

Fault Pattern Oriented Defect Diagnosis for Memories

Chih-Wea Wang, Kuo-Liang Cheng, Jih-Nung Lee, Yung-Fa Chou,
Chih-Tsun Huang, and Cheng-Wen Wu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan

Frank Huang and Hong-Tzer Yang
Spirox Co.
Hsinchu, Taiwan

Abstract

Failure analysis (FA) and diagnosis of memory cores plays a key role in system-on-chip (SOC) product development and yield ramp-up. Conventional FA based on bitmaps and the experiences of the FA engineer is time consuming and error prone. The increasing time-to-volume pressure on semiconductor products calls for new development flow that enables the product to reach a profitable yield level as soon as possible. Demand in methodologies that allow FA automation thus increases rapidly in recent years. This paper proposes a systematic diagnosis approach based on failure patterns and functional fault models of semiconductor memories. By circuit-level simulation and analysis, we have also developed a fault pattern generator. Defect diagnosis and FA can be performed automatically by using the fault patterns, reducing the time in yield improvement. The main contribution of the paper is thus a methodology and procedure for accelerating FA and yield optimization for semiconductor memories.

Keywords: bitmap, failure analysis (FA), fault pattern, memory testing, memory diagnostics, semiconductor memory.

1. Introduction

Embedded memory test design has become an essential part of the system-on-chip (SOC) development infrastructure [1]. According to the recent ITRS report, the memory cores will occupy more than 90% of the chip area in less than ten years [2]. The yield of on-chip memories thus will dominate the chip yield. Go/no-go testing is no longer enough for embedded memories in the SOC era. Memory diagnostics is quickly becoming a critical issue, so far as

manufacturing yield and time-to-volume of SOC products are concerned. Effective memory diagnostics and failure analysis (FA) methodologies will help improve the yield of SOC products, especially with rapid revolution in new product development and advanced process technologies [1].

The yield loss concerned here is mainly due to wafer process defects. Failure analysis can be used to identify the defects causing the yield loss. Based on the analysis results the process can be tuned and/or the design can be modified to enhance the yield. Conventionally, an FA engineer first detects and locates the faulty cell or region, and then performs a series of physical de-processing, e-beam probing, or electron microscope inspection [3, 4]. However, without proper methodologies and tools it is more and more difficult to perform the defect-level testing and diagnostics, as we get into the deep-submicron age. Bitmaps and wafer maps are tools that have been commonly used in FA, because the failure patterns and distributions are helpful for the FA engineers to narrow down the potential cause of the failure, based on their experience [3, 5, 6]. Many commercial memory testers and associated yield analysis tools can be used for this purpose. The test engineers also can define a particular set of failure patterns based on the past analysis results. Automatic analysis tools will identify and locate these particular patterns in the failed memory or wafer, for subsequent identification of actual defects [7]. Inductive fault analysis also has been used to link the defects to fault models for semiconductor memories [8], but the work needs continuous improvement, as new memory designs and technologies keep coming out in a fast pace. For memories, fault models are usually defined at the functional level. Based on the faulty circuit behavior, a test algorithm can be developed to detect the faults that are modeled [9, 10]. The quality of the test algorithm usually is determined by its length and fault coverage. Memory fault diagnostics procedure also has been developed based on functional fault models [11]. Alterna-

tively, faults can be transformed into signatures, with respect to some test algorithm, and be used in the diagnostic procedure [12–14]. Both the failure pattern and fault type approaches have drawbacks. There are defects causing the memory to fail with different behavior but same failure pattern. Relying only on that information limits the accuracy of diagnostic results. Also, the current fault models may be good for production test, but they are normally insufficient for diagnostic test that requires much more detailed fault models [15]. So far the fault modeling still relies on the manual analysis.

In this paper, we propose a *fault pattern* based approach for memory diagnosis. Both bitmaps and functional fault models are used to enhance the results. With fault patterns, a failed memory can be classified and identified both by the failure patterns and the faults associated with these patterns. Similar to the previous memory fault diagnostics approach [12], a *defect dictionary* is constructed to map the actual failure response of a memory to possible defects through the fault patterns. This fault pattern based approach can easily be automated. We have developed a systematic diagnostics procedure that targets realistic memory defects, given the actual layout, particle size distribution, and process parameters. Experimental results using industrial SRAM chips justify the effectiveness of the fault patterns.

2. Memory Diagnostics

When a memory is tested, a bitmap can be constructed automatically by the memory tester. The bitmap is a topological representation of the test result, showing physical locations of the failed bits detected. Failure patterns (such as failed word lines, failed bit lines, random bit failures, etc.) on the bitmap can further be identified. By FA and/or process simulation, the possible defects that caused the failures can then be mapped to the failure patterns. In [3, 5, 6], some of the frequently used failure patterns were discussed. Figure 1 shows four failure pattern examples. The failure patterns of primary concern include a single failed cell, a cluster, a column and/or row, and multiple columns and/or rows. Failure pattern identification can be integrated into the memory BIST module, i.e., by the internal BIST scheme [6]. It also can be done by an external software that analyzes the compressed failure patterns [5]. However, mapping failure patterns to defects is not trivial.



Figure 1. Four failure pattern examples.

There are research works on mapping defects to functional faults. Fault model-based testing and diagnostics that can be done automatically has been presented in [12, 15].

March-based test and diagnostic algorithms are easy to implement. Figure 2 shows a March 17N diagnostic algorithm [12] for stuck-at fault (SAF), transition fault (TF), address decoder fault (AF), stuck-open fault (SOF), and coupling fault (CF). This March algorithm consists of several March elements, separated by semicolons. The \uparrow stands for the ascending order of the address sequence, and \downarrow stands for the descending order. Inside the parentheses is the specification of a series of Read/Write operations and the corresponding data background. These Read/Write operations are to be applied to each address, one by one, following the address order in front of the parentheses. All operations inside the parentheses have to be performed before we proceed to the next address. We use the March signature [12] to represent the results from all operations in the test algorithm, which are either correct (represented by a 0) or incorrect (represented by a 1). We assume here that only the read operations can detect the failure. For some special memories, however, the write-through and write-verify operations can also detect the errors. For example, the signatures of SAF0 (stuck-at-0 fault), SAF1 (stuck-at-1 fault), and RDF0 (read disturb fault) are stored in a dictionary as shown in Table 1. The fault diagnostics procedure is to compare the actual responses with the March signatures stored in the dictionary, such that each faulty cell can be identified as having one of the prespecified faults in the dictionary [12]. The difficulty of this approach is that the fault models commonly used may be good for production test, but they are normally insufficient for diagnostic test, as the latter requires much more detailed fault models, e.g., there are occasions when the signature of the failed cell does not match any signature in the dictionary.

$$\begin{aligned} &\uparrow (w0); \uparrow (r0, w1, r1); \uparrow (r1, w0, r0); \uparrow (r0, w1); \\ &\downarrow (r1, w0, r0); \uparrow (r0); \downarrow (r0, w1, r1); \uparrow (r1); \end{aligned}$$

Figure 2. A March 17N diagnostic algorithm.

Table 1. Some March signatures for the March 17N algorithm.

Fault	March Signature
SAF ₀	00011000010000011
SAF ₁	01000011000111000
RDF ₀	00000001000011000

We combine the failure patterns and fault types, and propose the notion of *fault pattern*. Figure 3 gives some fault pattern examples. A fault pattern is similar to a failure pattern, except that each faulty cell in the pattern is associated by a fault that can be identified in the fault dictionary. A proper FA procedure can be used to construct the fault patterns, which will be discussed next.

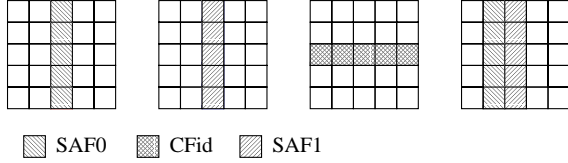


Figure 3. Fault pattern examples on a 5×5 cell array.

3. Fault Pattern

A cause-effect approach can be used to predict the faulty behavior of a memory when a defect occurs. In the conventional FA method, a large number of defects of various types and sizes are placed one at a time on the memory layout (usually SRAM), and the resulting circuit for each defect is extracted. Circuit simulation is then used to determine the faulty behavior that is obtained as a result of each injected defect. Finally, the resulting faulty behaviors are transformed into realistic functional fault models.

Our purpose is different. We will use the FA-based methodology to explore the faulty behavior both for a cell and for the whole array. We do not stress fault modeling here. For defect-level diagnosis, we will obtain the mapping between a defect and its faulty behavior. In the cause-effect approach, every environment parameter must be considered in the prediction phase. These parameters include the test algorithm, operating speed, voltages applied during the test/diagnosis process, etc.

3.1. Defect Model

Before discussing defect-level diagnostics, we should define the defect models. In many previous works, the defects are modeled as a disk of undesired particles or extra/missing material. According to their positions and conductivities, these defects may cause extra connection (short) or disconnection (open) on a certain layer or between different layers of wires. To perform simulation and analysis, we model these defects at the circuit level. For example, we add an extra resistor between two nodes to model the extra material or particle on the layout, as shown in Fig. 4(a). Similarly, the missing material or particles on a conducting layer will break the connection, so we add an extra node and a large resistor into the netlist to model such type of defect, as shown in Fig. 4(b). As another example, an undesired particle on the via can lead to an open connection, with a similar effect in the netlist as shown in Fig. 4(c). We assume all major defects can be modeled at the circuit level with extra resistors and nodes. A circuit-level simulator is used to determine the faulty behavior of any defect. For high-frequency behavior, however, one may need to use a more sophisticated model that includes capacitors and/or inductors.

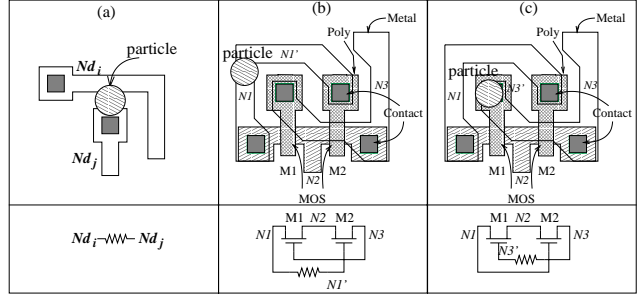


Figure 4. (a) A short defect; (b) an open defect; (c) a missing contact defect.

The original defects may have different size, shape, conductivity, distribution, etc., so the faulty behavior may not be the same for different memory designs and processes. Exhaustive defect injection and analysis has been performed for fault modeling [16]. Exhaustive injection and analysis is very time consuming. It also includes defects (and thus fault models) that are less likely to exist in real circuits. For realistic defects the layout, process information, and past defect data (in addition to circuit design) should be considered [17]. Exhaustive defect injection has another problem in defect-level diagnostics. A single failure behavior may be mapped to multiple suspect defects, including realistic defects and redundant ones. The result is a decrease in diagnostic resolution, and thus an increase in the time and cost to identify the real defect in the subsequent debugging process. Defect models thus should be confined to those that are necessary.

3.2. Automatic Defect Injection

An embedded memory design can be used in different SOC products, using different process technologies. Even if the chip is manufactured by the same foundry, the process still may be modified from time to time. The size, distribution, and conductivity of a particle defect may also change. For accurate defect diagnostics, the set of target defects and the corresponding fault patterns need to be updated. Apparently the update process must be automatic. Here we propose an automatic procedure to generate the fault dictionary, which helps map the realistic defects to failures. The realistic defects are obtained from the memory layout, particle distribution, size and conductivity of particle defects. Figure 5 shows the proposed memory defect diagnostics (MDD) system, which consists of an automatic FA (AFA) subsystem and a fault/failure pattern analysis module. The AFA subsystem can automatically inject defects, perform simulations, and generate the defect dictionary with fault/failure patterns. With the fault bitmap generated from the MECA system we proposed before [12]

and the fault dictionary, the suspect defects can be identified by fault/failure pattern analysis. Additional realistic failure or fault patterns specified by the user can also be included for analysis. The AFA tool reads the memory layout in the GDSII format. It also needs the size and conductivity of particle defects to determine the failure modes, if any. If there is a failure, the defect will be translated to a corresponding resistor for circuit-level simulation.

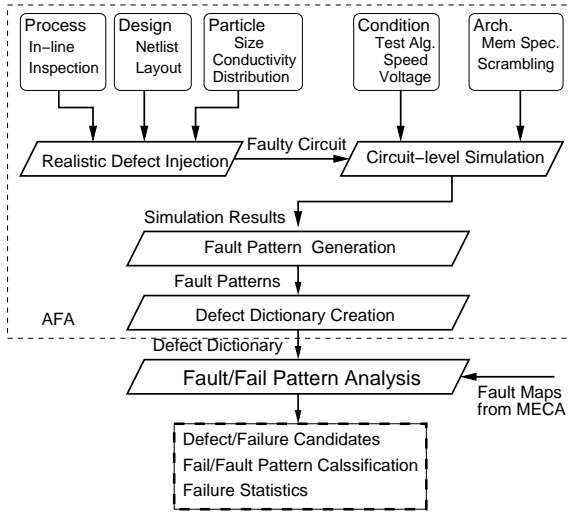


Figure 5. The memory defect diagnostics (MDD) system.

The defect distribution is probabilistic, e.g., the resistances of the open defects in metal layers have a distribution as shown in Fig. 6, for a 180-nm CMOS process [18]. We can inject the open defects with resistances $100k\Omega$, $200k\Omega$, ..., $1G\Omega$ (based on the defect distribution) into the layout and simulate them to get the failure behavior. We calculate a weight for each resulting failure behavior according to the defect distribution. When we perform diagnostics, we may have multiple suspect defects. The engineer can do further analysis based on their weights.

The defects are classified into three groups: (intra-layer) shorts, opens, and missing contacts. We will discuss them in detail below. The defect injection also is classified into two types: the intra-cell and inter-cell injections. For failure pattern exploration we need to consider the entire cell array, i.e., we need to cover the intra-cell and inter-cell cases. The regularity of memories enables us to consider only a small block of the memory without losing important information, so the complexity is greatly reduced. Of course the fault patterns still need to cover the whole array.

1. Short: Although a short may occur between adjacent layers due to, e.g., bad oxide, shorts at the same layer are more likely and are considered currently. Oxide defects are a serious problem in the manufacturing pro-

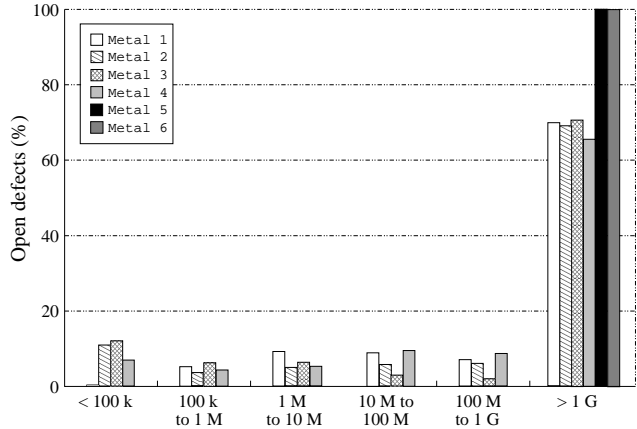


Figure 6. Distribution of metal open resistance [18].

cess and can be identified by other approaches. The defect injection procedure will determine possible shorts from the given defect size and memory layout. For each short in the set, an extra (small) resistor will be inserted into the netlist, with a prespecified resistance. The faulty netlist will be simulated in the next stage. We only consider single defects in the injection procedure. Assume the total number of metal and poly layers is n , which are denoted as $\{LM_0, LM_1, \dots, LM_{n-1}\}$; and the number of contact and via layers is m , which are denoted as $\{LC_0, LC_1, \dots, LC_{m-1}\}$. The main task in the injection procedure is to calculate the probability of a short between two polygons, given a defect size. We use each node of a polygon as the center to draw a circle with a radius the diameter of the particle defect, and determine if it crosses other polygons. An example is shown in Fig. 7(a).

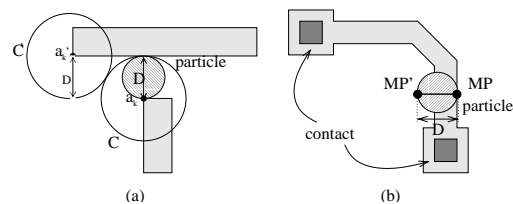


Figure 7. Defect search examples: (a) short defects; and (b) open defects.

2. Open: We only consider open defects at the metal and poly layers. It is defined as a particle defect that breaks a connection. We need to determine if the defect size is larger than the metal/poly width. Figure 7(b) shows an example, where the polygon with a particle defect has two contacts. We define all contacts and vias as the *fanout* of the polygons at a certain layer. According to fanouts, we split a polygon into smaller ones and de-

termine the width of each segment. If the open defect occurs, we insert an extra (large) resistor and an extra node to separate the signals.

3. Missing contact: The missing contact is a special type of open defect. The injection procedure is similar to that for the open defect. The contacts and vias are identified first, and then the procedure will separate the nodes (layers) by inserting a (large) resistor.

```

Short:
for each layer  $l_i \in \{LM_0, LM_1, \dots, LM_{n-1}\}$ 
  for each polygon  $P_j(a_0, a_1, \dots, a_s) \in l_i, j = 0$  to  $q_i - 1$ 
    for each point  $a_k, k = 0$  to  $s$ 
      Draw a circle  $C$ , with center  $a_k$  and radius  $D$ 
      for each polygon  $P_h \in l_i, h = 0$  to  $q_i - 1, h \neq j$ 
        Partition  $P_h$  into a set of segments  $\{L_0, L_1, \dots, L_x\}$ 
        defined by the points
        for each segment  $L_f, f = 0$  to  $x$ 
          Determine the intersection point between  $L_f$  and  $C$ 
          If the intersection point exists
             $Nd_j = \text{trace\_nodename}(P_j)$ 
             $Nd_h = \text{trace\_nodename}(P_h)$ 
             $\text{defective\_circuit} = \text{insert\_short\_resistor}(Nd_j, Nd_h, R)$ 
            Store the  $\text{defective\_circuit}$  in  $DC$ 

Open:
for each layer  $l_i \in \{LM_0, LM_1, \dots, LM_{n-1}\}$ 
  for each polygon  $P_j \in l_i, j = 0$  to  $q_i - 1$ 
    Split  $P_j$  into smaller polygons  $\{p_0, p_1, \dots, p_b\}$  by fanouts
    for each  $p_k(a_0, a_1, \dots, a_e) \in \{p_0, p_1, \dots, p_b\}$ 
      for each point pair  $(a_h, a_{h+1}), h = 0$  to  $e - 1$ 
        Determine the midpoint  $MP$  and slope of the line
        segment  $\langle a_h, a_{h+1} \rangle$ 
        Draw a perpendicular line segment of length  $D$ 
        from  $MP$  to  $MP'$ 
        /* This line segment must cross  $p_k$ . */
        If  $MP'$  is not inside  $p_k$ 
           $\{fa_1, fa_2\} = \text{trace\_fanout}(p_k)$ 
           $\text{node\_set\_1} = \text{trace\_all\_node\_from\_fanout}(fa_1)$ 
           $\text{node\_set\_2} = \text{trace\_all\_node\_from\_fanout}(fa_2)$ 
           $Nd_1 = \text{trace\_nodename}(\text{node\_set\_1})$ 
           $Nd_2 = \text{trace\_nodename}(\text{node\_set\_2})$ 
           $\text{exchange\_nodename\_to\_NP}(\text{node\_set\_1})$ 
           $\text{defective\_circuit} = \text{insert\_short\_resistor}(Nd_2, NP, R)$ 
          Store the  $\text{defective\_circuit}$  in  $DC$ 

Missing Contact/Via:
for each layer  $l_i \in \{LC_0, LC_1, \dots, LC_{n-1}\}$ 
   $\text{node\_set\_up\_layer} = \text{trace\_all\_node\_from\_via}(l_i)$ 
   $\text{node\_set\_dn\_layer} = \text{trace\_all\_node\_from\_via}(l_i)$ 
   $Nd_1 = \text{trace\_nodename}(\text{node\_set\_up\_layer})$ 
   $Nd_2 = \text{trace\_nodename}(\text{node\_set\_dn\_layer})$ 
   $\text{exchange\_nodename\_to\_NP}(\text{node\_set\_up\_layer})$ 
   $\text{defective\_circuit} = \text{insert\_short\_resistor}(Nd_2, NP, R)$ 
  Store the  $\text{defective\_circuit}$  in  $DC$ 

```

Figure 8. The defect injection procedure.

Figure 8 shows the injection procedure for the three defect types. Assume 1) each layer l_i contains q_i polygons; 2) the diameter of the particle defect is D ; 3) the resistance

of the particle defect is R ; and 4) DC stores the defective circuits (initially, $DC = \{\}$) to be simulated.

An experiment has been conducted using a commercial $0.25\mu\text{m}$ embedded SRAM design. The cell layout is shown in Fig. 9, where BL, BLb, D, and Db denote the bit-line, bit-line, cell-data, and cell-data, respectively. A software tool has been developed to generate the realistic defects using the procedure shown above. The tool reads the memory layout (in GDSII) and the circuit netlist as inputs, and produces the set DC as outputs. The open defects and missing contacts for this experimental case are summarized in Table 2. The numbers 250, 300, and 350 in the table denote the diameters of the point defects. The last two columns indicate whether the defect at the contact or via will lead to an open node. For example, the contact defects will break the nets of D, Db, GND, VDD, BL, and BLb, but the WL will not be affected. Table 3 lists the realistic short defects for this case, with the particle defect size ranging from 250nm to 400nm. As shown in the table, there are totally 11 possible short defects for this cell. Note that the result depends on the defect distribution, process parameters, cell layout, etc., so some of the node pair combinations are not listed as they are not likely to occur. One can also assign a weight to each defect, denoting its priority. The weight can be used in the subsequent simulation and creation of the fault pattern dictionary.

Table 2. Realistic open defects.

Node	Open defect (nm)			Contact	Via
	250	300	350		
D		V	V	V	
Db		V	V	V	
GND			V	V	
VDD				V	V
BL				V	V
BLb				V	V
WL	V	V	V		

3.3. Simulation of Faulty Circuits

After defect injection and generation of the faulty circuits, circuit simulations are performed to predict the faulty behavior caused by each and every defect. To speed up the simulation, we use just a small memory block instead of the entire memory. It can be a bank or slot of the whole memory. The concern here is that the block must have the same condition and environment as the whole memory, including loading, address scrambling, etc. The simulation uses the same conditions as in production test and diagnostic test. We perform the simulation using the specified test algorithm, AC parameters, and DC parameters. The parameters are important. For example, a resistive short may cause a

Table 3. Realistic short defects for the experimental case.

Shorting Nodes		Poly				Metal 1				Metal 2			
		250	300	350	400	250	300	350	400	250	300	350	400
WL	Db	V	V	V	V								
WL	D	V	V	V	V								
BLb	GND							V	V				
BL	GND							V	V				
VDD	Db					V	V	V	V				
VDD	D							V	V				
GND	Db							V	V				
GND	D							V	V				
D	Db				V	V	V	V	V				
VDD	BL												V
VDD	BLb												V

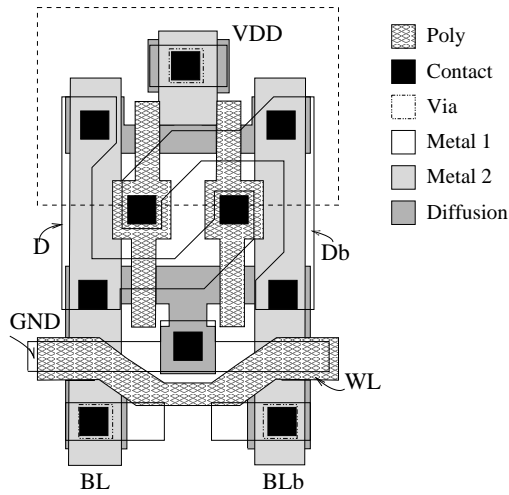


Figure 9. Cell layout of the experimental case.

degradation in the rise/fall time. A lower operating speed may not be able to detect the problem.

3.4. Fault Pattern Generation

The March signature of a test algorithm with respect to a specific fault is defined as the (Read) results after applying the test algorithm to the memory when the given fault is present [12, 15]. March signatures can be used to identify fault types after testing, by comparing the signatures with those stored in the March dictionary [12, 15]. However, experiments on real memory chips have shown that there are signatures that cannot be identified with known fault models from the dictionary that was constructed by fault simulation. One way to solve the problem is to develop more fault models and/or longer tests for diagnosis purposes, but that is usually not cost-effective, and the sophisticated faults are not necessarily realistic, therefore may not be useful for

FA or design engineers. We have noted that these unrecognized fault types (signatures) do not affect the efficiency of defect-level diagnostics in our system.

Our goal here is to isolate the defects based on the test response. We use the March signature to represent the test result for every cell in the memory. We obtain not only the failure pattern, but the fault type of each cell as well. The fault patterns are then generated for the target memory. Figure 10 shows 16 fault pattern examples. Other fault patterns not shown in this figure include two adjacent cells (the base cell and neighbor cell) with same/different fault types, etc. Each fault pattern is caused by a certain defect with a particular resistance value. The 16 fault patterns shown in the figure are all based on a 10Ω short resistance and $1M\Omega$ open resistance. Random faults occur when the voltage difference on a sense amplifier is too small. Details will be discussed below.

A defect has a corresponding failure behavior after the simulation stage, and its behavior can be represented in form of March signature. We translate these signatures to known fault types as in our previous approach [12]. However, this may not succeed, as mentioned above. When it fails, we assign a temporary fault name to the fault pattern associated with the signature. These fault patterns can still be mapped to the potential defects.

We now define a special fault type—the random fault [19]. A random fault occurs when the voltage difference on a sense amplifier is too small, such that the output of the sense amplifier is affected by the noise or the mismatch in the sense amplifier circuit itself. Under this condition, apparently the output of the memory cannot be predicted by simulation. To detect the random fault, we can monitor the voltage of the bit-line pair when the sense amplifier is being operated. For example, a small resistive short between the bit-line pair will reduce the voltage difference and the sense amplifier will fail. When we detect this condition, the fault pattern generator will record this fault as a random fault.

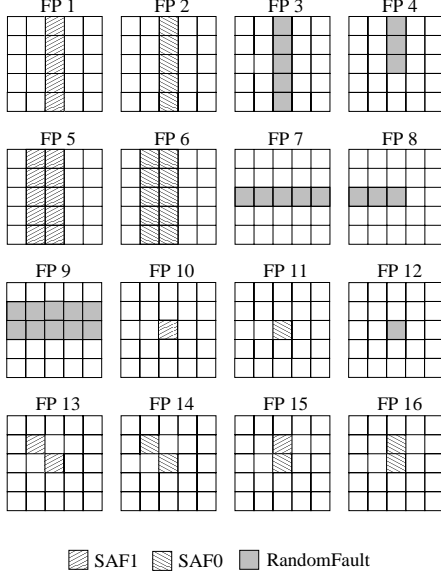


Figure 10. Fault pattern examples.

The response of this cell may be correct or incorrect during testing/diagnostics. We will not try to identify the exact fault type for the cell with random fault in the diagnostic stage. We will identify the failure pattern instead.

3.5. Defect Dictionary Creation

In the last stage of the MDD system we create the *defect dictionary* that maps the fault patterns to possible defects. In this stage, we collect the fault patterns and the corresponding defects into a dictionary. An example is shown in Table 4. It is possible that a certain fault pattern is mapped to multiple possible defects. The last column of the table denotes the number of possible defects. As shown in the table, each possible defect has a weight (w_{1a} to w_{22a}). When we perform the diagnosis and obtain a set of possible defects, the weights are used to determine the priority in the analysis that follows. The weight for a defect can be calculated from the defect distribution. In this work, we use the product of particle size, frequency, and resistance as the weight. Also, the results from in-line inspection can contribute to the weight if process information is available.

4. Defect Diagnostics Using Fault Patterns

Figure 11 shows the sets of possible defects obtained from a failure pattern, a fault type, and a fault pattern, respectively. The sets are all from the same failed chip caused by the same defect. Normally, the real defect should be contained in the intersection of the sets. As shown in the figure, both the failure pattern and fault type result in larger suspect defect sets, while the fault pattern usually leads to a much

Table 4. A defect dictionary example.

Fault pattern	Suspect defects	#
FP1	GND shorts to BLb with R_{1a} (w_{1a}) VDD shorts to BL with R_{1b} (w_{1b})	2
FP2	GND shorts to BL with R_{2a} (w_{2a}) VDD shorts to BLb with R_{2b} (w_{2b})	2
FP3	BL shorts BLb with R_{3a} (w_{3a})	1
FP4	BL opens with R_{4a} (w_{4a}) BLb opens with R_{4b} (w_{4b})	2
FP5	BLb _i shorts to BL _{i+1} with R_{5a} (w_{5a})	1
FP6	BL _i shorts to BLb _{i+1} with R_{6a} (w_{6a})	1
FP7	WL shorts to VDD or GND with R_{7a} (w_{7a})	2
FP8	WL open with R_{8a} (w_{8a})	1
FP9	WL _j shorts to WL _{j+1} with R_{9a} (w_{9a})	1
FP10	WP shorts to D with R_{10a} (w_{10a}) VDD shorts to D with R_{10b} (w_{10b}) GND shorts to Db with R_{10c} (w_{10c}) BLb shorts to D with R_{10d} (w_{10d}) WP _{j+1} shorts to Db with R_{10e} (w_{10e}) BL _{i+1} shorts to D with R_{10f} (w_{10f})	6
FP11	WP shorts to Db with R_{11a} (w_{11a}) VDD shorts to Db with R_{11b} (w_{11b}) GND shorts to D with R_{11c} (w_{11c}) BL shorts to Db with R_{11d} (w_{11d}) WP _{j+1} shorts to D with R_{11e} (w_{11e}) BLb _{i+1} shorts to Db with R_{11f} (w_{11f})	6
FP12	D shorts to Db with R_{12a} (w_{12a}) D opens with R_{12b} (w_{12b}) Db opens with R_{12c} (w_{12c})	3
FP13	Db _{i,j} shorts to Db _{i+1,j+1} with R_{13a} (w_{13a})	1
FP14	D _{i,j} shorts to D _{i+1,j+1} with R_{14a} (w_{14a})	1
FP15	D _{i,j} shorts to D _{i,j+1} with R_{15a} (w_{15a})	1
FP16	D _{i,j} shorts to Db _{i,j+1} with R_{16a} (w_{16a})	1
FP17	Db _{i,j} shorts to D _{i,j+1} with R_{17a} (w_{17a})	1
FP18	Db _{i,j} shorts to Db _{i,j+1} with R_{18a} (w_{18a})	1
FP19	D _{i,j} shorts to Db _{i+1,j} with R_{19a} (w_{19a})	1
FP20	D _{i,j} shorts to D _{i-1,j} with R_{20a} (w_{20a})	1
FP21	Db _{i,j} shorts to D _{i+1,j} with R_{21a} (w_{21a})	1
FP22	Db _{i,j} shorts to D _{i-1,j} with R_{22a} (w_{22a})	1

smaller suspect defect set. The larger suspect defect set obtained from failure pattern (or fault type) also suggests that different defects may result in the same failure pattern (or fault type), but most likely different fault pattern. Note that the sets obtained from the failure pattern and the fault type (and therefore, the fault pattern) can be quite different. For example, both the short defect between BL and VDD and that between BLb and VDD result in a failed column (the failure pattern), but the fault types of the cells in these two failed columns are different. Figure 12(a) shows the same failure pattern resulting from these two defects. Their different *fault patterns* are shown in Fig. 12(b). Of course this is just one of many possible situations. It is also possible

that different defects result in the same fault pattern, e.g., the short between BL and VDD and that between BLb and GND may result in the same fault type for the cells in the respective failed column. Nevertheless, the fault pattern can provide more accurate information than the failure pattern or fault type alone.

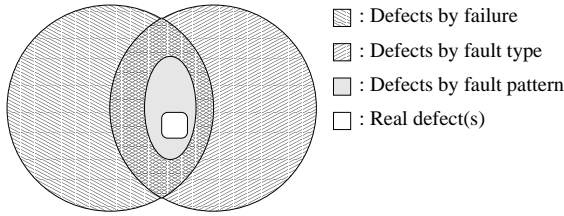


Figure 11. Defect isolation by failure pattern, fault type, and fault pattern.

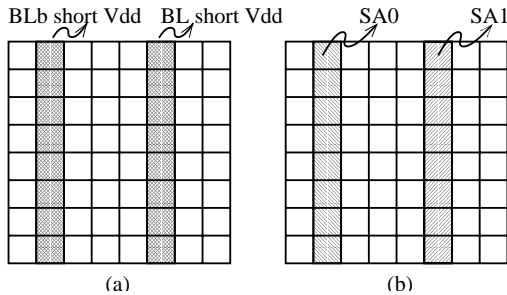


Figure 12. (a) Failure patterns and (b) fault patterns for two different defects.

From the FA results, we can construct the mapping table between possible defects and fault patterns (called the *defect dictionary*). Using the same diagnostic test algorithm for analysis (simulation) as well as for testing the memory chips to collect the test results, we can build the failure bitmap and fault bitmap for each failed memory chip using the MECA system [12]. With the defect dictionary and fault bitmap, we can identify the possible defects more efficiently and accurately than using failure pattern.

5. Experimental Results

We have done an experiment on an industrial case, using single-port SRAM chips. The memory is of size 64K×12, composed of four banks. It has 1,024 rows and 768 columns, and each bank has 512 rows and 384 columns. They were tested by the March 17N diagnostic test algorithm (discussed in Sec. 2) using the Credence Kalos-XP tester. With proper modification of the test program, the memory tester was able to generate the failure bitmap for

each operation. By parsing the output from the tester and analyzing it using the MECA system, the fault bitmaps were constructed. The memory scrambling data were used to construct the physical bitmaps for the failed chips. We show the bitmaps for some failed memory chips. The Type 1 bitmap is exemplified in Fig. 13(a), where there is a faulty row in one of the four banks. The Type 2 bitmap contains a lot of faulty columns (and some random faults) within one of the four banks, as exemplified in Fig. 13(b). For this memory design, a total of 22 fault patterns were found.

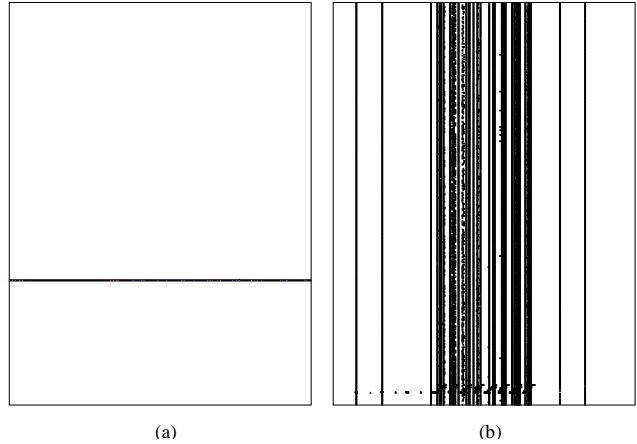


Figure 13. Partial bitmaps for two of the memory chips: (a) Type 1 bitmap, and (b) Type 2 bitmap.

For the Type 1 bitmap, the result is the same whether the memories are tested with the March C- or March 17N algorithm—the failed cells are all in one row. The failed cells were all detected by the second March element, $(ra, w\bar{a})$, where a can be 0 or 1. When tested by March 17N, the cells are all fault free at the second Read operation of the third March element, $(ra, w\bar{a}, r\bar{a})$, but partially failed at the first Read operation. A possible faulty behavior is that the ra operation in the second March element reads the data from the previous $w\bar{a}$ operation, as the data can be latched on the data bus due to the write through effect. This also is why the $r\bar{a}$ operation in the third March element detects no faulty cell. After the $r\bar{a}$ operation, the data latched on the data bus is erased to the default value, so the ra operation in the third March element detects some failed cells (while some cells are fault free). In this case, the failures cannot be modeled by any functional fault in the existing fault set, though they can be detected by the test algorithms that were designed based on the fault set. The fault pattern is “a row failure with random faults”, and the suspect defect is a word-line open.

Consider Fig. 14, which is a small block of the Type 2 bitmap shown above, with the previously found 22 fault patterns used for matching. To simplify the search, the fault pattern with more failed cells is assigned a higher priority,

due to higher probability. For example, a column failure with SAF0 can be identified as one FP2 or n FP11s. However, n concurrent FP11 defects are less likely than one FP2 defect. For fault pattern FP2, the whole column fails with SAF0. We can find three such cases in this bitmap. From the defect dictionary (Table 4), the possible defects are the short defect between VDD and BLb as well as that between GND and BL. For fault pattern FP1, the whole column fails with SAF1. We can find one such case in this bitmap. The possible defects are the short defect between GND and BLb as well as that between VDD and BL. As another example, consider FP4, where a whole column fails with random faults (with about 10% fault-free cells). One such fault pattern was found in this bitmap. The possible defects are the open defect on BL or BLb as well as the short defect between BL and BLb. If we had used only the failure patterns, these five failed columns looked the same. Obviously the fault patterns greatly improve the accuracy in isolating possible defects, resulting in lower debugging time. Note that one fault bitmap may contain multiple combinations of fault patterns. For example, three adjacent cells stuck at 1 may contain one FP13 and one FP12, or two overlapped FP13s.

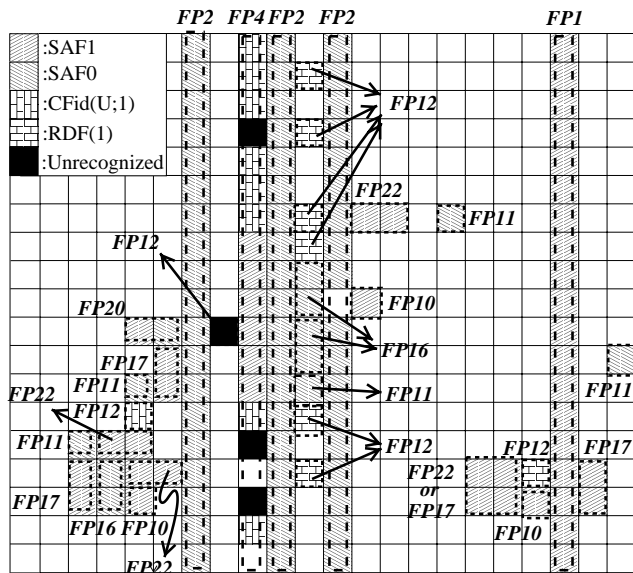


Figure 14. A partial fault bitmap.

Table 5 summarizes the experimental results. It lists the number of fault pattern occurrences, number of failed cells, and number of possible defects for each FP. For this target memory we have created 22 fault patterns with their specific defect information. As a result, there are seven FP1s in this experimental case, i.e., $6 \times 512 = 3072$ failed cells ($3072/21848 = 14\%$). No failed row with SAF was found in this case. With the proposed fault-pattern analysis approach, we partitioned the column failure patterns into 6 fault patterns (i.e., FP1, FP2, FP3, FP4, FP5, and

FP6), and reduced the possible defects from 9 (the union of FP1, FP2, FP3, FP4, FP5, and FP6) to only 2, reducing the search space for debugging. The failed cells that do not match any fault pattern are listed under the unknown pattern row, which represents only about 6.2% of all the failed cells. They may be caused by multiple defects in a single cell, a particle bigger than assumed, etc.

Table 5. Fault-pattern experimental results.

	FPS	Failed cells	Suspect defects
FP1	6	3072 (14%)	2
FP2	8	4095 (18.7%)	2
FP4	25	12019 (55.9%)	2
FP12	175	175 (0.8%)	3
FP13	90	180 (0.8%)	1
FP14	114	228 (1%)	1
FP16	86	172 (0.8%)	1
FP17	68	136 (0.6%)	1
FP21	83	166 (0.76%)	1
FP22	73	146 (0.67%)	1
Others*	50	107 (0.49%)	*
Unknown	—	1352 (6.2%)	—

*Others include FP10, FP11, FP15, FP18, FP19, and FP20. The corresponding number of candidate defects is shown in Table 4.

6. Conclusion

We have proposed a fault-pattern oriented methodology for semiconductor memory defect diagnostics, which greatly reduces the effort in memory product development and yield learning. The proposed notion of fault pattern combines the strengths of the conventional failure-pattern approach and our previous fault-type approach for easier isolation of real defects. Since the fault patterns have to be customized for different products and process technologies, automation is very important. We have also developed a systematic procedure to explore the fault patterns, which includes a layout-based defect injection tool that provides very accurate results from realistic defect models. With the proposed approach, high-quality defect diagnostics can be automated. An industrial case has been shown to justify the work. The main contribution of the paper is thus a methodology and procedure for accelerating FA and yield optimization for semiconductor memories.

7. Acknowledgment

The authors would like to thank Chuang Cheng and Kevin Chiu of Faraday Technologies, Hsinchu, for their assistance in our experiments.

References

- [1] Y. Zorian, "Embedded infrastructure IP for SOC yield improvement", in *Proc. IEEE/ACM Design Automation Conf. (DAC)*, New Orleans, June 2002, pp. 709–712.
- [2] Semiconductor Industry Association, "International technology roadmap for semiconductors (ITRS), 2001 edition", Dec. 2001.
- [3] J. Segal, A. Jee, D. Lepejian, and B. Chu, "Using electrical bitmap results from embedded memory to enhance yield", *IEEE Design & Test of Computers*, vol. 15, no. 3, pp. 28–39, May 2001.
- [4] Y. E. Hong, L. S. Leong, W. Y. Choong, L. C. Hou, and M. Adnan, "An overview of advanced failure analysis techniques for Pentium and Pentium Pro microprocessors", *Intel Technology Journal*, , no. 2, 1998.
- [5] 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.
- [6] I. Schanstra, D. Lukita, A. J. van de Goor, K. Veelen-turf, and P. J. van Wijnen, "Semiconductor manufacturing process monitoring using built-in self-test for embedded memories", in *Proc. Int. Test Conf. (ITC)*, Washington, DC, Oct. 1998, pp. 872–881.
- [7] M. A. Merino, S. Cruceta, A. Garcia, and M. Re-cio, "SmartBitTM: bitmap to defect correlation software for yield improvement", in *Advanced Semiconductor Manufacturing Conference and Workshop, IEEE/SEMI*, Boston, Sept. 2000, pp. 194–198.
- [8] R. Dekker, F. Beenker, and L. Thijssen, "Fault modeling and test algorithm development for static random access memories", in *Proc. Int. Test Conf. (ITC)*, 1988, pp. 343–352.
- [9] A. J. van de Goor and B. Smit, "Generating march tests automatically", in *Proc. Int. Test Conf. (ITC)*, 1994, pp. 870–878.
- [10] 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.
- [11] V. N. Yarmolik, Y. V. Klimets, A. J. van de Goor, and S. N. Demidenko, "RAM diagnostic tests", in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, San Jose, 1996, pp. 100–102.
- [12] C.-F. Wu, C.-T. Huang, C.-W. Wang, K.-L. Cheng, and C.-W. Wu, "Error catch and analysis for semi-conductor memories using March tests", in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (IC-CAD)*, San Jose, Nov. 2000, pp. 468–471.
- [13] 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.
- [14] V. Vardanian and Y. Zorian, "A march-based fault location algorithm for static random access memories", in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, France, July 2002, pp. 62–67.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] R. R. Montanes, J. P. de Gyvez, and P. Volf, "Resistance characterization for weak open defects", *IEEE Design & Test of Computers*, vol. 19, no. 5, pp. 18–26, Sept.-Oct. 2002.
- [19] A. J. van de Goor and J. E. Simonse, "Defining SRAM resistive defects and their simulation stimuli", in *Proc. Eighth IEEE Asian Test Symp. (ATS)*, Shanghai, Nov. 1999, pp. 33–40.