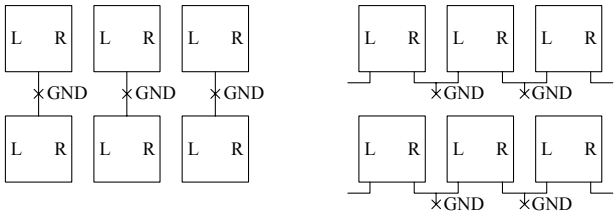

Figure 9: Sharing of GND contacts

Figure 9a shows how contact is shared between two cells in the same column; faults can now occur because writing or reading to one cell may sensitize a fault in the other cell.

In Figure 9b each cell has two GND contacts; the L and R invertors, which form a cell latch, each have their

own GND, which are shared with their row neighbors (see also Section 2.3). In case of a word-oriented memory, using adjacent folding (see Section 2.1), two cells sharing their GND contact will be accessed simultaneously. This means that a subset of the faults typical for two-port memories applies [14]. When reading two neighboring cells simultaneously, whereby the R node of the left cell and the L node of the right cell both contain a '0', a (deceptive) read destructive fault [15] can be sensitized.

4 Consequences of scrambling for addressing

The common addressing schemes, as well as the consequences of decoder srambling and folding, are discussed in the next sections.

4.1 Common addressing schemes

In formal test algorithms [2] the address is usually considered to be one-dimensional, such that only the distinction whether to increment (an $\Uparrow$ address order) or to decrement (a $\Downarrow$ address order) the current address can be made. Note that when the address order is irrelevant, as is often the case when the memory has to be initialized, then the $\Updownarrow$ symbol is used. However, in reality a memory cell has a two-dimensional address: the X-address, specifying the row of the cell, and the Y-address, specifying the column of the cell; and many ways of counting exist, denoted as *address sequences*. An almost infinite number of address sequences exist; however, for testing memories the following has become an industry accepted set. Some of the address sequences are shown in Figure 10, using a 4*4 cell array. The number in each cell indicates the step in the address sequence; i.e., the address sequence goes from step 0, to 1, to 2, through step 15.

1. xA: Fast-X addressing.
The row address is incremented most frequently. Figure 10a shows Fast-X addressing using an $\Uparrow$ address order; denoted as x$\Uparrow$. Figure 10b shows x$\Downarrow$, its inverse.

2. yA: Fast-Y addressing.
The column address is incremented most frequently, see Figure 10c.

3. gA: Gray code addressing.
The successive addresses differ in exactly one bit. Many Gray codes are possible , depending on which bit is considered the least significant address bit. An example is shown in Figure 10d. It depicts the following address sequence (address bit order: X-MSB, X-LSB, Y-MSB, Y-LSB): '0000', '0001', '0011', '0010', '0110', '0111', '0101', '0100', '1100', '1101', '1111', '1110', '1010', '1011', '1001', '1000'.

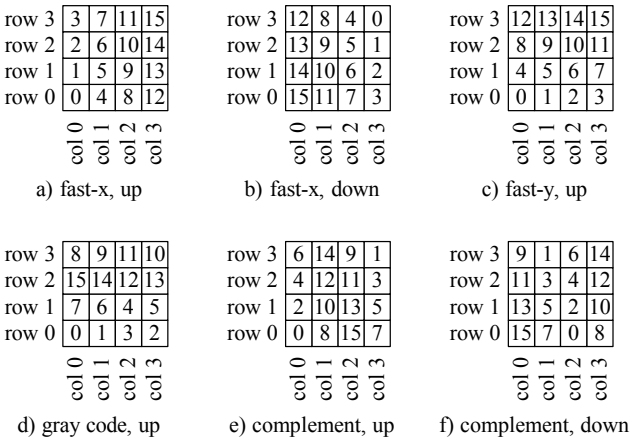

Figure 10: Some common addressing sequences

4. cA: Address complement addressing
Address increments/decrements of 1 are assumed for the even address steps, whereby in every second address step the inverse of the preceding address is used. Figure 10e shows Fast-X cA, denoted by the cx$\Uparrow$ symbol; Figure 10f shows its inverse cx$\Downarrow$

Fast-X and Fast-Y addressing usually use increments/decrements of 1, because of the high probability of coupling faults between cells who are topological row and/or column neighbors. Fast-X and Fast-Y addressing using address increments/decrements of 2^i are used to detect open faults in the address decoder paths [2, 16]. Gray code addressing is essential for testing asynchronous memories because its address sequence contains the worst-case patters for triggering the address-transition detection logic. Address complement is used to check for worst-case delays in the address decoding paths, because all predecoder gates and the local decoder gate have to switch for each new access.

4.2 Consequences of decoder scrambling and folding

Implementing a test using suitable memory ATE makes it easy to take decoder scrambling into account; the test algorithms can simply be programmed using the logical X- and Y-addresses (that exist separately because of folding) as if there were no scrambling, after which the address decoder scrambling of the to-be-tested memory can be be programmed separately into the ATE. The ATE then takes care of the correct logical to physical address translations.

In a BIST or CPU-test implementation, care must be taken to generate the desired physical addressing sequences. Partly, this can be done by just selecting carefully which address bits are connected to which bits (MSB-LSB) of an address counter; this way, basic fast-X or fast-Y addressing

can be selected. In addition, the scrambling equations (see Section 2.2) need to be implemented.

Figure 11a shows an example 8*2 bit memory. It has adjacent folding, a straightforward Y (column) decoder, and an X (row) decoder that has the scrambling of Figure 2. The logical operations to be performed on this memory in order to write a correct topological DB, are shown in Figures 11b (for x⇑ addressing) and 11c (for y⇑ addressing).

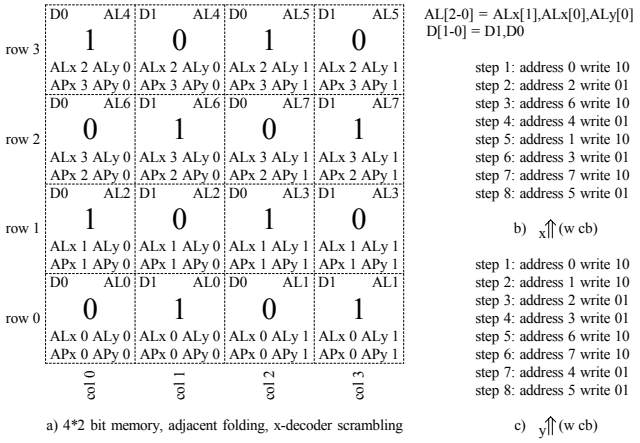

Figure 11: Example 8*2bit memory with logical operation sequences for x⇑ and y⇑ addressing with the *cb* DB

As can be seen from these figures, the exact logical operations depend on the scrambling properties of a memory, and thus are design dependent. Other desired address sequences, or distributed folding, or possible bitline twisting will result in different logical operation sequences to be performed.

5 Importance of scrambling

Industrial data, published in [7, 8], has shown the impact scrambling has on the fault coverage. In addition, a strong relationship between the fault models and the data backgrounds has been shown to exist. An experiment was performed whereby 800 4M*4 (16 Mbit) DRAM chips were tested with a large number of tests, which identified a total of 116 chips to be faulty. Many test were used in that experiment, of which the results of the following tests will be shown:

SCAN ($4n$): {⇕(w0); ⇕(r0); ⇕(w1); ⇕(r1)}
March C- ($10n$): {⇕(w0); ⇑(r0,w1); ⇑(r1,w0); ⇓(r0,w1); ⇓(r1,w0); ⇕(r0)}
PMOVI ($13n$): {⇓(w0); ⇑(r0,w1,r1); ⇑(r1,w0,r0); ⇓(r0,w1,r1); ⇓(r1,w0,r0)}
March LA ($22n$): {⇕(w0); ⇑(r0,w1,w0,w1,r1); ⇑(r1,w0,w1,w0,r0); ⇓(r0,w1,w0,w1,r1); ⇓(r1,w0,w1,w0,r0); ⇓(r0)}

The above tests have been performed using the so, cb, rs, cs, drs (see Figure 6), *double checkerboard* dcb and *double column stripe* dcs (not shown in Figure 6, see [7, 8]) DBs.

The results of these tests, using fast-X and fast-Y addressing are shown in Figure 12. For the above 7 DBs and the *union* of the 7 DBs; while address and data scrambling is compensated for. The *union* represents the collective fault coverage over all used DBs.

Variations of about 35% can be observed between fast-X and fast-Y; as well as between the different DBs for fast-X and fast-Y. Note that the fault coverage generally is highest for fast-Y, because that stresses the data retention of the DRAMs most, because all words of the same row will be accessed before accessing, and therefore also refreshing, the next row. In case of a very long sequence of operations per row, this will have to be interrupted for refresh.

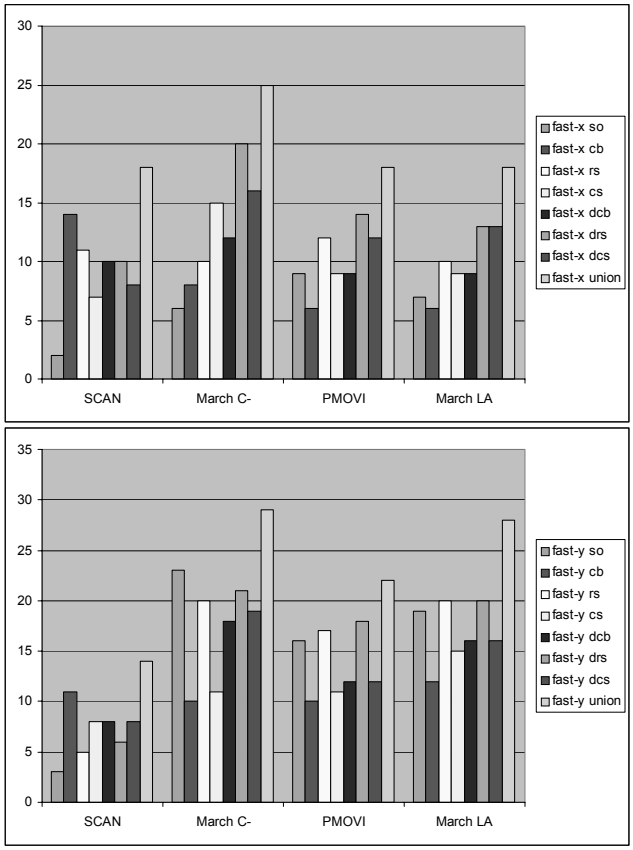

Figure 12: Effect of DBs on the fault coverage, using Fast-X and Fast-Y addressing

It should be noted that the SCAN test is a very special march test in the sense that it is well able to detect dynamic (i.e., speed related) faults, caused by slowness of the sense amplifiers, the write drivers and/or the address decoders. In order to detect these faults, the test should be performed using fast-X, and the DB should have alternating

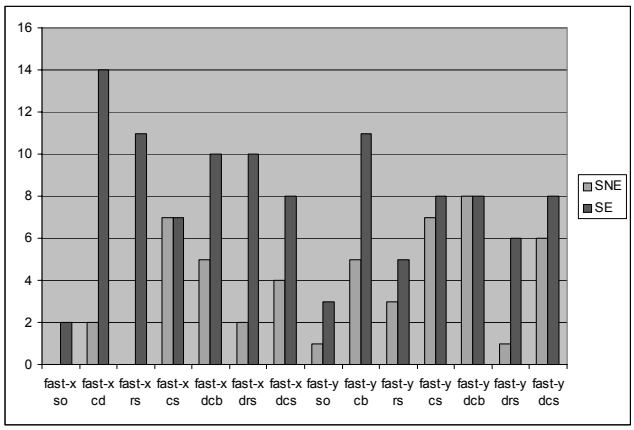

Figure 13: Fault overage of SCAN test as a function of DB, addressing order and scrambling

values along a column (this can be the rs or cb DB). The impact of the DBs and the address sequences on the fault coverage of SCAN is shown in Figure 13. In addition the fault coverage is shown for two applications of each test: the light-grey bars (*SNE*) show test results whereby scrambling has not been compensated for (by using only logical addresses and data patterns), and the dark-grey bars (*SE*) show the tests results whereby scrambling has been taken into account. From Figure 13 it is obvious that scrambling should be compensated for. In addition, the rs and cb DBs are the most effective, as could be expected, considering the properties of the fault model.

6 Conclusions

This paper has shown that address as well as data scrambling occurs as part of the normal design practice of RAMs. Industrial results have shown variations in fault overage of about 35% due to the use of different address orders and due to the use of different DBs. The impact of the used address sequence and DB has also shown to be test specific; for example, for the SCAN test the fast-X address sequence has to be used together with the cb or the rs DB. In addition, it has been shown that the fault coverage is reduced considerably when scrambling is not taken into consideration (see Figure 13).

The effects of bit-line twisting and contact sharing can be compensated for by repeating tests using different DBs. It has become industrial practice to apply a set of tests, using different address sequences as well as different DBs. The fast-X and fast-Y address sequences, and the so, cb, rs and cs DBs are used most often; they have shown to cover most cases of data scrambling; provided that address scrambling, as well as data scrambling due to I/O pin compatibility, is compensated for.

References

[1] A.J. van de Goor and Z. Al-Ars. Functional Memory Faults: A Formal Notation and a Taxonomy. In *Proc.* 18^{th} *VLSI Test Symposium*, pages 281–289, Montreal, Canada, April 2000.

[2] A.J. van de Goor. *Testing Semiconductor Memories, Theory and Practice*. ComTex Publishing, Gouda, The Netherlands, 1998, http://ce.et.tudelft.nl/∼vdgoor/.

[3] A.J. van de Goor, G.N. Gaydadjiev, V.N. Yarmolik, and V.G. Mikitjuk. March LA: A Tests For All Linked Memory Faults. In *Proc.* 7^{th} *Asian Test Conference*, pages 1–8 of supplement, Singapore, December 1998.

[4] A.J. van de Goor, I.B.S. Tlilli, and S. Hamdioui. Converting March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories. In *Records of IEEE International Workshop on Memory Technology, Design and Testing*, pages 46–52, San Jose, CA, USA, August 1998.

[5] A.J. van de Goor and I.B.S. Tlilli. March tests for word-oriented memories. In *Proc. Design Automation and Test in Europe*, pages 501–508, Paris, France, February 1998.

[6] R. Dekker, F. Beenker, and L. Thijssen. Fault Modeling and Test Algorithm Development for Static Random Access Memories. In *Proc. IEEE International Test Conference*, pages 343–352, September 1988.

[7] A.J. van de Goor and J. de Neef. Industrial evaluation of DRAM tests. In *Proc. Design Automation and Test in Europe*, pages 623–630, Munich, Germany, March 1999.

[8] A.J. van de Goor and A. Paalvast. Industrial evaluation of DRAM SIMM tests. In *Proc. IEEE International Test Conference*, pages 426–435, Atlantic City, NJ, USA, October 2000.

[9] P.G. Shepard. Programmable Built-In Self-Test Method and Controller for Arrays. USA Patent Number 5,633,877, 1997.

[10] N. Kirschner. An Interactive Descrambler Program for RAMs with Redundancy. In *Proc. IEEE International Test Conference*, pages 252–257, 1982.

[11] N. Hasan and C.L. Liu. Minimum Fault Coverage in Reconfigurable Arrays. In *Digest of Papers of the* 18^{th} *Annual International Symposium on Fault-Tolerant Computing*, pages 348–353, Kyoto, Japan, 1988.

[12] S. Nakahara, K. Higeta, M. Kohno, T. Kawamura, and K. Kakitani. Built-in Self-Test for GHz Embedded SRAMs Using Flexible Pattern Generator and New Repair Algorithm. In *Proc. IEEE International Test Conference*, pages 301–310, Atlantic City, NJ, USA, September 1999.

[13] H.I. Schanstra and A.J. van de Goor. Industrial Evaluation of Stress Combinations for March Tests applied to SRAMs. In *Proc. IEEE International Test Conference*, pages 983–992, Atlantic City, NJ, USA, September 1999.

[14] S. Hamdioui, A.J. van de Goor, M. Rodgers, and D. Eastwick. March tests for realistic faults in two-port memories. In *Records of IEEE International Workshop on Memory Technology, Design and Testing*, pages 73–78, August 2000.

[15] R.D. Adams and E.S. Cooley. Analysis of a Deceptive Destructive Read Memory Fault Model and Recommended Testing. In *Proc. IEEE North Atlantic Test Workshop*, May 1996.

[16] M. Klaus and A.J. van de Goor. Tests for Resistive and Capacitive Defects in Address Decoders. To be published in *Proc.* 10^{th} *Asian Test Conference*, Kyoto, Japan, November 2001.