A Simulation-Based Methodology for SRAM Defect Analysis and Diagnosis

Rei-Fu Huang, Jen-Chieh Yeh, Chih-Wea Wang, Chih-Tsun Huang, and Cheng-Wen Wu
Laboratory for Reliable Computing (LaRC)
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013, ROC

Abstract

We propose a methodology for systematically injecting defects into an SRAM and simulating the effects. The detect-to-fault mapping tables can be constructed after the simulation. With such tables, memory diagnosis for design debugging and yield improvement can be done more efficiently. The analysis of the circuit defects helps the memory designers and process engineers locate the weakness of the memory chips. The defect and fault detectability obtained from the analysis also can be used for optimizing the test algorithm and the redundancy structure of the memory. Experimental results show that using a March 17*N algorithm for a typical SRAM design, we found only 2.47% of the defects that have escaped the test.*

1. Introduction

As semiconductor memory technologies and the level of integration continue to improve, the challenge of testing and diagnosing advanced memory chips and cores (in system chips) grows. Specifically, an efficient memory diagnosis methodology for yield enhancement has been considered critical for system-on-chip (SOC) [1–5]. Fault-model based test and diagnosis algorithms [1,5–8] and physical defect analysis [9–11] for semiconductor memories have been studied recently. It is noted that a defect may be covered by different faults, and it is also possible for different defects to be modeled by the same fault [8, 9, 12]. The complex relationship between defects and faults makes memory defect diagnosis difficult.

In this paper, we propose a systematic approach to injecting defects into an SRAM and analyzing the defect effect by simulation. The analysis includes the establishment of the relationship between the defects and functional faults by the *defect mapping tables*. By injecting the transistor open, node short, transistor stuck-on, transistor stuck-off, and node break defects (to be defined in Sec. 3) into the SRAM cell array, we can establish their relationship with the functional fault models. Therefore, the defect table can be used to diagnose the defective memories by using the memory test algorithm which is derived for the functional fault models. The approach can be used for different memory designs and manufacturing processes, though we show only an SRAM case here. Experimental results show that using the March 17*N* algorithm for the specific memory design and defects, we found only 2.47% of the defects that have escaped the test.

2. Simulation for Defect Analysis

Defect injection at the layout level usually results in a violation of the design rules, preventing the EDA tools from accurate circuit extraction. So far as circuit parameters can be obtained, we are able to perform defect analysis at the circuit level. A circuit simulator is used to provide the faulty behavior of the defective memory. For simplicity, we consider only the following defects in our experiments: open, short, stuck-on, stuck-off, and break defects (to be defined in Sec. 3). The approach can be extended to other defects in the memory.

Figure 1 shows the major steps and tools of the proposed Defect ANalysis and DIagnosis (DANDI) system. Given the RAM cell array circuit and the test algorithm, DANDI automatically injects defects into the RAM. The details of defect injection and circuit simulation will be described in Sec. 3. The memory layout can be used to provide additional information, reducing the number of defects required for injection and simulation. After defect injection, a circuit simulator is used for simulating the given March test algorithm, and the *March signatures* (also called *traces* or *syndromes*) [1, 2, 13, 14] of the faulty memory will be derived. A March signature for a fault under a test algorithm is a vector that records the Read operations in the test which detect the fault. The signatures of the functional faults are then compared with the signatures obtained from the injected defects. We group the defects that result in the same signature with a functional fault into an equivalent defect class. Using the signatures, a faster switch-level simulator instead of a circuit-level simulator (such as SPICE) can be used [15], greatly reducing the simulation time. After comparing the defect signature with the fault signature, the defect mapping tables can be constructed. By looking up the defect mapping tables, possible defects that lead to a certain fault can be identified. Moreover, based on the analysis result more realistic functional faults can be defined for different memory circuits, and unrealistic faults can be dropped. The information also helps one optimize the test algorithm. Finally, the simulation result also provides the number of the first detected defects for each Read operation, which can be used to further shorten the test algorithm under the specified test constraints.

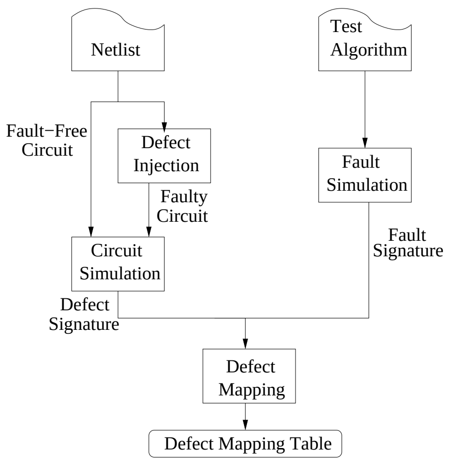

Figure 1. DANDI: the proposed defect analysis and diagnosis system.

3. Defect Injection and Circuit Simulation

Enumerating all possible defects is not possible. We consider only a few typical defects at the transistor level, but the list of defects in DANDI can easily be updated. Only a single defect is injected at a time, which can be a (transistor) open, (node) short, (transistor) stuck-on, (transistor) stuck-off, or (node) break defect. The *open* defect is defined as an open gate, source, drain, or body for a MOS transistor. This defect is modeled by inserting a *large* resistance (that is user-specified) as exemplified in Fig. 2(A). The *short* defect is defined as a resistive short between two nodes, as exemplified in Fig. 2(B). It can be an intra-cell short or inter-cell short. A *small* resistance (also user-specified) is inserted between the two nodes to model the defect. Examples of the *stuck-on* and *stuck-off* defects (including their simulation models) are shown in Fig. 2(C) and (D), respectively.

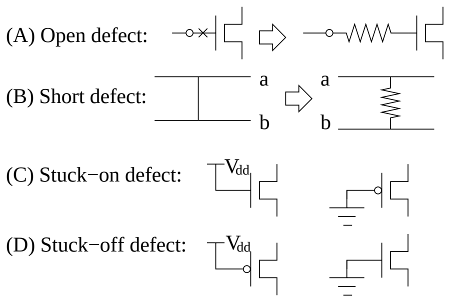

Figure 2. Examples of the open, short, stuck-on, and stuck-off defects.

Figure 3 shows a *break* defect example. Note that the circuit netlist is represented by nodes and devices. Here a *node* connects multiple devices as shown in the figure, and a break defect partitions the devices into two groups (i.e., divides the original node into two nodes). All possible ways of grouping will be considered. Let $x_1, x_2, \ldots, x_n$ be the devices connected by the node, where $n > 3$, then the number of break defects with respect to the node is $\sum_{k=2}^{\lfloor n/2 \rfloor} C_k^n$. When one of the groups consists of a single device, the break defect actually reduces to an open defect as defined above. Since there are numerous ways to designing the layout for the SRAM cell, our defect injector explores all the possible defect locations, or ways to divide the node.

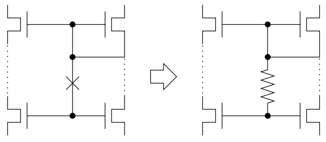

Figure 3. A break defect example.

4. Analysis and Diagnosis

We use a 4×4 bit-oriented SRAM as an experimental case for deriving the relationship of the defects and faults. An SRAM cell is shown in Fig. 4. The RAM diagnostic algorithm we use for this case is obtained using the tool described in [13], which is a March 17*N* algorithm:
$\Uparrow (w0), \Uparrow (r0, w1, r1), \Uparrow (r1, w0, r0), \Uparrow (r0, w1), \Downarrow (r1, w0, r0), \Uparrow (r0), \Downarrow (r0, w1, r1), \Uparrow (r1)$.

Table 1 shows the March signature-based fault dictionary [6, 13] of the March 17*N* test algorithm. It lists the March signatures of the conventional functional faults, including the address-decoder fault (AF), idempotent coupling fault (CFid), inversion coupling fault (CFin), state coupling fault (CFst), read disturb fault (RDF), stuck-at fault (SAF), stuck

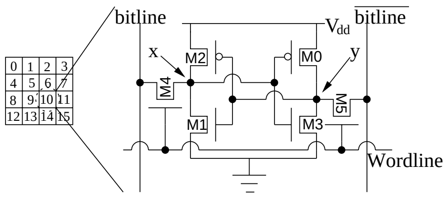

Figure 4. Schematic of an SRAM cell.

open fault (SOF), and transition fault (TF). The two parameters inside the parentheses of a coupling fault indicate the operations/states of the aggressor and victim cells, respectively. The symbols D, U, 0, 1, and $\sim$ represent the

Table 1. Fault dictionary of the March $17N$ test algorithm.

Fault Type	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
AF<	0	.	1	.	0	.	.	.	.	.	.
AF>	.	.	.	.	.	1	.	.	0	.	.
CFid(D;0)<	.	.	1	.	.	.	.	.	.	.	.
CFid(D;0)>	.	.	.	.	.	1	.	.	.	.	.
CFid(D;1)<	.	.	.	.	.	.	.	0	0	.	.
CFid(D;1)>	.	.	.	.	0	.	.	.	.	.	.
CFid(U;0)<	.	.	.	.	.	.	.	.	.	.	1
CFid(U;0)>	.	.	1	.	.	1	.	.	.	.	.
CFid(U;1)<	0	.	.	.	0	.	.	.	.	.	.
CFid(U;1)>	.	.	.	.	.	.	.	.	0	.	.
CFin(D;∼)<	.	.	1	.	.	.	.	0	0	.	.
CFin(D;∼)>	.	.	.	.	0	1	.	.	.	.	.
CFin(U;∼)<	0	.	.	.	0	.	.	.	.	.	1
CFin(U;∼)>	.	.	1	.	.	1	.	.	0	.	.
CFst(0;0)<	.	.	1	.	.	.	.	.	.	1	.
CFst(0;0)>	.	1	.	.	.	1	.	.	.	.	.
CFst(0;1)<	.	.	.	0	.	.	.	0	0	.	.
CFst(0;1)>	0	.	.	.	0	.	0	0	.	.	.
CFst(1;0)<	.	1	.	.	.	1	.	.	.	.	1
CFst(1;0)>	.	.	1	.	.	.	.	.	.	1	1
CFst(1;1)<	0	.	.	.	0	.	0	.	.	.	.
CFst(1;1)>	.	.	.	0	.	.	.	.	0	.	.
RDF(0)	.	.	.	.	0	.	.	0	0	.	.
RDF(1)	.	.	1	.	.	.	.	.	.	.	1
SAF(0)	.	1	1	.	.	1	.	.	.	1	1
SAF(1)	0	.	.	0	0	.	0	0	0	.	.
SOF1	0	1	.	0	.	1	.	.	0	.	.
SOF2	0	.	1	.	.	1	.	.	0	.	.
SOF3	0	.	1	.	.	1	.	.	.	1	.
TF(D)	.	.	.	0	0	.	0	0	0	.	.
TF(U)	.	1	1	.	.	1	.	.	.	1	1

down transition operation, up transition operation, constant 0 state, constant 1 state, and inversion operation, respectively. The symbols $>$ and $<$ stand for the relative position between the aggressor and victim cells: the former means that the address of the aggressor is higher than that of the victim, and the latter means the other way around. For example, CFid(D;0)$<$ denotes the CFid that occurs when there is a down transition in the aggressor that has a lower address than the victim, and the content of the victim is forced to 0. Each column in the table represents one of the 11 Read operations in the March $17N$ algorithm. A dot (.) entry means that the fault will not be detected by the corresponding Read operation. A 1 entry indicates an incorrect Read result where the expected value is 1, while a 0 entry indicates an incorrect Read result where the expected value is 0. For example, if there is an $AF <$, the first, third, and fifth Read operations will return the values 1, 0, and 1, but the expected values are 0, 1, and 0, respectively, so the March signature is $(0.1.0......)$. Note that in the table SAF(0) and $TF(U)$ have the same March signature. It is impossible to differentiate these two faults if we initialize the memory cell array by the all-0 background [1], unless timing effect is considered.

The process of mapping the defect signatures (behavior) to the fault signatures (as listed in Table 1) is called *defect mapping*. Based on the analysis by DANDI, we found that an open defect usually leads to an SAF, TF, or RDF, but it may also result in a signature not listed in the table. Such a signature cannot be mapped to a fault in the fault list, so the corresponding fault is called an *unmodeled fault*. The open defect mapping results are summarized in Table 2 (also see Fig. 4 for the MOS transistor labels), where the A and B entries represent two unmodeled faults with the March signatures $(0...0..00..)$ and $(..1..1....1)$, respectively. Note that A or B may be identified as a modeled fault if we apply a longer test algorithm. If that is not possible, then the set of fault models are not sufficient, so new fault models should be introduced. The two dashes in the table mean that the respective opens are undetectable by the March $17N$ algorithm (but they may be detected by another algorithm).

Table 2. Mapping table for the open defects.

	M0	M1	M2
Gate open	$SAF(1)$	$TF(U)$	$TF(U)$
Drain open	$SAF(1)$	$SAF(1)$	$TF(U)$
Source open	$SAF(1)$	$SAF(1)$	$TF(U)$
Body open	–	$RDF(0)$	–
	M3	**M4**	**M5**
Gate open	$SAF(1)$	$SAF(1)$	$SAF(1)$
Drain open	$TF(U)$	$SAF(1)$	$SAF(1)$
Source open	$TF(U)$	B	A
Body open	$RDF(1)$	$RDF(1)$	$RDF(0)$

The SRAM cell (see Fig. 4) has 7 independent lines, so a total of $C_2^7 = 21$ shorts inside the cell (called the *intra-cell shorts*) are considered. An intra-cell short may affect the behavior of other cells. Table 3 shows the defect mapping results of the intra-cell shorts, where C_k denotes the k-th cell in the 16-cell SRAM shown in Fig. 4. Note that the base cell (i.e., the cell currently being tested) is C_{10}, but all cells affected by a short in C_{10} are listed in the table. The 7 different lines are *bl* (bit line), $\overline{bl}$, *wl* (word line), x, y, Vdd, and GND. Each row in the table represents a short between two of the 7 lines in cell C_{10} (the table for other base cells can be derived

Table 3. Mapping table for the intra-cell short defects.

	C_1	C_2	C_3	C_4	C_6	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{14}
$\overline{bl}-bl$		SAF(1)			SAF(1)			SAF(1)			G
$\overline{bl}-wl$		SAF(1)		G	SAF(1)	SAF(1)	SAF(1)	SAF(1)	SAF(1)	E	G
$\overline{bl}-x$		TF(U)			TF(U)			SAF(1)			TF(U)
$\overline{bl}-y$		TF(U)			TF(U)			TF(U)			TF(U)
$\overline{bl}-$Vdd		TF(U)			TF(U)			TF(U)			TF(U)
$\overline{bl}-$GND		SAF(1)			SAF(1)			SAF(1)			G
$wl-bl$		TF(U)		D	TF(U)	H	SAF(1)	B	SAF(1)	F	TF(U)
$wl-x$						SAF(1)	SAF(1)	SAF(1)	SAF(1)		
$wl-y$						SAF(1)	SAF(1)	SAF(1)	SAF(1)		
$wl-$Vdd	TF(U)	TF(U)	TF(U)			SAF(1)	SAF(1)	SAF(1)	SAF(1)		
$wl-$GND						SAF(1)	SAF(1)	SAF(1)	SAF(1)		
$y-bl$		TF(D)			SAF(1)			SAF(1)			G
$y-x$								SAF(1)			
$y-$Vdd								TF(U)			
$y-$GND								SAF(1)			
GND$-bl$		TF(U)			TF(U)			TF(U)			TF(U)
GND$-x$								TF(U)			
Vdd$-x$								SAF(1)			
$bl-x$		TF(D)			TF(D)			SAF(1)			C
$bl-$Vdd		SAF(1)			SAF(1)			SAF(1)			G

in a similar way). The respective faults for the affected cells are shown as the table entries. Again, there are several unmodeled faults, labeled as $B, C, \ldots, H$, whose signatures are (..1..1....1), (...0..000..), (...00...0..), (0......00..), (0..0.......), (0..0..000..), and (01.001000..), respectively. Among them, only B (for the short between the word line and the bit line) affects the base cell. Table 3 lists 20 different shorts. The short between Vdd and GND is excluded, as its diagnosis is straightforward. The shorts among Vdd, GND, x, and y affect only the base cell, but not others. Any of the shorts wl-x, wl-y, wl-Vdd, and wl-GND results in an $SAF(1)$ on the entire row (i.e., cells 8, 9, 10, and 11). The behavior of $\overline{bl}$-Vdd is identical to that of bl-GND, while the $\overline{bl}$-GND and bl-Vdd shorts have the same behavior. Any of these four defects results in the column failure (i.e., faulty cells 2, 6, 10, and 14). Finally, the March signatures of bl-$\overline{bl}$, bl-Vdd, and GND-$\overline{bl}$ are the same. This actually is circuit-dependent, as two shorted lines can form a voltage divider, and the voltage level of the shorted lines is determined by the resistance ratio.

Figure 5 shows the fault patterns (patterns of the faulty cells) for the 20 intra-cell short defects. Using the March signatures and the fault patterns, diagnosis of the defects becomes simple and systematic.

The inter-cell short defects include the shorts between the bit lines of neighboring columns and word lines of neighboring rows. Table 4 shows the defect mapping results of four shorts between the second and third rows, and between the second and third columns. Analysis of shorts between other rows and columns is similar. From the table, we see unmodeled faults A and B again. A word line short results in SAFs in all the cells of the two affected rows. A short between two bit lines leads to the signature (..1..1....1) (for the unmodeled fault B). Other situations are also given in the table.

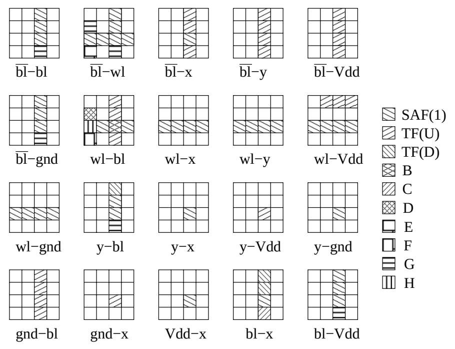

Figure 5. Fault patterns of the intra-cell short defects.

Table 4. Mapping table for the inter-cell short defects.

	C_1	C_2	C_5	C_6
bl_2-bl_3	B	B	B	B
$\overline{bl_2}-bl_3$	SAF(1)	B	A	B
$\overline{bl_2}-\overline{bl_3}$	TF(D)	SAF(1)	RDF(0)	A
	C_9	C_{10}	C_{13}	C_{14}
bl_2-bl_3	B	B	B	B
$\overline{bl_2}-bl_3$	A	B	A	B
$\overline{bl_2}-\overline{bl_3}$	RDF(0)	A	RDF(0)	A
	C_4	C_5	C_6	C_7
wl_2-wl_3	SAF(1)	SAF(1)	SAF(1)	SAF(1)
	C_8	C_9	C_{10}	C_{11}
wl_2-wl_3	SAF(1)	SAF(1)	SAF(1)	SAF(1)

The stuck-on and stuck-off defects have simpler behavior. They only affect the base cell, resulting in an SAF(1) or TF(U), as shown in Table 5.

Table 5. Mapping table for the stuck-on/off defects.

	M0	M1	M2
Stuck-on	SAF(1)	TF(U)	TF(U)
Stuck-off	TF(U)	SAF(1)	SAF(1)
	M3	M4	M5
Stuck-on	SAF(1)	SAF(1)	TF(U)
Stuck-off	TF(U)	SAF(1)	SAF(1)

We consider only the break defects of nodes x and y, each of which connecting the gates or drains of five MOS devices, as shown in Figure 6. Therefore, a total of $C_2^5 = 10$ break defects for each of the two nodes are simulated. Table 6 is the resulting defect mapping table, for x (upper rows) and y (lower rows). In the table, e.g., x_1x_2 denotes the partition of the five terminals of x into two groups: (x_1, x_2) and (x_3, x_4, x_5). Our simulation results show that the break defects of x and y do not affect other cells. Most of the equivalent faults are $SAF(1)$ and $TF(U)$.

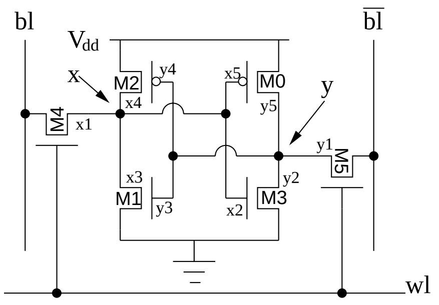

Figure 6. All terminals of the nodes x and y.

Table 6. Mapping table for the break defects.

x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3
SAF(1)	SAF(1)	TF(U)	B	RDF(0)
x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
SAF(1)	SAF(1)	SAF(1)	SAF(1)	SAF(1)
y_1y_2	y_1y_3	y_1y_4	y_1y_5	y_2y_3
TF(D)	TF(U)	A	SAF(1)	A
y_2y_4	y_2y_5	y_3y_4	y_3y_5	y_4y_5
TF(U)	TF(U)	TF(U)	SAF(1)	SAF(1)

From our analysis, eight unmodeled faults are discovered for the specific RAM under the March 17N test algorithm. The unmodeled faults A and B are single-cell faults, so they affect only the defective cell. Each of other unmodeled faults $(C, D, \ldots, H)$ affects the victim cells when there exists an intra-cell short defect. We list the original functional faults with similar signatures as the unmodeled faults in Table 7. Different test algorithms and memory circuits may have different results. However, our methodology can easily be adapted for use in such cases.

Table 7. Comparison of modeled and unmodeled faults.

Fault Type	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
A	0	.	.	.	0	.	.	0	0	.	.
RDF(0)	.	.	.	.	0	.	.	0	0	.	.
B	.	.	1	.	.	1	.	.	.	.	1
RDF(1)	.	.	1	.	.	.	.	.	.	.	1
C	.	.	.	0	.	.	0	0	0	.	.
CFst(0;1)<	.	.	.	0	.	.	.	0	0	.	.
D	.	.	.	0	0	.	.	.	0	.	.
CFst(1;1)>	.	.	.	0	.	.	.	.	0	.	.
E	0	.	.	.	.	.	.	0	0	.	.
CFid(D;1)<	.	.	.	.	.	.	.	0	0	.	.
F	0	.	.	0	.	.	.	.	.	.	.
CFid(U;1)<	0	.	.	.	0	.	.	.	.	.	.
G	0	.	.	0	.	.	0	0	0	.	.
SAF(1)	0	.	.	0	0	.	0	0	0	.	.
H	0	1	.	0	0	1	0	0	0	.	.
CFst(0;0)>	.	1	.	.	.	1	.	.	.	.	.

The faulty signatures have high correlation with the cell design. For example, the $\overline{bl_2}$-bl_3 short leads to an extra leakage path (see Fig. 7), which results in the tug of war between the two memory cells. The defective memory output may be affected by the device ratios of the pMOS and nMOS transistors in the cells (i.e., M0, M1, M2, and M3). Figure 8 shows one possible result using several pMOS/nMOS channel width ratios, by Spice simulation of the defective circuit. According to the figure, when we increase the channel width for all M1 and M3 nMOS transistors, we observe an increased discharge time as well as a lower final $\overline{bl_2}$ voltage. In this case, the defect can be covered by the delay fault model.

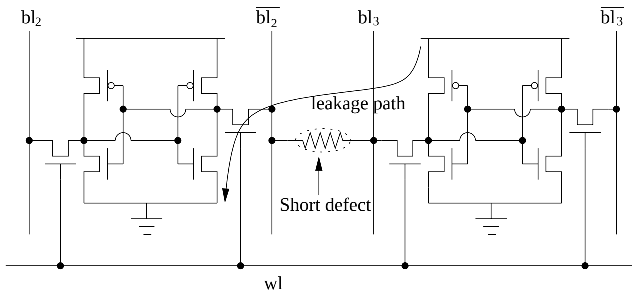

Figure 7. The inter-cell $\overline{bl_2}$-bl_3 short.

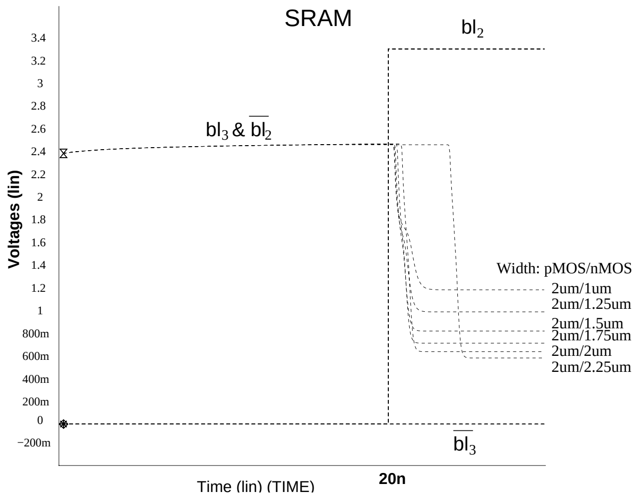

Figure 8. Spice simulation result using several pMOS/nMOS channel width ratios.

Using DANDI on the March 17*N* algorithm for the specific memory design and defects discussed above, we found only 2.47% of the defects that have escaped the test. Among the detected defects, about 21% result in unmodeled faults, and more than 73% result in SAFs and TFs.

5. Conclusions

We have presented a methodology for systematically injecting defects into an SRAM and simulating the effects. A typical SRAM circuit and its common defects have been used for demonstrating the methodology. The detect mapping tables were constructed after the simulation. With such tables, memory diagnosis for design debugging and yield improvement is shown to be done efficiently. The analysis of the circuit defects helps the memory designers and process engineers locate the weakness of the memory chips. The defect and fault detectability information obtained from the analysis also is useful for optimizing the test algorithm and the redundancy structure of the memory.

References

[1] T. J. Bergfeld, D. Niggemeyer, and E. M. Rudnick, “Diagnostic testing of embedded memories using BIST”, in *Proc. Design, Automation and Test in Europe (DATE)*, Paris, Mar. 2000, pp. 305–309.

[2] C.-W. Wang, C.-F. Wu, J.-F. Li, C.-W. Wu, T. Teng, K. Chiu, and H.-P. Lin, “A built-in self-test and self-diagnosis scheme for embedded SRAM”, in *Proc. Ninth IEEE Asian Test Symp. (ATS)*, Taipei, Dec. 2000, pp. 45–50.

[3] J.-F Li, K.-L. Cheng, C.-T. Huang, and C.-W. Wu, “March-based RAM diagnosis algorithms for stuck-at and coupling faults”, in *Proc. Int. Test Conf. (ITC)*, Baltmore, Oct. 2001, pp. 758–767.

[4] J. T. Chen, J. Rajski, J. Khare, O. Kebichi, and W. Maly, “Enabling embedded memory diagnosis via test response compression”, in *Proc. IEEE VLSI Test Symp. (VTS)*, Marina Del Rey, California, Apr. 2001, pp. 292–298.

[5] D. Niggemeyer and E. Rudnick, “Automatic generation of diagnostic March tests”, in *Proc. IEEE VLSI Test Symp. (VTS)*, Marina Del Rey, California, Apr. 2001, pp. 299–304.

[6] C.-F. Wu, C.-T. Huang, K.-L. Cheng, and C.-W. Wu, “Simulation-based test algorithm generation for random access memories”, in *Proc. IEEE VLSI Test Symp. (VTS)*, Montreal, Apr. 2000, pp. 291–296.

[7] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, Kluwer Academic Publishers, Norwell, Massachusetts, 2000.

[8] S. Hamdioui and A. J. van de Goor, “An experimental analysis of spot defects in SRAMs: realistic fault models and tests”, in *Proc. Ninth IEEE Asian Test Symp. (ATS)*, Taipei, Dec. 2000, pp. 131–138.

[9] K. Zarrineh, A. P. Deo, and R. D. Adams, “Defect analysis and realistic fault model extensions for static random access memories”, in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, San Jose, Aug. 2000, pp. 119–124.

[10] A. Jee, J. E. Colburn, V. S. Irrinki, and M. Puri, “Optimizing memory tests by analyzing defect coverage”, in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, San Jose, Aug. 2000, pp. 20–25.

[11] A. Jee, “Defect-oriented analysis of memory BIST tests”, in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, Isle of Bendor, France, July 2002, pp. 7–11.

[12] T. M. Mak, D. Bhattacharya, C. Prunty, B. Roeder, N. Ramadan, J. Ferguson, and Y. Jianlin, “Cache RAM inductive fault analysis with fab defect modeling”, in *Proc. Int. Test Conf. (ITC)*, Oct. 1998, pp. 862–871.

[13] C.-F. Wu, C.-T. Huang, C.-W. Wang, K.-L. Cheng, and C.-W. Wu, “Error catch and analysis for semiconductor memories using March tests”, in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, San Jose, Nov. 2000, pp. 468–471.

[14] C.-F. Wu, C.-T. Huang, K.-L. Cheng, and C.-W. Wu, “Fault simulation and test algorithm generation for random access memories”, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 4, pp. 480–490, Apr. 2002.

[15] P. Nagaraj, S. Upadhyaya, K. Zarrineh, and D. Adams, “Defect analysis and a new fault model for multi-port SRAMs”, in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT)*, San Francisco, Oct. 2001, pp. 366–374.