Built-In Redundancy Analysis for Memory Yield Improvement

Article *in* IEEE Transactions on Reliability · January 2004

DOI: 10.1109/TR.2003.821925 · Source: IEEE Xplore

4 authors, including:

Jin-Fu Li
National Central University
129 PUBLICATIONS 1,453 CITATIONS
SEE PROFILE

C.T. Wu
Industrial Technology Research Institute
552 PUBLICATIONS 12,255 CITATIONS
SEE PROFILE

Built-In Redundancy Analysis for Memory Yield Improvement

Chih-Tsun Huang, *Member, IEEE*, Chi-Feng Wu, *Member, IEEE*, Jin-Fu Li, *Member, IEEE*, and Cheng-Wen Wu, *Senior Member, IEEE*

***Abstract*—With the advance of VLSI technology, the capacity and density of memories is rapidly growing. The yield improvement and testing issues have become the most critical challenges for memory manufacturing. Conventionally, redundancies are applied so that the faulty cells can be repairable. Redundancy analysis using external memory testers is becoming inefficient as the chip density continues to grow, especially for the system chip with large embedded memories. This paper presents three redundancy analysis algorithms which can be implemented on-chip. Among them, two are based on the local-bitmap idea: the *local repair-most* approach is efficient for a general spare architecture, and the *local optimization* approach has the best *repair rate*. The *essential spare pivoting* technique is proposed to reduce the control complexity. Furthermore, a simulator has been developed for evaluating the repair efficiency of different algorithms. It is also used for determining certain important parameters in redundancy design. The redundancy analysis circuit can easily be integrated with the built-in self-test circuit.**

***Index Terms*—Built-in self-diagnosis, built-in self-test, DRAM, embedded memory, memory testing, redundancy analysis, SRAM, yield improvement.**

ACRONYMS

BIST	built-in self-test
BISD	built-in self-diagonsis
BISA	built-in self-analysis
BIRA	built-in redundancy-analysis
RM	repair-most
FC	fault collection
SA	spare allocation
LRM	locl repair-most
LO	local optimization
EP	essential pivot
ESP	essential spare pivoting

NOTATION

M	numer of rows of a memory array
N	number of columns of a memory array
l	fault line of a row or column
F	total number of faulty cells
F_l	number of faulty cells in l
$F\perp$	number of orthogonal faulty cells
r_a	available spare rows
C_a	available spare columns
n_r	number of faulty rows not covered
n_c	number of faulty columns not covered
m	number of rows of local bitmap
n	number of columns of local bitmap
B	local bitmap, where B^i is the ith row B_j is the jth column, and B^i_j indicates an individual flag, $0 \leq i \leq m, 0 \leq j \leq n$
S_R	row selection vector
S_C	column selection vector
C_R	row repair cost
C_C	column repair cost
Ω	$\Omega_{i=0}^{m-1}\vec{a}_i = \vec{a}_0\vert\vec{a}_1\vert\cdots\vert\vec{a}_{m-1}$, where $\vert\vec{a} = a_0a_1\cdots a_{n-1}$ is an n-bit vector and $\vert$ is the bitwise OR operator
$OR(\vec{a})$	$OR(\vec{a}) = a_0\vert a_1\vert\cdots\vert a_{n-1}$
$\vec{a}\&\vec{b}$	$\vec{a}\&\vec{b} = \{(a_0\cdot b_0),(a_1\cdot b_1),\ldots,(a_{n-1}\cdot b_{n-1})\}$
E_{th}	threshold number
P_R	row pivot
P_C	column pivot

Manuscript received January 1, 2002; revised March 1, 2003.

C.-T. Huang, C.-F. Wu, and C.-W. Wu are with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan (e-mail: cthuang@larc.ee.nthu.edu.tw).

J.-F. Li is with the Department of Electrical Engineering, National Central University, Chung-Li, Taiwan.

Digital Object Identifier 10.1109/TR.2003.821925

I. SUMMARY AND CONCLUSIONS

THREE algorithms suitable for built-in redundancy analysis have been proposed. The algorithms are simple compared to conventional analysis algorithms on the memory tester. Among them, two are based on the local-bitmap idea, i.e., the LRM and LO algorithms. Our analysis shows that the LRM algorithm is highly scalable for general memory and redundancy architectures, and the LO algorithm optimizes spare allocation in the local bitmap, resulting in the best repair rate. Also, the time overhead of LO is low for a practical spare architecture. The third approach, i.e., the ESP algorithm, does not require a bitmap. It greatly simplifies the control circuit, and results in the lowest time and area overhead among the three BIRA schemes. It was shown to achieve a high repair rate for a mature fabrication process with small area overhead. The estimated area overhead of an individual LRM circuitry is below 0.3% for a 64Mbit embedded DRAM core.

Analysis results by our simulator, BRAVES, justify the feasibility of these algorithms. BRAVES not only verifies the repair efficiency of the proposed algorithms, but also helps the designer select appropriate design parameters, such as the number of spare rows and columns, and the size of the local bitmap. In addition, other BIRA algorithms can also be simulated and evaluated systematically. It is convenient for selecting the proper analysis algorithm, and optimizing the spare architecture when given a fault size distribution and a yield distribution. The BIRA algorithm and circuit can also be optimized for the highest repair efficiency by using BRAVES.

II. Introduction

MOS memories are very important microelectronic components—they represent about 30% of the world semiconductor market [1]. We have seen continuous innovation in VLSI technology which keeps increasing the capacity and density of semiconductor memories, integrating more and more storage cells in a silicon chip; the capacity of memory chips roughly doubles every 18 months [2]. As we keep shrinking the feature size and increasing the integration density and chip size, keeping a profitable yield level is very hard. To avoid yield loss, redundancies (i.e., spare rows and columns of storage cells) are often added so that most faulty cells can be repaired (i.e., replaced by spare cells) [3]–[5]. Redundancy, however, adds to cost in another form because of the area overhead and potential yield loss. Therefore, analysis of redundancies to maximize yield (after repair) and minimize cost is an important process during manufacturing. Redundancy analysis using expensive memory testers is becoming inefficient (and therefore not cost-effective) as the chip density continues to grow. The use of embedded memories creates yet another problem; embedded memories are even harder to deal with using external testers [6]. It has been widely noted that embedded memories will play an increasingly important role in the semiconductor market over the next few years, because the system-on-chip market is booming and almost every system chip contains some type of embedded memory. There is a prediction that embedded memories will dominate more than 90% of the system chip area within a decade while the chip complexity keeps increasing dramatically [7].

Testing embedded memory is more difficult than testing commodity memory in general, unless built-in self-test (BIST) is used [6], [8]–[13]. Although BIST is a promising solution, current BIST schemes are only for functional testing, so they cannot replace external memory testers entirely. Extension of BIST to built-in self-diagnosis (BISD) has been shown to be feasible [14], [15]. However, BIST with diagnosis support is still not enough because the large amount of diagnosis data is needed to transfer through the channel with limited bandwidth to the external tester. Therefore, built-in redundancy-analysis (BIRA) and built-in self-repair (BISR) are now among the top items to be incorporated with the memory core.

Several BIRA/BISR techniques have been proposed in the past, e.g., [14], [16]–[25]. In [17], a DRAM with BIST and BISR circuits has been presented, using spare rows of SRAM cells as the redundancy. The design put emphasis on the power-on self-test and self-repair at the memory board level. In [18], a hierarchical BISR structure has been proposed to facilitate memory test, fault location, and self repair. In [14], a microprocessor-based BISD circuit was presented for repairable embedded SRAM with programmable modules. A BISR scheme using neural network circuits was proposed in [19]. Moreover, an embedded high-density SRAM with BISR capability has been proposed [20], allowing autonomous repair of the SRAM core with negligible timing penalty. The spare architecture was simplified (it has only spare columns), so a simple greedy algorithm can be used for diagnosis and self-repair. Recently, a self-test and self-repair technique has been implemented using spare row and spare column clusters with small hardware overhead [21]. Several segment-based or block-based redundancy architectures have been proposed to improve the repair efficiency while keeping the simplicity of the analysis algorithms [22], [23]. A built-in self-repair analyzer for embedded DRAMs has been proposed [24]. Moreover, its shared analyzer architecture has been presented for multiple memory cores in the system chip [25]. The exhaustive repair permutations are calculated in parallel under restricted redundancy architecture, i.e., two spare rows and two spare columns for each block. Both the area overhead and time complexity are limited for the specific redundancy architecture. Most of the previous works either assume a simplified redundancy architecture, i.e., they assume only spare rows or spare columns are available, but not both, or derive the algorithm suitable for external memory tester with powerful computation capability. The approaches using a 2-D spare architecture also assume limited number of spare lines to restrict the complexity of repair permutations. However, high-density memories normally have a 2-D spare architecture (with both spare rows and columns) which is better in terms of repair efficiency. The efficiency of the spare architecture depends on the manufacturing process. Therefore, restricted spare architecture limits the efficiency of yield improvement. In addition, trade-off between hardware complexity and efficiency of the analysis algorithms has to be weighed if we consider BIRA.

In this paper, we propose three BIRA algorithms. We will show that, by our approaches, high repair rate and low area overhead can be achieved. The BIRA (or Built-In Self-Analysis (BISA) [24], [25]) techniques are flexible and can cooperate both with the external tester and BIST. Therefore, the proposed techniques facilitate not only the ATE and laser-fuse based repair for better yield [21], but also the in-field BIST and soft-fuse based repair for higher reliability [17], [20], [21]. In addition, when cooperating with BIST, redundancy analysis can be done concurrently when testing, so there is little time penalty for the analysis.

The rest of the paper is organized as follows. The Section III briefly reviews the conventional redundancy analysis algorithms. In Section III, three redundancy analysis algorithms will be proposed. Two of them are based on the proposed local bitmap concept, i.e., the *local repair-most* (LRM) and *local optimization* (LO) approaches. The LRM algorithm is suitable for a general memory architecture. It is an improvement over the conventional *repair-most* algorithm for BIRA. The LO algorithm performs an exhaustive search within the local bitmap to obtain a high repair rate. Our third algorithm does not require the local bitmap. It is especially suitable for an imbalanced redundancy architecture where, e.g., the number of spare rows is greater than the number of spare columns. Imbalanced redundancy architectures can be found in many modern high-density memories. In Section IV we will show simulation results to demonstrate the efficiency of the proposed schemes. The simulator which we have developed, called BRAVES (Built-in Redundancy Analysis Verification and

Evaluation System), is also a handy tool for spare architecture design. Conclusions will be given in the last section.

III. Conventional Redundancy Analysis Algorithms

Before the discussion of more efficient approaches, we define the terminology and notation first. A memory *block* consists of M rows and N columns of storage cells, i.e., an $M \times N$ array of cells. The origin of the cell array is the upper left corner. There are r spare rows and c spare columns. We follow the definitions given in [26].

Definition 1: A faulty line l is either a row or a column on which one or more faulty cells exist. The number of faulty cells in l is F_l. A faulty line is either a faulty row or a faulty column.

Definition 2: A faulty line is said to be covered if all faulty cells on the line have been scheduled to be repaired by specific spare rows and/or spare columns.

Definition 3: A faulty cell which does not share any row or column with any other faulty cell is referred to as an orthogonal faulty cell.

Let the numbers of available spare rows and available spare columns during the analysis process be denoted as r_a and c_a, respectively. During the redundancy analysis, any faulty line which consists of k faulty cells requires either 1 spare line in the same direction or k perpendicular spare lines.

Definition 4: Any faulty row (column) with k faulty cells (i.e., $F_l = k$), where $k > c_a (k > r_a)$, is a must-repair faulty line.

Lemma 1: For the must-repair faulty row (column) l, its F_l can be limited to $c + 1 (r + 1)$ without affecting the fact that the faulty line l is a must-repair line.

If there are $c + 1$ or more faulty cells on the faulty row, it is a must-repair row (see Definition 4). And Lemma 1 indicates that a fault counter which counts up to $c + 1$ can be used for the must-repair analysis.

Conventionally, redundancy analysis is done on a memory tester using software. The tester stores the bitmap (a map of the faulty cells) after a diagnostic test, and performs redundancy analysis based on the bitmap. Software analysis is slow, so normally only simple heuristic algorithms are used. Most such algorithms consist of two phases: the *must-repair* phase followed by the *final-repair* phase. In the must-repair phase, all the must-repair faulty lines are identified first by counting the number of faulty cells (F_l) for each faulty line, limiting the number of remaining faulty cells. With Lemma 1, the storage to record F_l for each faulty line can be restricted effectively. In the final-repair phase, simple algorithms, such as the *repair-most* algorithm or *fault-driven* algorithm, are used. The repair-most approach calculates F_l of the remaining faulty lines and selects the faulty line with the largest F_l for the repair one by one; while the fault-driven algorithm maps the redundancy analysis to the tree-searching problem with branch-and-bound heuristics, exploring possible combinations of row/column selection. For example, a greedy redundancy analysis algorithm called the *repair-most* (RM) algorithm was proposed in [27] for the final-repair phase. Error counters for the respective faulty rows and columns are required in the RM analysis. On the other head, the fault-driven algorithm (based on exhaustive search)

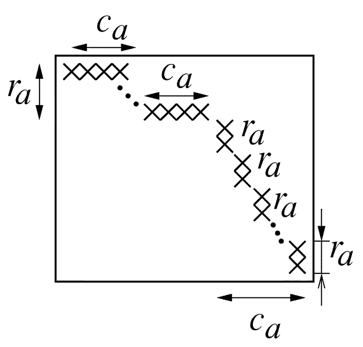

Fig. 1. A defective memory block of the worst case.

generates all possible spare allocation for finding the optimal one [28]. The exhaustive search approach is slow. Also, that finding the optimal solution is NP-complete has been proven by transforming it in polynomial time to the bipartite-graph clique problem [29]. Branch-and-bound and approximation algorithms can be used to reduce the search time. An improved approach called the *faulty-line covering* technique and a heuristic criterion allowing fast repair were later proposed in [26]. In addition, there are other redundancy analysis techniques which are mainly for fast repairability decision [30], [31]. Most of the conventional redundancy analysis approaches, however, assume the availability of a memory tester with high computing power, memory capacity, and flexibility. Such memory testers would be very expensive. It is clear that the analysis time (or tester throughput) is critical so far as cost is concerned [31]. The use of BIRA (with BIST) will greatly increase the tester throughput.

Unlike the fault-driven approach, the repair-most approach is straightforward and simple. The F_l values of all faulty lines are calculated during the construction of the bitmap. The faulty lines are ordered according to their F_l values—the first line to be repaired is the one with the largest F_l value. Row and column counters are still required for keeping the fault counts of the faulty lines. Apparently the bitmap is a necessary tool for the repair-most analysis.

Let F denote the total number of faulty cells in the memory block, and $F_\perp$ represents the number of orthogonal faulty cells. Two important early termination conditions for redundancy analysis are as follows.

Condition 1: After the must-repair phase, $F > 2r_a c_a$.

Condition 2: $F_\perp > r_a + c_a$.

If either condition is true, then the analysis process stops, early termination conditions help to identify the memory which can not be repaired by available spares. Due to Lemma 1, we have the following two additional early termination conditions.

Condition 3: $n_r > r_a$ if $c_a = 0$, where n_r is the number of faulty rows not covered so far.

Condition 4: $n_c > c_a$ if $r_a = 0$, where n_c is the number of faulty columns not covered so far.

If any of the above four conditions is true, then the memory block is unrepairable.

A defective memory block of the worst case is shown in Fig. 1, where $F = 2r_a c_a$ after the must-repair phase, and all the available spare rows and columns have to be used for repairing the faulty lines. Specifically, the available spare rows cover r_a faulty rows, where each faulty row has c_a faulty cells, and among these faulty rows, no cells share the same column address. Therefore, the faulty cells in these rows are located in $c_a \times r_a$ different columns. Similarly, the c_a faulty lines covered by the available spare columns have $r_a \times c_a$ row addresses for the faulty cells. In this special case, the width of the bitmap is $c_a r_a + c_a$ and the height is $r_a c_a + r_a$. Therefore, in general, the size of the bitmap can be limited to $(r_a c_a + r_a) \times (c_a r_a + c_a)$ instead of $M \times N$ after the must-repair phase. Moreover, without the must-repair phase, the maximum size of the bitmap can be limited to $(r(c+1)+r) \times (c(r+1)+c)$ due to Lemma 1.

In addition to the bitmap, a total of $(r_a c_a + r_a)$ row counters, and $(c_a r_a + c_a)$ column counters are required. Each row counter has $\log(c_a + 1)$ bits, and each column counter has $\log(r_a + 1)$ bits. The row counter for a faulty row l stores F_l, which is tagged by the $\log N$-bit row address. Similarly, the column counter for a faulty column l stores F_l, which is tagged by the $\log M$-bit column address.

The hardware implementation of the fault-driven algorithms is complicated because of the complex control of tree expansion, such as back-and-forth or stack operation, including those using exhaustive search and those using heuristics. In addition, the tree searching adds significant time penalty for the built-in circuitry. Therefore, the concepts of the proposed BIRA algorithms are based on the simple redundancy analysis approach, i.e., the repair-most algorithm, which consists of counting and comparing operations and is suitable for hardware design in nature.

IV. New Built-In Redundancy Analysis Approaches

A. *Repair-Most Using Local Bitmap*

Storing the full bitmap on-chip for the purpose of redundancy analysis obviously is not feasible. Our goal is for the BIRA circuit to properly allocate redundancies in parallel with the BIST operation. The *repair rate* of the redundancy analysis is defined as the ratio of good memories after the redundancy analysis and all the memories. The area overhead should be low and the repair efficiency should be high, where the *repair efficiency* is defined as the repair rate with respect to unit area overhead. The proposed algorithms require only a small array of size $m \times n$ for storing the *local bitmap*. This greatly reduces the silicon overhead for the bitmap and the row/column counters. The parameters, m and n, can be constant or proportional to r and c, respectively, depending on the defect distribution. We will show later that the local-bitmap technique results in a fast and small BIRA circuit with good repair efficiency, and the BIRA circuit can be easily integrated with the BIST circuit.

The Local Repair-Most (LRM) algorithm is shown in Fig. 2, where T_R and T_C are the row and column address tags, respectively. The address tags (i.e., registers) store the m row and n column addresses of the faulty cells which are recorded in the local bitmap. The local bitmap B consists of $m \times n$ flags, where B^i is the ith row, B_j is the jth column, and B^i_j indicates an individual flag, respectively. The redundancy analysis algorithm consists of two functions, i.e., *fault collection* (FC) and *spare allocation* (SA). They are shown in the figure as LRM_FC() and LRM_SA(), respectively. The former collects the faulty-cell addresses and constructs the local bitmap, and the latter performs spare allocation when necessary.

```
LRM_FC() {
    $r_a = r$; $c_a = c$;
    foreach (faulty_cell_detected()) {
        $(\hat{R}, \hat{C})$ = faulty_address();
        $p_x$ = Matching_tag_index($T_R$, $\hat{R}$) or Next_available_tag($T_R$) ;
        $p_y$ = Matching_tag_index($T_C$, $\hat{C}$) or Next_available_tag($T_C$) ;
        if ($p_x \geq m$ or $p_y \geq n$) { LRM_SA(); }
        $T_{R_{px}} = \hat{R}$;
        $T_{C_{py}} = \hat{C}$;
        $B^{p_x}_{p_y} = 1$;
    }
    LRM_SA();
}

LRM_SA() {
    if ($r_a$==0 or $c_a$==0) {
        Check_Early_Termination();
        if ($r_a$==0) Allocate_spare_column($T_{C_j}, \forall j$);
        if ($c_a$==0) Allocate_spare_row($T_{R_i}, \forall i$);
    } else {
        for all $i$, $0 \leq i < m$ {
            $C_{R_i}$= Ones_count($B^i$);
            if ($C_{R_i} > c_a$) { Allocate_spare_row($T_{C_{R_i}}$); }
            }
        for all $j$, $0 \leq j < n$ {
            $C_{C_j}$= Ones_count($B_j$);
            if ($C_{C_j} > r_a$) { Allocate_spare_col($T_{C_{C_j}}$); }
            }
        $C_{R_{pr}}$ = max($C_{R_i}$, $0 \leq i < m$);
        $C_{C_{pc}}$ = max($C_{C_j}$, $0 \leq j < n$);
        if ($C_{R_{pr}} > C_{C_{pc}}$) { Allocate_spare_row($T_{R_{pr}}$); }
        else if ($C_{R_{pr}} < C_{C_{pc}}$) { Allocate_spare_column($T_{C_{pc}}$); }
        if ($C_{R_{pr}}$==$C_{C_{pc}}$) {
            if ($r_a \geq c_a$) { Allocate_spare_row($T_{R_{pr}}$); }
            else { Allocate_spare_column($T_{C_{pc}}$); }
        }
    }
}
```

Fig. 2. The local repair-most algorithm.

The BIST logic will activate the BIRA process whenever a faulty cell is detected during testing. The faulty cell address will be latched and compared with the address tags as shown in LRM_FC(). If the row or column address of the current faulty cell is different from that of any previously stored (and tagged) faulty cell in the local bitmap, a new address tag is used and the next available position in the bitmap is allocated. If the current row (column) address is the same with a tagged one, then the row (column) tags remain the same. In LRM_FC(), p_x and p_y are location indices of the bitmap for the current faulty cell. If $p_x \geq m$ or $p_y \geq n$, the bitmap is full and incremental repair should be performed immediately, i.e., LRM_SA() should be called.

LRM_SA() is used to allocate a spare line for the faulty line with the largest number of faulty cells. Whenever a spare line is used to replace a faulty line, the corresponding entries in the local bitmap have to be cleared. Note that the repair of a faulty line may not release enough room in the bitmap for a new faulty cell. For example, consider the bitmap as shown in Fig. 3. If the address of the next faulty cell is $(\hat{R}, \hat{C}) = (9, 9)$, then repairing neither Row 1 nor Row 5 can release a blank column in the bitmap for the new faulty cell which has a column address of 9. In this case, additional lines are selected until the new address can be recorded in the bitmap.

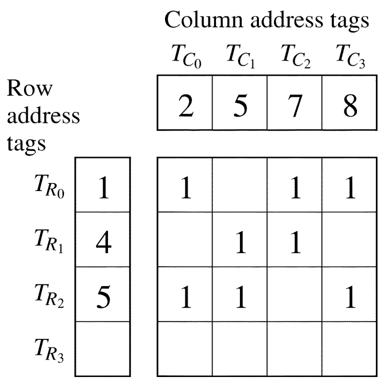

Fig. 3. A local bitmap example.

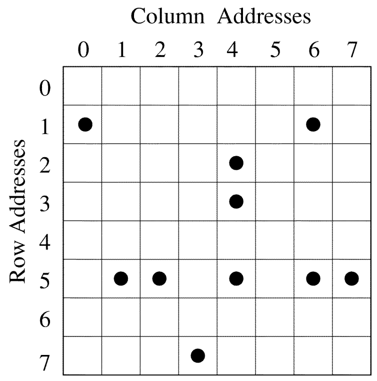

Fig. 4. A memory block with defects.

As an example for illustrating our LRM algorithm, consider the memory block as shown in Fig. 4, where we assume $r = c = 2$. Also, assume we perform a row-wise march algorithm during BIST. The physical cell location is also provided to facilitate the analysis. The LRM analysis process for the memory block is shown in Fig. 5. After the first five faulty cells have been detected, the bitmap is as shown in Fig. 5(a). The sixth faulty cell (i.e., cell(5,2)) has the same row address as the fourth row address tag of the current bitmap, but a column address different from any existing one. Since the bitmap is full, LRM_SA() is called. Row 1 is selected (shaded in the figure) for repair at this stage, resulting in the bitmap as shown in Fig. 5(b). The next 3 faulty cells (i.e., cell(5,4), cell(5,6), and cell(5,7)) are all located in the same row (i.e., Row 5) as shown in Fig. 5(c), so Row 5 is selected for repair next. Finally, after all faulty cells are processed, Column 4 and Column 3 are selected in sequence for repair according to their weights and the type of remaining spare lines, as shown in Fig. 5(c). In summary, Row 1, Row 5, Column 4, and Column 3 are replaced by spare lines after the repair process.

To search for the faulty line with the largest fault count, two heuristic rules are applied:

1) The address of the newly detected faulty cell is used together with the current bitmap to help determine the line with the largest fault count. It is possible that the row address is previously recorded in the bitmap but there is no room for the column address, or vice versa.
2) Any row (column) with a fault count exceeding $c_a(r_a)$ is regarded as a must-repair faulty row (column).

Additionally, Condition 3 or 4 can be applied if $r_a = 0$ or $c_a = 0$. Condition 1 can be used for early termination if there is a fault counter for keeping the total fault number and the fault count from the must-repair faulty lines can be dynamically subtracted from the total fault count. Early termination by Condition 2 can be applied with additional address registers for the $r + c$ orthogonal faults. This is implemented in our simulation for the LO (Local Optimization) algorithm to be described next.

B. Local Optimization

In LRM, selecting the faulty line with the largest fault count is time consuming, because it may need to pick more than one line for repair (the number of lines to be repaired is a variable). Additionally, the repair rate of LRM is limited by the bitmap size. To cope with the problem, we propose another algorithm, i.e., the *Local Optimization* (LO) algorithm. The LO algorithm has a better repair rate, though it also uses the local bitmap. It searches for all possible ways of spare allocation when the bitmap is full. No row or column counters are needed for counting the faults as in LRM.

The LO algorithm is shown in Fig. 6. It consists of two functions: LO_FC() and LO_SA(). Similar to LRM_FC(), LO_FC() records the faulty cells in the local bitmap until it is full. When the bitmap is full, an exhaustive search is performed to allocate the spares to cover *all* the recorded faults. After that, the bitmap is cleared. Note that as opposed to LRM, the bitmap is entirely cleared after every call to LO_SA(), simplifying the control. Without loss of generality, assume the size of the local bitmap is $m \times n$, where $n < m$. The proposed scheme uses an n-bit counter, E, which produces an exhaustive binary sequence. The n-bit counter contains the bits $\{e_0, e_1, \ldots, e_{n-1}\}$, where $e_i = 1$ means that the column pointed to by the ith column address tag is selected for repair. The flag e_i has to be reset if the ith column address tag is empty. We assume column selection (search) has a lower cost than row selection (i.e., $n < m$). After column selection, the corresponding columns of the bitmap are cleared, and the next step is to select rows to cover the remaining faults.

As an example, let $E = \{0101\}$ and assume the current bitmap is as shown in Fig. 7. The second and fourth columns are selected for repair as indicated by the content of E. However, the fourth column is empty, so only the second column (with column address 5) is selected for repair. This is determined by the *column selection vector* whose value is $S_C = \{0100\}$. Similarly, the *row selection vector* is $S_R = \{0010\}$, indicating that there are two remaining faults in the 3rd row of the bitmap (with row address 5) after column selection, so Row 5 will be selected after Column 5 for subsequent repair.

Note that both S_R and S_C can be calculated by logical (AND-OR) operations between the local bitmap rows and E. The computation of S_R and S_C is shown in Fig. 6, and formulated as follows. Let the function Ω be defined as

$$\Omega_{i=0}^{m-1}\vec{a}_i = \vec{a}_0|\vec{a}_1|\cdots|\vec{a}_{m-1}, \tag{1}$$

where $\vec{a} = a_0a_1\cdots a_{n-1}$ is an n-bit vector and $|$ is the bitwise OR operator. Also, we define

$$OR(\vec{a}) = a_0|a_1|\cdots|a_{n-1} \tag{2}$$

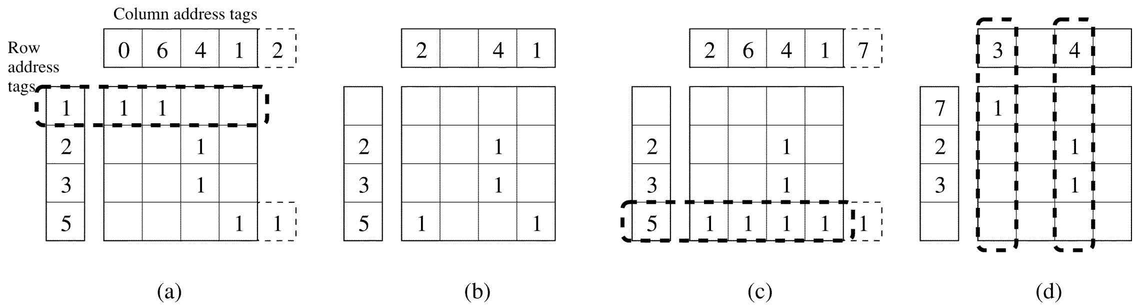

Fig. 5. LRM analysis process.

```
LO_FC() {
    $r_a = r$; $c_a = c$; $i_x = i_y = 0$;
    foreach (faulty_cell_detected()) {
        ($\hat{R}$, $\hat{C}$) = faulty_address();
        $p_x$ = Matching_tag_index($T_R$, $\hat{R}$) or $i_x$++;
        $p_y$ = Matching_tag_index($T_C$, $\hat{C}$) or $i_y$++;
        if ($p_x \geq m$ or $p_y \geq n$) {
            LO_SA();
            $p_x = p_y = 1$;
        }
        $T_{R_{p_x}} = \hat{R}$;
        $T_{C_{p_y}} = \hat{C}$;
        $B_{py}^{px} = 1$;
    }
    LO_SA();
}

LO_SA() {
    if ($r_a$==0 or $c_a$==0) {
        Check_Early_Termination();
        if ($r_a$==0) Allocate_spare_column($T_{C_j}, \forall j$);
        if ($c_a$==0) Allocate_spare_row($T_{R_i}, \forall i$);
    } else {
        for each $E$, $0 \leq E < 2^n$ {
            $S_R$ = {$OR(B^0 \& \overline{E})$, ..., $OR(B^{m-1} \& \overline{E})$};
            $S_C$ = $\Omega_{k=0}^{m-1}(B^k \& E)$;
            $C_R$ = Ones_count($S_R$);
            $C_C$ = Ones_count($S_C$);
            $C_{cur}$ = $C_R$ + $C_C$;
            if ($C_R \leq r_a$ and $C_C \leq c_a$ and $C_{cur} < C_{min}$)
                { $C_{min} = C_{cur}$; $S_{R,min} = S_R$; $S_{C,min} = S_C$; }
        }
        for each $i$, $0 \leq i < m-1$
            { if ($i$-th bit of $S_{R,min}$ == 1) { Allocate_spare_row($T_{R_i}$); } }
        for each $j$, $0 \leq j < n-1$
            { if ($j^{th}$ bit of $S_{C,min}$ == 1) { Allocate_spare_column($T_{C_j}$); } }
    }
}
```

Fig. 6. The local optimization algorithm.

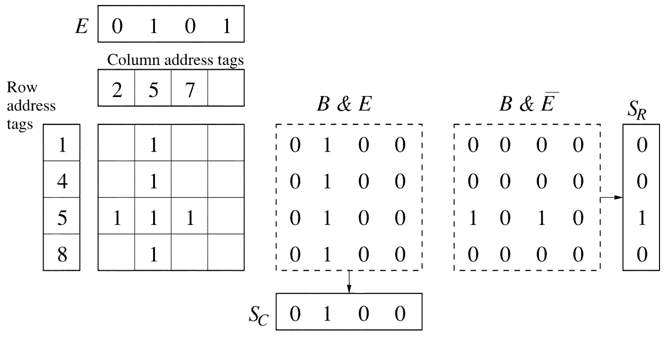

Fig. 7. Selection of columns to be repaired in local optimization.

and

$$\vec{a} \& \vec{b} = \{(a_0 \cdot b_0), (a_1 \cdot b_1), \ldots, (a_{n-1} \cdot b_{n-1})\}. \tag{3}$$

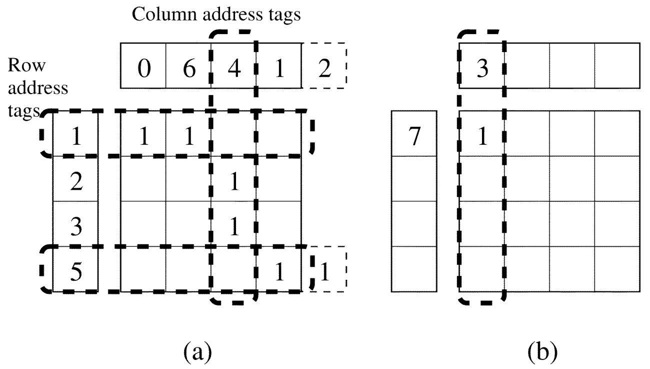

Fig. 8. An example for local optimization.

The row selection vector (S_R), column selection vector (S_C), row repair cost (C_R), and column repair cost (C_C) are calculated as follows (see also LO_SA() in Fig. 6).

$$\begin{aligned} S_R &= \{OR(B^0 \& \overline{E}), \ldots, OR(B^{m-1} \& \overline{E})\}, && (4) \\ S_C &= \Omega_{k=0}^{m-1}(B^k \& E), && (5) \\ C_R &= \text{Ones\_count}(\mathrm{S_R}), && (6) \\ C_C &= \text{Ones\_count}(\mathrm{S_C}), && (7) \end{aligned}$$

where B^k represents the kth row vector of the local bitmap. The vector S_C represents E excluding the blank column tags, and the vector S_R represents the rows with the remaining faults after column selection. In LO_SA(), all possible values of E are counted, i.e., we try all possible combinations of column selection. The optimal repair selection for each LO_SA() call can be obtained using

$$C_{\min} = \min_{\forall E}\{aC_R + bC_C\}, \tag{8}$$

where a and b are the weights of spare row selection and spare column selection, respectively. The weights are determined by the respective costs of an implementation. Once the minimum cost is found, the corresponding selection vector is used to allocate the spare lines.

Consider the same memory block as shown in Fig. 4 again. Let the weights be assigned as $a = b = 1$ for simplicity. The bitmap construction process is the same with LRM for the first 6 faults, as shown in Fig. 8(a). However, an exhaustive search is performed after that to cover all faults recorded in the bitmap, and Row 1, Row 5, and Column 4 are selected upon the detection of the sixth faulty cell, cell(5,2), by calculating $S_{R,\min}$ and $S_{C,\min}$. Assume that after this search, all the remaining faults are covered except the last fault from cell(7,3). As shown in Fig. 8(b), the corresponding column, i.e., Column 3, is selected if only one spare column is left. The LO algorithm usually requires a longer execution time (due to LO_SA()) than LRM for large m and n values, because exhaustive search is performed. However, for an *imbalanced* spare architecture (i.e., $r \neq c$), LO can take advantage of it by counting all combinations of E along the direction with less spare lines. Moreover, faults recorded in the bitmap are all covered after every LO_SA()) call, reducing the number of total iterations.

The heuristics and early termination conditions of LRM still apply to LO as discussed previously. In addition, a heuristic which deals with orthogonal faults, based on Condition 2, can be used to improve the repair rate. The heuristic is as follows. Every incoming fault which does not match any of the row and column address tags in the local bitmap is stored in an additional *orthogonal fault register* (OFR). When a fault is detected, we need to perform an additional comparison of the fault with those in the OFRs. If there is a row or column match then both the matched and matching addresses are recorded in the bitmap, with the one in the OFR cleared. Orthogonal faults in the OFRs are processed after all other faults are covered. The remaining spare lines are selected to cover those orthogonal faults, one by one, using any order. This heuristics obviously complicates the control, but improves the repair rate greatly. Condition 2 can be applied together with this heuristics.

C. Essential Spare Pivoting

We have shown algorithms which use a small bitmap instead of the full bitmap. Because repair-most algorithms rely on a bitmap as the basic tool for redundancy analysis, it seems that a bitmap is inevitable. This is not true. In this section we propose the *essential spare pivoting* (ESP) algorithm without using a bitmap. We will show later that the repair rate using ESP is still high, with a greatly simplified implementation and reduced area.

Following the previous discussion, we observe the following general guidelines for redundancy analysis.

1) A faulty row is more suitable for row repair when there are more faults in the row than in any of the columns of the corresponding faulty cells in the row. Likewise, a faulty column is more suitable for column repair when there are more faults in the column than in any of the rows of the corresponding faulty cells in the column.
2) An orthogonal fault can be repaired by either a spare row or a spare column. Orthogonal faults should be processed after all others.

The first guideline leads to the repair-most-based algorithms. A full bitmap or various local bitmaps are required to perform the analysis. Because our goal is to repair without bitmaps, we revise the first guideline as follows.

- For any faulty row (column), if the number of faulty cells is greater than or equivalent to a threshold number (E_{th}), repair it by a spare row (column).

This guideline is similar to the must-repair rule, except that the decision is based on a customized threshold number, E_{th}, instead of r_a and c_a. In the analysis procedure, we maintain a counter for the number of faults in each faulty line. When the

```
ESP_FC() {
    $p = 0$; $r_a = r$; $c_a = c$;
    foreach (faulty_cell_detected()) {
        $(\hat{R}, \hat{C})$ = faulty_address();
        if ($\exists\ 0 \leq i < p$ such that $P_{R_i} == \hat{R}$) { $E_{R_i} = 1$; }
        else if ($\exists\ 0 \leq j < p$ such that $P_{C_j} == \hat{C}$) { $E_{C_j} = 1$; }
        else {
            $P_{R_p} = \hat{R}$;
            $P_{C_p} = \hat{C}$;
            $p$++;
            if ($p > r + c$) { repair_termination("unrepairable"); }
        }
    }
    ESP_SA(); }

ESP_SA() {
    for all $k$, $0 \leq k < p$ {
        if ($E_{R_k}$ == 1) { allocate_spare_row($P_{R_k}$); }
        if ($E_{C_k}$ == 1) { allocate_spare_column($P_{C_k}$); }
    }
    for all $k$, $0 \leq k < p$ {
        if ($E_{R_k} == E_{C_k}$ == 0)
            if ($r_a > 0$) { allocate_spare_row($P_{R_k}$); }
        else if ($c_a > 0$) { allocate_spare_column($P_{C_k}$); }
        else { repair_termination("unrepairable"); }
    }
}

allocate_spare_row($\hat{R}$)
    { $R_{R_{r_a}} = \hat{R}$; $r_a = r_a - 1$; }
allocate_spare_column($\hat{C}$)
    { $R_{C_{c_a}} = \hat{C}$; $c_a = c_a - 1$; }
```

Fig. 9. The essential spare pivoting algorithm.

number reaches E_{th}, it is marked as an *essential line*. We assign E_{th} to be 2, so threshold comparison is greatly simplified. The second guideline shown above states that an orthogonal fault should be recognized early but processed after all other. The reason is that, e.g., while $c_a > 0$ and $r_a > 0$, if we repair an orthogonal fault by a spare row before repairing other nonorthogonal faults, we may lose the chance to repair more faults with this spare row, because that orthogonal fault can also be repaired by a spare column. With this guideline and the proposed guideline, we have developed a new algorithm called the *essential spare pivoting* (ESP) algorithm, which is shown in Fig. 9.

The function ESP_FC() collects the faulty-cell addresses and stores them in the P_R (*row pivot*) and P_C (*column pivot*) register files. Both have $r + c$ registers, and all the registers are initially empty. An incoming faulty-cell address $(\hat{R}, \hat{C})$ is compared with the existing row pivots and column pivots in the register files. If there is a row-address match or column-address match, the matched pivot is marked as an *essential pivot* (EP). During the FC phase, if the number of the pivot pairs exceeds $r + c$, this memory is unrepairable and the process terminates. If there is no match, the row address and column address of the current faulty cell are stored in the P_R and P_C registers, respectively. In ESP_FC() we apply the revised first guideline. Note that in the algorithm shown in Fig. 9, We will show in Section V that the repair rate is high when $E_{th} = 2$, and only a flag is needed along with each pivot to indicate whether it is an EP.

The function ESP_SA() allocates spares to repair faults according the contents of the P_R and the P_C registers. It consists of two stages. In the first stage we allocate spare rows for the essential row pivots and spare columns for the essential column

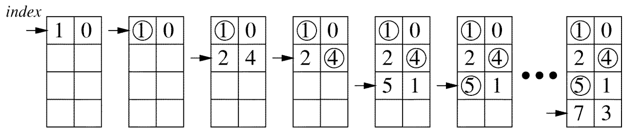

Fig. 10. Fault collection example in ESP.

pivots. After the first stage, the pivot registers contain all and only the addresses of the orthogonal faults, because they have never matched other faulty-cell addresses. We can repair these faults by either spare rows or spare columns. In ESP_SA(), we simply allocate available spare rows before spare columns.

Now we explain the ESP algorithm by an example. The memory block under test is the same as shown in Fig. 4. The faulty cells detected are, in sequence, cell(1,0), cell(1,6), cell(2,4), cell(3,4), cell(5,1), cell(5,2), cell(5,4), cell(5,6), cell(5,7), and cell(7,3). The FC procedure is illustrated in Fig. 10. In the figure, the P_R and P_C registers are shown as the left and right columns of the register array, respectively. For each faulty-cell address, the P_R is stored in the left column and the P_C is stored in the right column. There is a circle on a pivot if it is an essential pivot. In the beginning, the register array is empty, so the first address (1,0) is stored in the first row of the array directly. The second address (1,6) matches (1,0) in the row address, so the P_R of cell(1,0) is marked as an EP. Similarly, the address (2,4) is inserted directly, while the address (3,4) matches (2,4) in its column address, thus the P_C of cell(2,4) is marked as an EP. This procedure continues until the address (7,3) is recorded. The SA procedure is simple: first we allocate spares for the EPs—Row 1, Row 5, and Column 4, then we use a spare column to repair the orthogonal fault on cell (7,3).

The major advantage of the ESP is mainly its simple in implementation, which results in smaller area overhead than other algorithms. The revised first guideline provides a simple search method for orthogonal faults. In the SA stage of the ESP algorithm, orthogonal faults and nonorthogonal faults can be easily separated by checking their EP flags. The automatic recognition for orthogonal faults greatly increases the repair efficiency. These features make the ESP algorithm small, fast, and easily implementable.

The proposed BIRA algorithms are presented for localized redundancy architecture. However, the cost function in spare allocation can be easily adapted for the redundancy architecture with global or shared spare resources. With the help of an evaluation tool for repair efficiency, the most effective BIRA algorithm with optimized spare architecture can be found for specific manufacturing process systematically.

V. Repair Efficiency and Overhead Analysis

We have developed a simulator, called BRAVES (Built-in Redundancy Analysis Verification and Evaluation System), for analyzing the efficiency of the redundancy analysis and repair algorithms. The distribution of defect sizes on memory chips usually is modeled by mixed Poisson statistics using the Gamma distribution, resulting in a Polya-Eggenberger distribution [32]. Fig. 11 shows the overall flow of BRAVES. The input data consists of three categories: the distribution of the faulty cells and

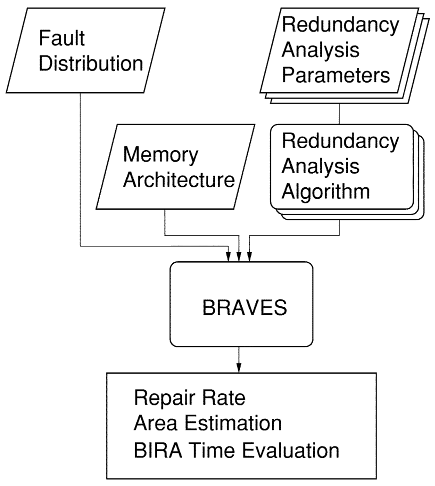

Fig. 11. Overall flow of BRAVES.

faulty lines; the memory architecture, including the geometry of the cell array and its redundancy architecture; and the parameters for each redundancy analysis algorithm, such as the size of the local bitmap, the specified threshold E_{th}, etc. Under the given memory and redundancy information, BRAVES generates the memory samples and injects faults accordingly. In addition, different redundancy algorithms are simulated (e.g., optimal algorithm, most repair algorithm, LRM, LO and ESP, etc.). The users can assign different parameters for each algorithm and evaluate the resultant repair rate and estimated area overhead, choosing the BIRA algorithm with the best repair efficiency. Furthermore, the hardware execution time of each BIRA can be evaluated if the cycle time is modeled in the simulator properly. Therefore, BRAVES provides a systematic methodology for the redundancy architecture and the BIRA design.

Recently, some extensive and robust yield analysis, estimation and reliability measurement based on a realistic fault model have been proposed [33], [34]. In BRAVES, we assume a mixed Poisson and exponential model, because it is sufficiently accurate yet simple. Moreover, different conditions can be applied in our simulator for different redundancy analysis algorithms. The comparison among different algorithms can be done and their differences can be displayed graphically. There are two types of faults which we inject into the memory to be simulated, i.e., the *cell fault* and *line fault*. The cell fault represents an independent individual fault, while a line fault occurs when multiple faults exist in the same line, such as the case of a faulty wordline or bitline. Because of the lack of the overall bitmap, our BIRA approaches are sensitive to the detecting order of the faulty addresses. Because the order varies a lot for different test algorithms and defect types, it is complicated to model and analyze the relationship between the realistic defects and the address detecting order for different tests. For example, when running March test algorithm, there will be one error bitmap for each read operation. Some neighbor faulty cells may be detected in different reads. In that case, the faulty cells will appear in different error bitmaps [35], and the resultant faulty addresses will not be in sorted order. In addition, the detecting order of the faulty addresses may be back and forth for other non-March test. In BRAVES, the simulation engine analyzes the faulty addresses

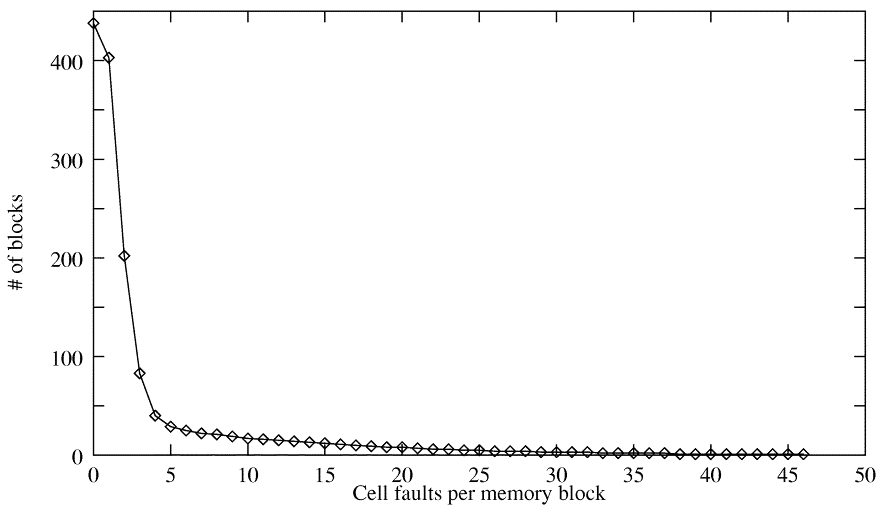

Fig. 12. Size distribution for cell faults.

in linear or sorted detecting order for simplicity. To statistically minimize the sensitivity to the address order, the simulator analyzes a large amount of memory samples with randomly generated errors. For the evaluation of the BIRA for the existing production line, however, BRAVES can simulate the repair rate based on the given test algorithm and the faulty addresses in realistic order for any test algorithm.

In our analysis, we simulated a total of 1,552 memory blocks with a block size of 1024×64 bits. A different number of cell faults and line faults are injected into each memory block at random locations. Fig. 12 shows the size distribution of cell faults using a mixed Poisson and exponential distribution model. The size distribution for line faults looks similar, except the probabilities are lower. Also, r ranges from 6 to 10 and c from 2 to 6. The size of the local bitmap is assumed to be 8×4 for the LO algorithm and $r \times c$ for the LRM, instead of 24×16 for the full size bitmap without the must-repair phase. From the report of the BRAINS, the LO and ESP algorithms are satisfactory in most cases, and perform very close to the optimal algorithm in terms of repair rate. A high repair rate implies a high yield after repair, if the area overhead is the same. The simulation results of algorithms using the local bitmap approach are satisfactory due to the assumption of knowing the physical layout (cell location) of the memory blocks (which is reasonable for BIRA applications), and test algorithms are performed in the physical address order instead of the logical address order. While the order of faulty cell detection affects the repair rate, common test algorithms are applied in a linear address order, which facilitates the line covering process and achieves higher repair rate. However, the repair rate decreases slightly when faults are detected among multiple test phases, leading to the faulty addresses being discovered out of order. This is a limitation of analysis algorithms using the local bitmap (or even without local bitmap) to achieve built-in implementation.

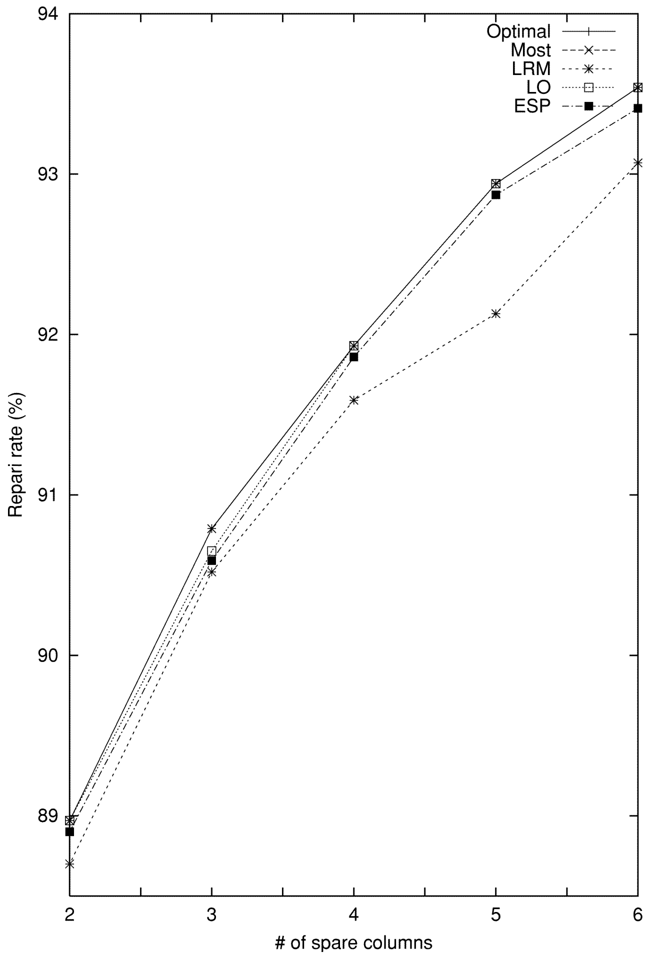

Fig. 13. Simulation result for $r = 10$.

Fig. 13 shows one slice of the whole simulation result, i.e., when $r = 10$ and c ranges from 2 to 6. The optimal and repair-most algorithms are simulated for reference. The solid line depicts the optimal result, which is the upper bound of the repair rate. The repair rate of the LO algorithm is close to the optimal solution in the figure because the heuristics for orthogonal faults are used. The ESP algorithm is better than the LRM algorithm (and is also close to optimal) if most of the faults are independent cell faults as is assumed in this case. The orthogonal-fault heuristics can also improve the repair rate of the LRM algorithm. The relative efficiency of these algorithms will vary in different memory configurations and spare architectures.

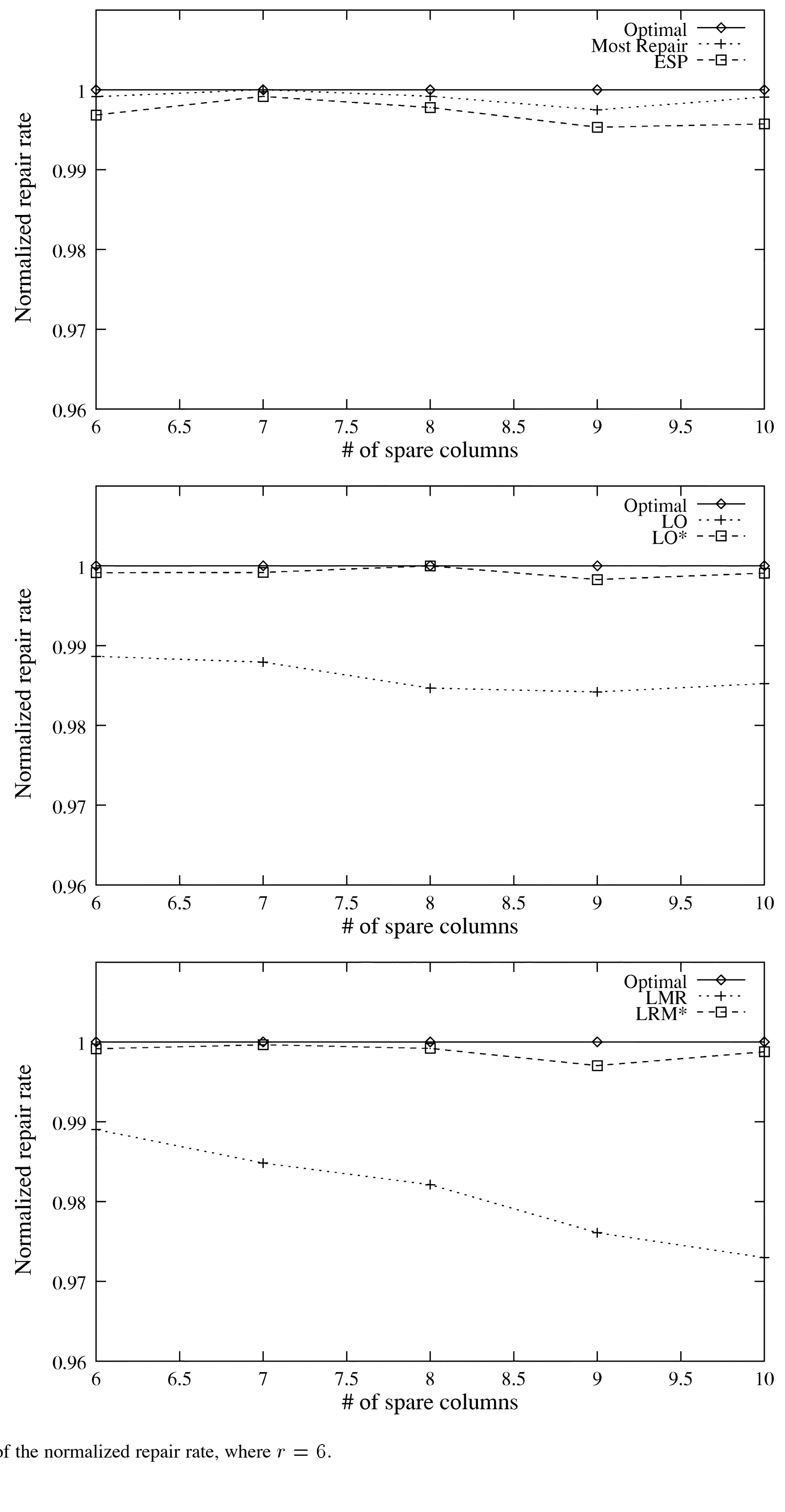

Fig. 14. Simulation result of the normalized repair rate, where $r = 6$.

A simulation of the normalized repair rate (with respect to the optimal algorithm) with the same memory and spare architecture is shown in Fig. 14. Normalization is done with respect to the repair rate of the optimal algorithm. More faults are injected into this simulation to enlarge the difference among the algorithms. In the figure, the line LO^* represents the LO algorithm with the orthogonal-fault heuristics. Similarly, the line LRM^* is the LRM algorithm using an $(r+c) \times (r+c)$ bitmap (while the original LRM algorithm uses an $r \times c$ bitmap). On average, the LO^* algorithm has the best repair rate (close to op-

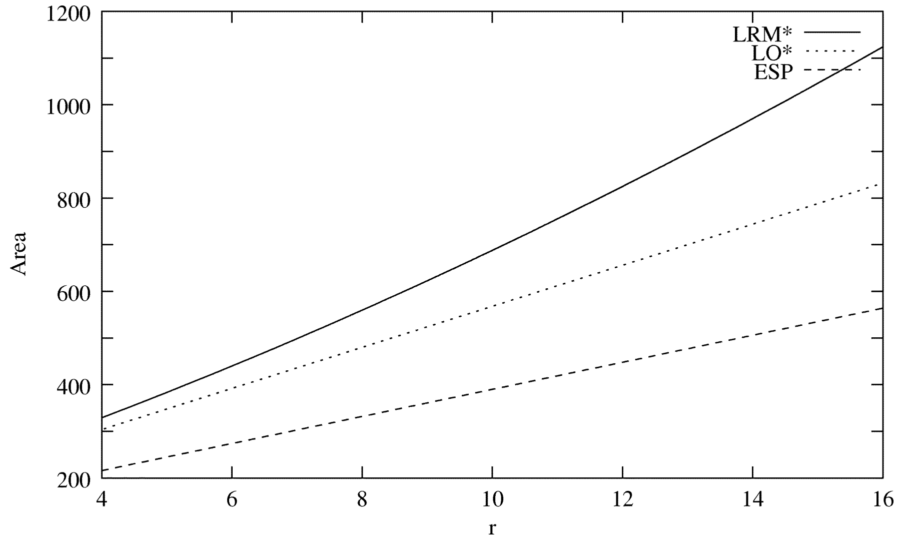

Fig. 15. Comparison of area overhead, where $M = 1024, N = 64, 4 \leq r \leq 16$, and $c = 4$.

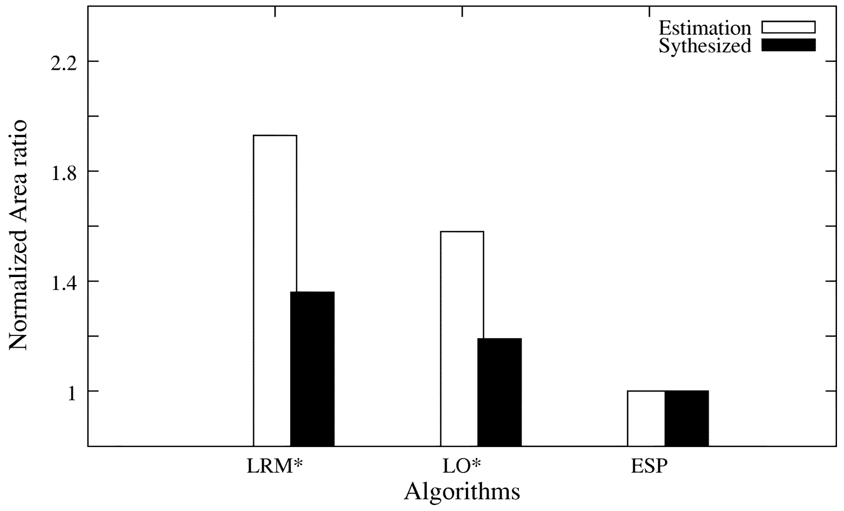

Fig. 16. Estimated and real area overhead, normalized with respect to the best result in each case, where $M = 1024, N = 64, m = r = 8$, and $n = c = 4$.

timal). The ESP algorithm also performs efficiently, especially for imbalanced spare architectures, such as when the number of spare rows is much greater than the number of spare columns. When the probability of cell fault is low and the probability of line fault is also low (as in a mature fabrication process), the ESP algorithm is a good approach due its simple control, low area and time overhead, as well as the efficient repair rate (see Figs. 14–16). The repair rate of the LRM algorithm is highly dependent on its bitmap size. This can be seen from the difference between the LRM and LRM* lines in Fig. 14. The increase in the bitmap size results in an improvement on its repair rate, as shown in the figure. There is a trade-off between its repair rate and the (time and area) overhead.

Though increasing the bitmap size improves the repair rate for LRM and LO, the complexity of exhaustive search grows faster for LO. In practice, the difference in repair rate between these two algorithms is small when the fault size distribution is a mixed Poisson distribution, i.e., the probability that a memory block contains a lot of faulty cells is small. This is normally true for a mature fabrication process. Therefore, among these algorithms, the area and time overhead will become the major concern when we select the algorithm for BIRA. Practically, the repair rates of different algorithms can be compared by BRAVES when using the distribution from the real process.

We now discuss hardware (area) overhead. The storage requirement is calculated for estimating the hardware overhead because all the proposed algorithms need storage cells with matching capability, and the storage cells dominate the silicon area of the BIRA circuit. For the two local-bitmap based algorithms (i.e., LRM* and LO*), an $m \times n$ array is required. A row fault counter and a column fault counter require $\lceil \log(n+1) \rceil$ bits and $\lceil \log(m+1) \rceil$ bits, respectively, for the LRM* algorithm. A row address tag and a column address tag need $\lceil \log M \rceil + 1$ bits and $\lceil \log N \rceil + 1$ bits, respectively. The orthogonal-fault heuristics require $(r+c) \cdot (\lceil \log M \rceil + \lceil \log N \rceil + 1)$

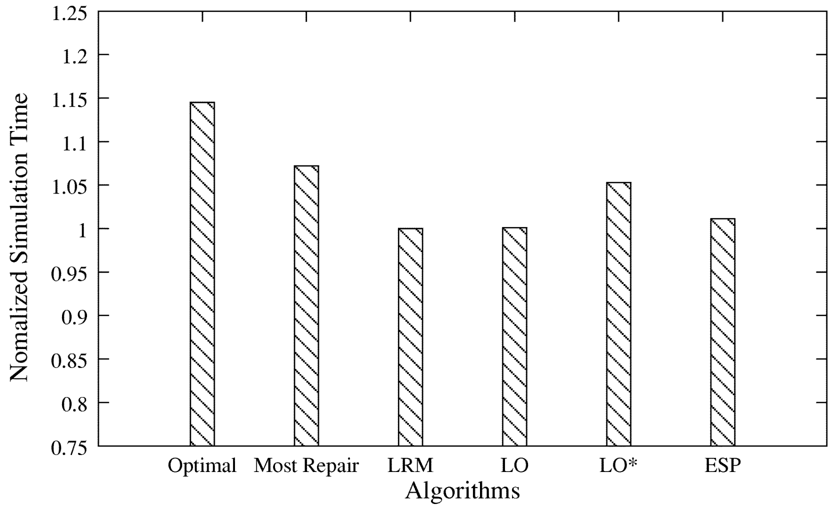

Fig. 17. Simulation time for the BIRA algorithms.

storage cells. Therefore, the area overhead of the three proposed redundancy analysis algorithms are estimated as follows:

$$\begin{aligned} A_{LRM*} = & m \cdot n \\ & + [(\lceil \log M \rceil + 1) \\ & \quad + (\lceil \log(n+1) \rceil)] \cdot m \\ & + [(\lceil \log N \rceil + 1) \\ & \quad + (\lceil \log(m+1) \rceil)] \cdot n, \end{aligned} \tag{9}$$

$$\begin{aligned} A_{LO*} = & m \cdot n \\ & + (\lceil \log M \rceil + 1) \cdot m \\ & + (\lceil \log N \rceil + 1) \cdot n \\ & + (r+c) \cdot (\lceil \log M \rceil + \lceil \log N \rceil + 1), \end{aligned} \tag{10}$$

$$\begin{aligned} A_{ESP} = & (r+c) \\ & \cdot [(\lceil \log M \rceil + 1) + (\lceil \log N \rceil + 1)], \end{aligned} \tag{11}$$

$$\begin{aligned} A_{spare\_register} = & (\lceil \log M \rceil + 1) \\ & \cdot r + (\lceil \log N \rceil + 1) \cdot c, \end{aligned} \tag{12}$$

where $A_{spare\_register}$ denotes the area of the spare row and column registers, which are required for all three algorithms. Fig. 15 compares the area overhead of the estimated storage elements, where we assume that the memory size is 1024 × 64 bits, r ranges from 4 to 16, and c is 4. For LO*, we assign $m = r$ and $n = c$, while $m = n = r + c$ for LRM* as in Fig. 14. From the equations and the figure, we see that LRM* has a higher area complexity than LO* with respect to the bitmap size. Also, ESP has the least storage requirement because of its simple control and the nature of processing orthogonal faults. For memories with a complicated spare architecture, a larger bitmap will be needed to retain the repair rate if we use LRM* or LO*. In that case, the storage cells and routing area may dominate the area overhead.

The proposed BIRA circuits have been integrated with the BIST design which we have developed [6] using a 0.25 μm single-poly-5-metal CMOS process. The overhead considering the control circuitry in different algorithms is shown in Fig. 16. Note that in (9)–(12) and Fig. 15 we consider only the storage cells. The result is the estimated area overhead. We normalize the result with respect to the best method (i.e., ESP) in the figure, which compares with the normalized area overhead (with respect to the best method—ESP, again) of the synthesized designs.

The estimated area overhead reflects the real synthesis data. Furthermore, the overhead of the control logic is also an important factor. From the real synthesis data, ESP results in the smallest area overhead. The area overhead of LO* is lower than that of LRM*. In addition, the orthogonal-fault heuristics of LO* is very efficient (see Fig. 14) with low area overhead. The LRM* has better repair rate than LRM. However, the improvement comes from the larger local bitmap. Nevertheless, LO* is not as good for high-redundancy memories because of its exponential time complexity of spare allocation. To reduce the overall area overhead, the BIRA circuit can be shared with multiple memory blocks, while the test/analysis time can be reduced by applying the BIRA to several memory blocks in parallel.

According to the synthesis report, the area overhead of the proposed BIRA schemes is approximately from 0.52% to 1.17% as compared with a 16 Mbit embedded DRAM core of 0.17 μm process. Assume that $M = 1024$, $N = 64$, $r = 8$, and $c = 4$; $m = r$, $n = c$ for LO* and $m = n = r + c$ for LRM*. For a 64 Mbit DRAM core, the area overhead is below 0.3% for any of the proposed BIRA algorithms. It has a high repair rate as shown in Fig. 14. Multiple copies of the BIRA circuit can be implemented if the overhead is affordable. The parallelism will reduce the computation time of the redundancy analysis effectively. Otherwise, the shared BIRA can be used for area-restricted designs.

For time complexity reference, Fig. 17 shows the software simulation time for different BIRA algorithms. Note that the hardware execution time of these algorithms may have a different relationship. In general, the optimal solution costs the

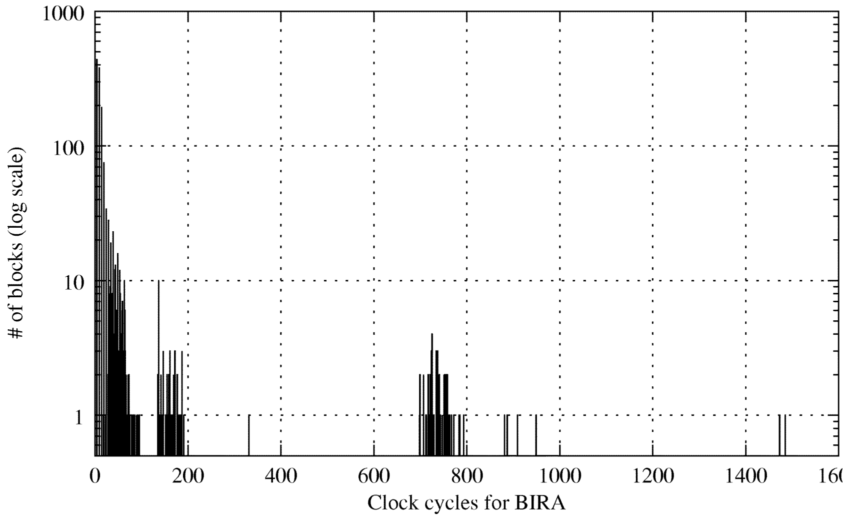

Fig. 18. Time distribution of the ESP execution cycles.

most, and the repair-most algorithm second. The LO* algorithm which dealing with the orthogonal-fault heuristics is the most time-consuming one among the three proposed algorithms. Fig. 17 shows that LRM, LO, and ESP have about the same computation speed. In this analysis, the difference between LRM and LO is very small (0.1% in normalized simulation time, i.e., about 32 seconds), when the referenced simulation time is 4040.53 seconds, or about 67 minutes. Because the absolute simulation time reflects the software complexity of these redundancy analysis algorithms, the complexity of our approach is affordable for a large amount of analysis. For a high-redundancy memory, the execution time of LO will be longer than LRM, and ESP will be the fastest. In addition, BRAVES can be used to model and estimate the execution cycles of the BIRA hardware. Fig. 18 shows the execution cycle distribution of ESP for the analysis of 1,552 memory samples, including unrepairable ones. Most of the cases require less than 60 clock cycles (ranging from 4 to 1485 cycles, and 50.33 cycles on average) because of the size distribution of the injected faults. In addition, the early termination can speed up the analysis for the unrepairable memories with many faults. As compared with the IFA9N March test, the maximum execution cycles of ESP circuitry only add a negligible 0.25% overhead to the test time of 589 824 ($9 \times 1024 \times 64$) read/write operations, assuming that each read or write takes only one clock cycle. For a 16 Mbit embedded DRAM core with 256 blocks and the frequency of 100 MHz, the BIRA will take 3.8 ms approximately, assuming all the memory samples require the analysis with the maximum cycles.

In summary, the LO* algorithm with the orthogonal-fault heuristics has the best repair rate for different memory and spare architectures according to our simulation results. It however has to perform exhaustive search of the local bitmap, so can be slow when a large bitmap is required. Nevertheless, for an imbalanced redundancy architecture LO is a good choice, because the exhaustive search is done along the smaller dimension of the bitmap. The search time can be greatly reduced with the same repair rate. The LRM approach is more scalable because expanding the bitmap does not increase the time of spare allocation in an exponential rate. The ESP algorithm has the least area overhead among the three in general. It also achieves a good repair rate, especially for imbalanced spare architectures. For a mixed Poisson fault-size distribution as assumed in this paper, ESP has the best repair efficiency, because of its high repair rate and low area/time overhead.

REFERENCES

[1] M. Lin, Ed., "1997 Semiconductor Industry Annual Report," (in Chinese), Industrial Technology Research Institute (ITRI), Hsinchu, Taiwan, ITIS project report, 1997.

[2] G. E. Moore, "Progress in digital integrated electronics," in *Proc. IEEE IEDM*, 1975, pp. 11–13.

[3] R. P. Cenker, D. G. Clemons, W. P. Huber, J. P. Petrizzi, F. J. Procyk, and G. M. Trout, "A fault-tolerant 64 k dynamic random-access memory," *IEEE Trans. Electron Devices*, vol. 26, pp. 853–860, June 1979.

[4] R. T. Smith, J. D. Chipala, J. F. M. Bindels, R. G. Nelson, F. H. Fischer, and T. F. Mantz, "Laser programmable redundancy and yield improvement in a 64 k DRAM," *IEEE J. Solid-State Circuits*, vol. 16, pp. 506–514, Oct. 1981.

[5] C. A. Benevit, J. M. Cassard, K. J. Dimmler, A. C. Dumbri, M. G. Mound, F. J. Procyk, W. Rosenzweig, and A. W. Yanof, "A 256 k dynamic random-access memory," *IEEE J. Solid-State Circuits*, vol. 17, pp. 857–862, Oct. 1982.

[6] C.-T. Huang, J.-R. Huang, C.-F. Wu, C.-W. Wu, and T.-Y. Chang, "A programmable BIST core for embedded DRAM," *IEEE Design & Test of Computers*, vol. 16, pp. 59–70, Jan.–Mar. 1999.

[7] "Semiconductor industry association," in *International Technology Roadmap for Semiconductors (ITRS)*, Dec. 2000.

[8] K. K. Saluja, S. H. Sng, and K. Kinoshita, "Built-in self-testing RAM: A practical alternative," *IEEE Design & Test of Computers*, vol. 4, pp. 42–51, Feb. 1987.

[9] M. Franklin and K. K. Saluja, "Built-in self-testing of random-access memories," *IEEE Computer*, pp. 45–56, Oct. 1990.

[10] B. Nadeau-Dostie, A. Silburt, and V. K. Agarwal, "Serial interface for embedded-memory testing," *IEEE Design & Test of Computers*, vol. 7, pp. 52–63, Apr. 1990.

[11] P. Camurati, P. Prinetto, M. S. Reorda, S. Barbagallo, A. Burri, and D. Medina, "Industrial BIST of embedded RAMs," *IEEE Design & Test of Computers*, vol. 12, pp. 86–95, 1995.

[12] S. Tanoi, Y. Tokunaga, T. Tanabe, K. Takahashi, A. Okada, M. Itoh, Y. Nagatomo, Y. Ohtsuki, and M. Uesugi, "On-wafer BIST of a 200-Gb/s failed-bit search for 1-Gb DRAM," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1735–1742, Nov. 1997.

[13] J. Dreibelbis, J. Barth, H. Kalter, and R. Kho, "Processor-based built-in self-test for embedded DRAM," *IEEE J. Solid-State Circuits*, pp. 1731–1740, Nov. 1998.
[14] R. P. Treuer and V. K. Agarwal, "Built-in self-diagnosis for repairable embedded RAMs," *IEEE Design & Test of Computers*, vol. 10, pp. 24–33, June 1993.
[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] P. Mazumder and J. S. Yih, "A novel built-in self-repair approach to VLSI memory yield enhancement," in *Proc. Int. Test Conf. (ITC)*, 1990, pp. 833–841.
[17] A. Tanabe, T. Takeshima, H. Koike, Y. Aimoto, M. Takada, T. Ishijima, N. Kasai, H. Hada, K. Shibahara, T. Kunio, T. Tanigawa, T. Saeki, M. Sakao, H. Miyamoto, H. Nozue, S. Ohya, T. Murotani, K. Koyama, and T. Okuda, "A 30-ns 64-Mb DRAM with built-in self-test and self-repair function," *IEEE J. Solid-State Circuits*, vol. 27, pp. 1525–1533, Nov. 1992.
[18] T. Chen and G. Sunada, "Design of a self-testing and self-repairing structure for highly hierarchical ultra-large capacity memory chips," *IEEE Trans. VLSI Systems*, vol. 1, pp. 88–97, June 1993.
[19] P. Mazumder and Y.-S. Jih, "A new built-in self-repair approach to VLSI memory yield enhancement by using neural-type circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, pp. 124–136, Jan. 1993.
[20] I. Kim, Y. Zorian, G. Komoriya, H. Pham, F. P. Higgins, and J. L. Lweandowski, "Built in self repair for embedded high density SRAM," in *Proc. Int. Test Conf. (ITC)*, Oct. 1998, pp. 1112–1119.
[21] D. K. Bhavsar, "An algorithm for row-column self-repair of RAMs and its implementation in the alpha 21 264," in *Proc. Int. Test Conf. (ITC)*, 1999, pp. 311–318.
[22] N. Park and E. Lombardi, "Repair of memory arrays by cutting," in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, San Jose, Aug. 1998, pp. 124–130.
[23] S.-K. Lu and C.-H. Hsu, "Built-in self-repair for divided word line memory," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 2001, pp. 13–16.
[24] T. Kawagoe, J. Ohtani, M. Niiro, T. Ooishi, M. Hamada, and H. Hidaka, "A built-in self-repair analyzer (CRESTA) for embedded DRAMs," in *Proc. Int. Test Conf. (ITC)*, 2000, pp. 567–574.
[25] J. Ohtani, T. Ooishi, T. Kawagoe, M. Niiro, M. Maruta, and H. Hidaka, "A shared built-in self-repair analysis for multiple embedded memories," in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, vol. 4, 2001, pp. 187–190.
[26] W.-K. Huang, Y.-N. Shen, and F. Lombardi, "New approaches for the repair of memories with redundancy by row/column deletion for yield enhancement," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, pp. 323–328, Mar. 1990.
[27] M. Tarr, D. Boudreau, and R. Murphy, "Defect analysis system speeds test and repair of redundant memories," *Electronics*, pp. 175–179, Jan. 1984.
[28] J. R. Day, "A fault-driven, comprehensive redundancy algorithm," *IEEE Design & Test of Computers*, vol. 2, pp. 35–44, June 1985.
[29] S.-Y. Kuo and W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays," *IEEE Design & Test of Computers*, vol. 4, pp. 24–31, Feb. 1987.
[30] C.-L. Wey and F. Lombardi, "On the repair of redundant RAMs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, pp. 222–231, Mar. 1987.
[31] R. W. Haddad, A. T. Dahbura, and A. B. Sharma, "Increased throughput for the testing and repair of RAMs with redundancy," *IEEE Trans. Computers*, vol. 40, pp. 154–166, Feb. 1991.
[32] C. H. Stapper, A. N. McLaren, and M. Dreckmann, "Yield model for productivity optimization of VLSI memory chips with redundancy and partially good product," *IBM J. Research and Development*, vol. 24, pp. 398–409, May 1980.
[33] M. Choi and N. Park, "Dynamic yield analysis and enhancement of FPGA reconfigurable memory system," in *Proc. 18th IEEE Instrumentation and Measurement Technology Conf. (IMTC)*, vol. 1, Budapest, Hungary, May 2001, pp. 386–391.
[34] M. Choi, N. Park, F. Meyer, F. Lombardi, and V. Piuri, "Reliability measurement of fault-tolerant onboard memory system under fault clustering," in *Proc. 19th IEEE Instrumentation and Measurement Technology Conf. (IMTC)*, vol. 2, Anchorage, AK, May 2002, pp. 1161–1166.
[35] 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.

Chih-Tsun Huang received the B.S., M.S., and Ph.D. degrees from the Department of Electrical Engineering, NTHU, in 1994, 1996 and 2000, respectively. He is currently a post-doctoral researcher and an Adjunct Assistant Professor in the same department. He received the Best Paper Award of the 2003 IEEE Asia & South Pacific Design Automation Conference (ASP-DAC), and the Special Feature Award of the 2003 ASP-DAC University LSI Design Contest. His research areas include VLSI testing, high-performance embedded core and system design, design for testability and reliability, and embedded memory testing. Dr. Huang is a member of the IEEE.

Chi-Feng Wu received the B.S. degree in 1996, M.S. degree in 1998, and Ph.D. degree in 2001, all in Electrical Engineering, from National Tsing Hua University (NTHU), Hsinchu, Taiwan. His research interests are in the design and test of VLSI cores and systems. He is currently a CAD engineer at Realtek Semiconductor Corp., Hsinchu, Taiwan, investigating reusable IP design, system-on-chip integration and testing.

Jin-Fu Li received the B.S. degree in 1995 from the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, and the M.S. and Ph.D. degrees, both in the Department of Electrical Engineering, in 1999 and 2002, respectively, from National Tsing Hua University, Hsinchu, Taiwan. Since 2002 he has been with the Department of Electrical Engineering, National Central University, Chung-Li, Taiwan, where he is currently an Assistant Professor. His research interests include advanced VLSI/SOC design and testing, memory testing, memory diagnosis, memory self-repair, system-on-chip (SOC) testing, SOC test planning, and SOC test optimization.

Cheng-Wen Wu received the B.S.E.E. degree in 1981 from National Taiwan University, Taipei, Taiwan, and the M.S. and Ph.D. degrees in Electrical and Computer Engineering, in 1985 and 1987, respectively, from the University of California, Santa Barbara. From 1981 to 1983, he was an Ensign Instructor at the Chinese Naval Petty Officers' School of Communications and Electronics, Tsoying, Taiwan. From 1983 to 1984, he was with the Information Processing Center of the Bureau of Environmental Protection, Executive Yuan, Taipei, Taiwan. From 1985 to 1987 he was a post graduate researcher at the Center for Computational Sciences and Engineering at UCSB. Since 1988 he has been with the Department of Electrical Engineering, National Tsing Hua University (NTHU), Hsinchu, Taiwan, where he is currently a Professor. He also has served as the Director of the University's Computer and Communications Center from 1996 to 1998, and the Director of the University's Technology Service Center from 1998 to 1999. From August 1999 to February 2000, he was a visiting faculty of the ECE Department, UCSB. He then served as the Chair of the EE Department of NTHU from 2000 to 2003. He has been the Director of the IC Design Technology Center of the university since 2000.

Dr. Wu was the Technical Program Chair of the IEEE Fifth Asian Test Symposium (ATS'96), and the General Chair of ATS'00. He is the Editor-in-Chief for the Journal of the Chinese Institute of Electrical Engineers (JCIEE). He was an Editor for JCIEE from 2000 to 2003, and in 2001 he edited the JCIEE Special Issue on Design and Test of System-on-Chip. He also was a Guest Editor of the Journal of Information Science and Engineering (JISE), Special Issue on VLSI Testing. He received the Distinguished Teaching Award from NTHU in 1996, the Outstanding Electrical Engineering Professor Award from the Chinese Institute of Electrical Engineers (CIEE) in 1997, the Distinguished Research Awards from National Science Council in 2000 and 2002, the Industrial Collaboration Award from the Ministry of Education in 2001, the Best Paper Award of the 2002 IEEE International Workshop on Design & Diagnostics of Electronic Circuits & Systems, the Best Paper Award of the 2003 IEEE Asia & South Pacific Design Automation Conference (ASP-DAC), and the Special Feature Award of the 2003 ASP-DAC University LSI Design Contest. He is interested in design and test of high performance VLSI circuits and systems. Dr. Wu is a Life Member of the CIEE and a Senior Member of the IEEE.