Efficient Built-In Redundancy Analysis for Embedded Memories With 2-D Redundancy

Shyue-Kung Lu, Yu-Chen Tsai, Chih-Hsien Hsu, Kuo-Hua Wang, and Cheng-Wen Wu

***Abstract*—A novel redundant mechanism is proposed for embedded memories in this paper. Redundant rows and columns are added into the memory array as in the conventional approaches. However, the redundant rows and columns are divided into row blocks and column blocks, respectively. The reconfiguration is performed at the row (column) block level instead of the conventional row (column) level. Based on the proposed redundant mechanism, we first show that the complexity of the redundancy allocation problem is NP-complete. Thereafter, an extended local repair-most (ELRM) algorithm suitable for built-in implementation is proposed. The complexity of the ELRM algorithm is $O(N)$, where N denotes the number of memory cells. According to the simulation results, the hardware overhead for implementing this algorithm is below 0.17% for a 1024 $\times$ 2048-b SRAM. Due to the efficient usage of the redundant elements, the manufacturing yield, repair rate, and reliability can be improved significantly.**

***Index Terms*—Embedded memory, redundancy analysis, reliability, repair rate, yield.**

I. Introduction

According to the Semiconductor Industry Association (SIA) and ITRS 2001, the relative silicon area occupied by embedded memories will approach 94% by 2014 [1]. For example, the Compaq Alpha EV7 chip employs 135 million transistors for RAM cores alone, while the entire chip has 152 million transistors. Since embedded memories have higher complexity and higher density than other logic blocks, they have higher failure possibility. Therefore, an appropriate fault-tolerant and reliable design technique should be incorporated into the chip at the design stage.

In general, the reliability and fabrication yield of embedded memories can be improved by the incorporation of some form of redundancy. Conventional ways to add redundancy into an embedded memory array include the following.

- **Redundant rows or redundant columns** [2], [3]: By using this approach, redundant rows or columns are added into the memory array. One of the redundant rows/columns is used to replace the faulty row/column. The main advantage of this one-dimensional (1-D) approach is that it can be implemented easily. The repair efficiency of this approach can be low since a faulty row (column) cannot be replaced by the redundant columns (rows).
- **Redundant rows and redundant columns** [4]–[7]: By using this approach, both redundant rows and columns are incorporated into the memory array. When a faulty cell is detected, we can use a redundant row or a redundant column to replace it. It is more efficient than the first approach when multiple faulty cells exist in the memory array. The main drawback of this approach is that the optimal redundancy allocation problem becomes NP-complete [8], [26]. Although many heuristic algorithms have been proposed to solve this problem, it is still difficult to develop on-chip implementations for these algorithms. Due to the high bandwidth requirement of today's system-on-chip (SOC) designs, we usually have long bit lines and word lines for embedded memories. The repair efficiency will decrease since an entire (and long) redundant row (column) is required to repair a faulty row (column) containing only a small number of faulty cells.

In order to improve the efficiency of repairing embedded memories, a novel redundancy mechanism is proposed. Redundant rows and redundant columns are added into the memory array as in the conventional approaches. However, all of the memory rows/columns (including the redundant rows/columns) are divided into *row/column blocks* based on the divided word-line (DWL) [9] and divided bit-line (DBL) [10] approaches, respectively. Moreover, reconfiguration is performed at the row/column block level instead of the traditional row/column level. It can be found in our previous work [11] that this redundancy structure will result in higher repair rates, and the manufacturing yield will be improved significantly.

Traditionally, redundancy analysis is always performed externally by the host computer of automatic test equipment (ATE). However, embedded memories are difficult to access externally using an ATE. The ATE approach is also not suitable for field repair and enhancement of postmanufacturing reliability. Therefore, for test and repair of embedded memories, built-in self-test (BIST) and built-in self-repair (BISR) [11]–[16], [27], [31] techniques are receiving growing attention. Similarly, built-in redundancy analysis (BIRA) [17] is also very important in SOC applications.

In this paper, based on the proposed redundant mechanism, we first show that the complexity of optimal redundancy allocation based on the proposed redundancy mechanism is NP-complete. Thereafter, an extended local repair-most (ELRM) algorithm suitable for built-in implementation is proposed. According to the simulation results, the hardware

Manuscript received April 12, 2005; revised July 16, 2005.

S.-K. Lu is with the Very Large Scale Integration-/Computer-Aided Design Laboratory, Department of Electronic Engineering, Fu-Jen Catholic University, Taipei 24205, Taiwan, R.O.C. (e-mail: sklu@ee.fju.edu.tw).

Y.-C. Tsai is with Socle Technology Corporation, Hsinchu, Taiwan, R.O.C.

K.-H. Wang is with the Department of Computer Science and Information Engineering, Fu-Jen Catholic University, Taipei 24205, Taiwan, R.O.C.

C.-H. Hsu and C.-W. Wu are with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan 300, R.O.C.

Digital Object Identifier 10.1109/TVLSI.2005.863189

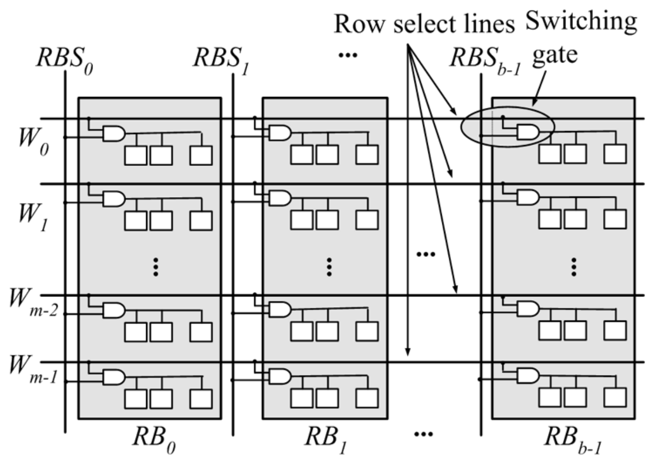

Fig. 1. DWL architecture.

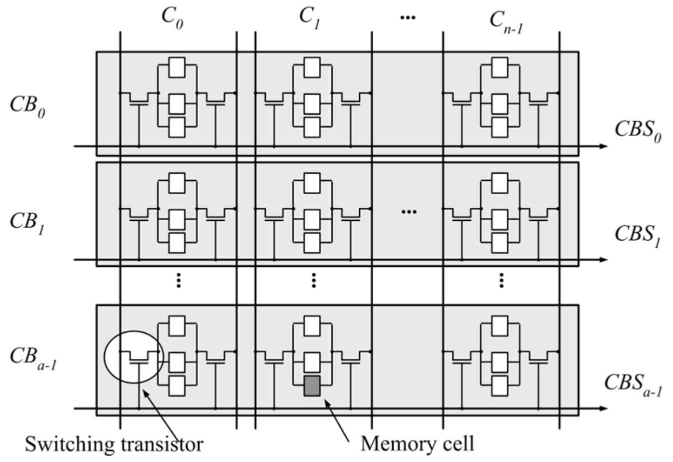

Fig. 2. DBL architecture.

overhead for implementing this algorithm is below 0.17% for a 1024×2048-b SRAM. Moreover, due to the efficient usage of the redundant elements, the manufacturing yield, repair rate, and reliability can be improved significantly.

The organization of this paper is as follows. Section II reviews the DWL and DBL structures. Section III models the spare allocation problem and shows that the complexity of this problem is NP-complete. A heuristic built-in redundancy analysis algorithm named the ELRM approach is proposed in Section IV. Section V gives an example to illustrate the proposed BIRA algorithm. Repair efficiency and hardware overhead are analyzed in Section VI. Section VII analyzes the yield improvement. Finally, some conclusions are given in Section VIII.

II. DWL and DBL Structures

The concepts of DWL and DBL techniques for RAMs were proposed in 1983 [9] and 1998 [10], respectively. Due to the power-down techniques for unused memory cells, their main advantages include lower power consumption and faster access time. The structures of DWL and DBL are shown in Figs. 1 and 2, respectively.

From Fig. 1, the main scenario of DWL is that each row of the memory cell array is divided into b row blocks by the word-line segments. If the memory has m rows $(W_0 - W_{m-1})$ and n columns $(C_0 - C_{n-1}), n/b$ columns are included in each *row bank* $(\mathrm{RB}_i, 0 \leq i \leq b-1)$. The DWL in each row block is activated by *switching gates*, which have two inputs—the *row select line* and the *row bank select line* $(\mathrm{RBS}_i, 0 \leq i \leq b-1)$. Similarly, the main scenario of DBL is that each column of the memory cell array is divided into a column blocks by the bit-line segments. If the memory has m rows, m/a rows are included in each *column bank* $(\mathrm{CB}_i, 0 \leq i \leq a-1)$. The subbit-lines in each column block are activated by the *switching transistors.* Furthermore, these switching transistors are controlled by the *column bank select lines* $(\mathrm{CBS}_i, 0 \leq i \leq a-1)$. Thus, only the memory cells connected to a subbit-line within a selected block are accessed in a given memory cycle.

Although DWL and DBL techniques possess several inherent advantages, the divided structures have not been used for fault-tolerant applications, except in our previous works [18], [19], [28], [29], [30]. In these works, only redundant rows or columns are added into the memory array. Therefore, the redundancy analysis approach is straightforward, and a content-addressable memory (CAM) can be used to store the bank and row (column) addresses of a faulty row (column) block. Reconfiguration is performed through the matching operations of the CAM block [18], [19].

III. Redundancy Allocation Problem for DWL and DBL Architectures

In [8] and [26], it is shown that the complexity of optimal redundancy allocation is NP-complete if redundant rows and columns are added into the memory array. The basic assumption is that an entire row or column is used as the basic replacement element. For the scenario where both the redundant rows and redundant columns are divided into row (column) blocks, redundancy analysis will be quite different. Therefore, the complexity of redundancy analysis for such scenario should also be derived. A memory array with two spare rows $(\mathrm{SR}_0, \mathrm{SR}_1)$ and two spare columns $(\mathrm{SC}_0, \mathrm{SC}_1)$ is divided into two row banks $(\mathrm{RB}_0, \mathrm{RB}_1)$ and two column banks $(\mathrm{CB}_0, \mathrm{CB}_1)$, as shown in Fig. 3. Inside the highlighted area in this figure, the intersection of a row bank and a column bank is called a divided array (DA). Therefore, the memory array is divided into $a \times b$ divided arrays $(\mathrm{DA}_{ij}, 0 \leq i \leq a-1, 0 \leq j \leq b-1)$.

For example, the memory array in Fig. 3 contains four DAs. The simplified version of Fig. 3 is shown in Fig. 4. A detailed DA is shown in Fig. 5. DAs are useful for our ELRM algorithm, which will be discussed later. If a row (column) block contains faulty cells, it is called a *faulty row (column) block*. The replacement scenario is as follows. A faulty row (column) block can be repaired with a redundant row (column) block in the same row (column) bank. As shown in our previous work [11], an address remapping CAM can be used to reconfigure the memory array. The address remapping CAM stores the row (column) bank address and row (column) address of a faulty row (column) block. In normal memory access, if the accessed address is stored in the remapping CAM, the match signal will enable the redundant row (column) blocks and disable the original memory array.

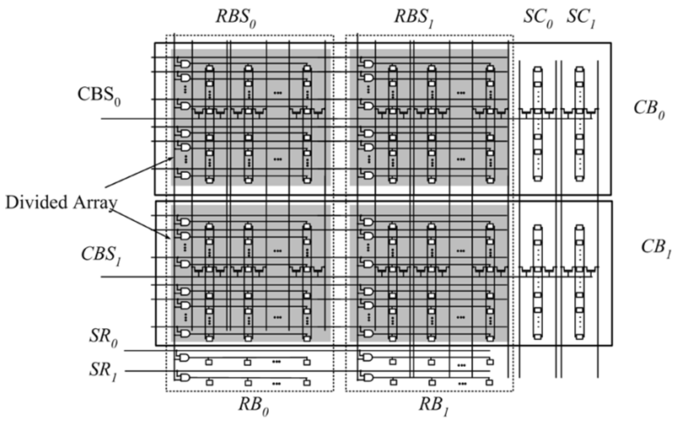

Fig. 3. Memory array with two spare rows and two spare columns, divided into two row banks and two column banks.

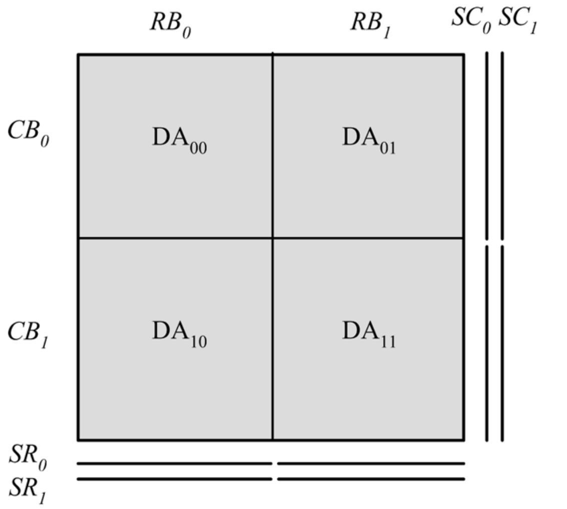

Fig. 4. Simplified version of Fig. 3.

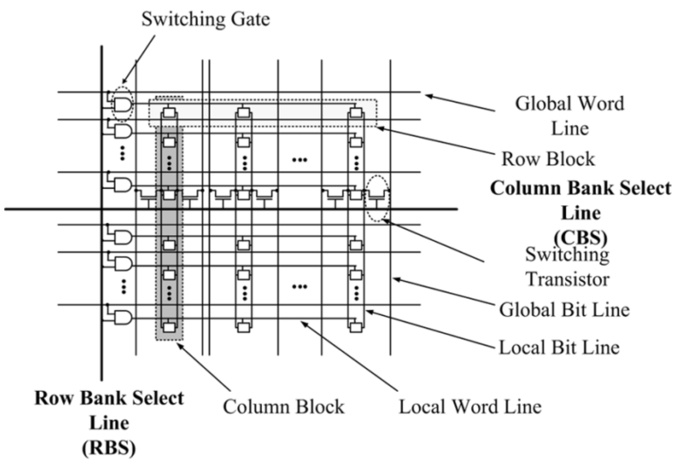

Fig. 5. Divided array (DA).

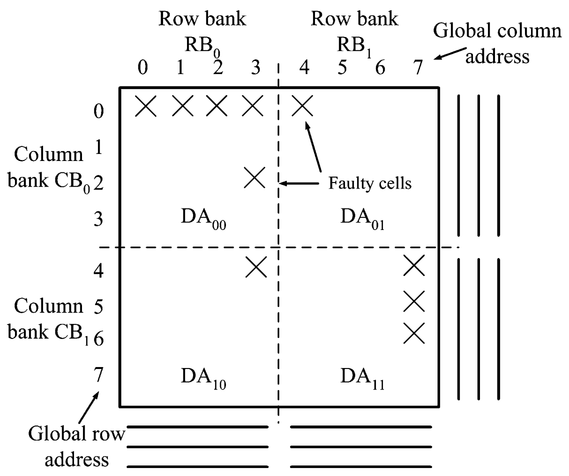

Fig. 6. 8 × 8 memory array, $a = 2, b = 2, \mathrm{SR} = 3$, and $\mathrm{SC} = 3$.

Since a faulty cell is within a faulty row block and a faulty column block, therefore, how to select a redundant row block or a redundant column block to replace it is still a problem. Therefore, we have to seek the optimal resource allocation to repair the memory array. It is evident that the problem to be solved is quite different from that in [8]. Our problem can be formalized as a *bipartite graph vertex covering problem* with constraints which will be described in the following.

A. Problem

For an $m \times n$ memory array divided into a column banks and b row banks, the numbers of spare rows and spare columns are denoted as SR and SC, respectively. An 8×8 array with $a = 2, b = 2, \mathrm{SR} = 3$, and $\mathrm{SC} = 3$ is shown in Fig. 6. The faulty cells are marked by $\times$'s. The coordinates of these faulty cells are (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (2, 3), (4, 3), (4, 7), (5, 7), and (6, 7), respectively. Our spare allocation problem is to find the minimum number of redundant row blocks and column blocks that cover all the faulty cells. The conjunctive form (CF) [8] of the allocation problem for Fig. 6 can be formalized as follows:

$$\begin{aligned}\mathrm{CF} = & (R_0 + C_0)(R_0 + C_1)(R_0 + C_2) \\ & \times (R_0 + C_3)(R_0 + C_4)(R_2 + C_3) \\ & \times (R_4 + C_3)(R_4 + C_7)(R_5 + C_7)(R_6 + C_7)\end{aligned}$$

where R_i denotes row i, C_j denotes column $j, 0 \leq i \leq m - 1, 0 \leq j \leq n - 1$, and R_i and C_j are Boolean variables. Therefore, they can take on the value of 0 or 1. For example, if a spare row is selected to cover R_0 ($R_0 = 1$), then the clauses $(R_0 + C_0), (R_0 + C_1), (R_0 + C_2), (R_0 + C_3)$, and $(R_0 + C_4)$ will have value 1. In other words, faulty cells (0, 0), (0, 1), (0, 2), (0, 3), and (0, 4) are repaired by using this spare row.

If the row (column) bank address is also taken into account, then the conjunctive form can be rewritten as

$$\begin{aligned}\mathrm{CF} = & (R_{00} + C_{00})(R_{00} + C_{10})(R_{00} + C_{20}) \\ & \times (R_{00} + C_{30})(R_{20} + C_{30})(R_{40} + C_{31})(R_{01} + C_{40}) \\ & \times (R_{41} + C_{71})(R_{51} + C_{71})(R_{61} + C_{71})\end{aligned}$$

where R_{ij} indicates the global row index i and the row bank index j, and C_{kl} indicates the global column index k and the column bank index l. R_{ij} and C_{kl} are both Boolean variables. If a redundant row block is chosen to repair R_{00} (i.e., assign 1 to R_{00}), then the clauses $(R_{00} + C_{00}), (R_{00} + C_{10}), (R_{00} + C_{20})$, and $(R_{00} + C_{30})$ will have the value 1. That is, faulty cells (0,

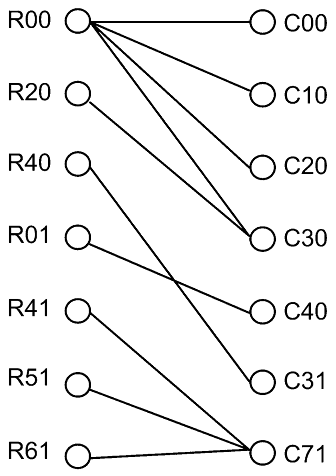

Fig. 7. Bipartite graph model of the allocation problem.

0), (0, 1), (0, 2) and (0, 3) will be repaired by using one of the redundant row blocks in RB_0.

B. Graph Model and Complexity Analysis

Definition: A graph $G(V, E)$ (or $G(V_1, V_2, E)$) is called a *bipartite graph* if its vertex set V is the disjoint union of set V_1 and V_2, and every edge in E has the form (v_1, v_2), where $v_1 \in V_1$ and $v_2 \in V_2$ [20].

Let the set of $R'_{ij}s$ and $C'_{kl}s$ described in the above problem be the vertex set V_1 and V_2, respectively. R_{ij} and C_{kl} are connected with an edge if there is a clause $(R_{ij}+C_{kl})$ in the CF. The corresponding bipartite graph of Fig. 6 is shown in Fig. 7. Each edge represents a faulty cell. Therefore, the spare allocation problem can be viewed as a bipartite vertex covering problem. In other words, we have to find the minimal set of R_{ij}'s and C_{kl}'s that covers all of the edges. For example, R_{00} is incident with four edges. Therefore, a row block in RB_0 can be used to repair these four faulty cells.

According to the problem specification, the memory array contains SR redundant rows and SC redundant columns. This means that the number of R_{ij}'s to be chosen cannot be greater than SR for each $j, 0 \leq j \leq b-1$. Similarly, the number of C_{kl}'s to be chosen cannot be greater than SC for each $l, 0 \leq l \leq a-1$. The allocation problem then can be view as a bipartite vertex covering problem with the constraints described above.

The formal manner to prove that the complexity of a problem is NP-complete contains the following two steps [20].

Step 1) Show that the problem is in the class of NP problems.

Step 2) Find a known problem in the NP class that can be transformed to our problem in polynomial time.

For our allocation problem, we do not need to use the above two steps. According to [20], we know that if a special instance of a problem is NP-complete, then the problem is also NP-complete. For our problem, we consider the special case that all the faulty memory cells are contained in a single divided array. An example is shown in Fig. 8, where all faulty cells are contained in DA_{00}. In this case, finding an optimal redundant row (column) block allocation is equivalent to finding an optimal

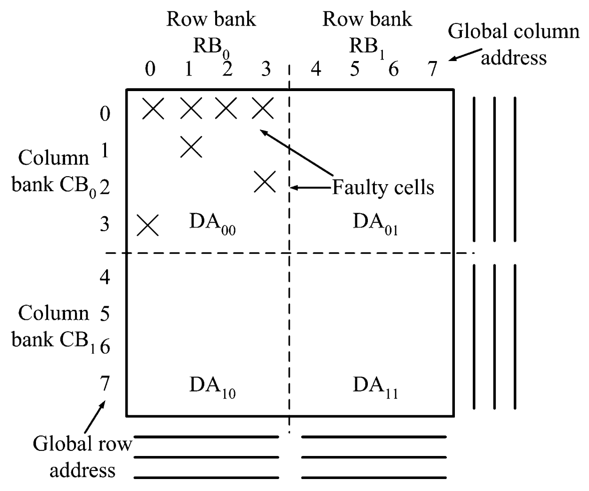

Fig. 8. Problem instance of the spare allocation problem.

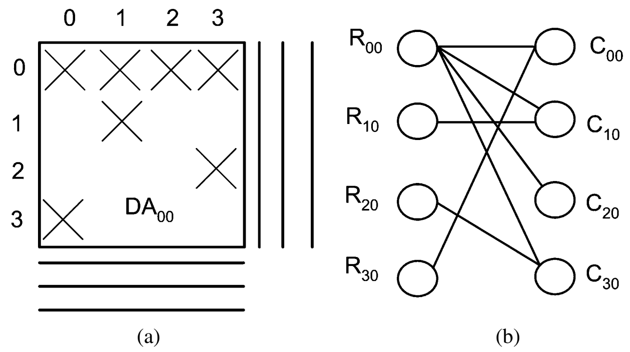

Fig. 9. (a) Faulty DA and (b) the corresponding bipartite graph.

redundant row (column) allocation. The faulty DA can be extracted as shown in Fig. 9(a), and its corresponding bipartite graph is shown in Fig. 9(b). Therefore, this case is equivalent to the problem addressed in [8]. The complexity is also NP-complete according to [8].

IV. Built-in Redundancy Analysis

As shown in the previous section, the redundancy allocation problem is NP-complete if the memory array is divided into divided arrays. To develop the built-in redundancy analysis (BIRA) circuit, the following should be noted: 1) the BIRA circuit should allocate redundancies in parallel with the BIST operation; 2) the hardware overhead should be low; and 3) the repair rate of the BIRA circuit should be high. In order to achieve these goals, a small array of size $p \times q$ is used for storing the local bitmap (LBM) instead of storing the whole bitmap [17]. The parameters p and q will affect the repair rate and the hardware overhead. They may also depend on the defect distribution. Therefore, they can be determined according to the tradeoff between the hardware overhead and the repair rate. Let LBM_{ij} denote an individual flag at the ith row and the jth column, $0 \leq i \leq p-1, 0 \leq j \leq q-1$.

The proposed ELRM is implemented by applying the local repair-most (LRM) algorithm [17] to each column bank sequentially. Let the local row (column) address of a cell denote the row (column) address of the cell within a DA. The $p \times q$ bitmap is accompanied by p local row address (LRA) tags $(\log_2(m/a)$

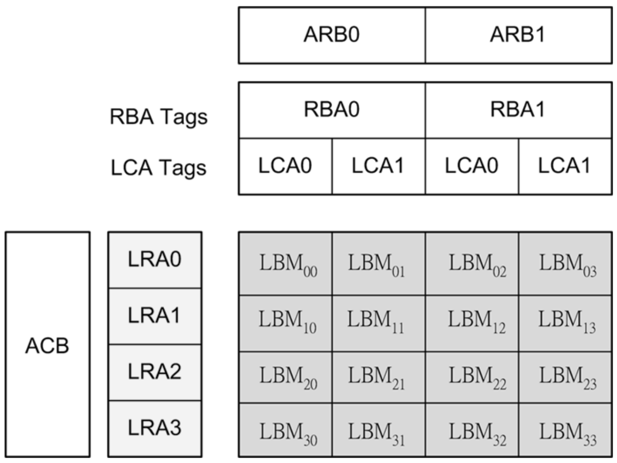

Fig. 10. Example bitmap of Fig. 6.

bits), q local column address (LCA) tags ($\log_2(n/b)$ bits), and b row bank address (RBA) tags ($\log_2 b$ bits).

For each RBA tag, there are q/b LCA tags associated with it. An example 4×4 bitmap of Fig. 6 is shown in Fig. 10, in which ARB_i denotes available row blocks in row bank $i, 0 \leq i \leq a-1$. Similarly, ACB denotes available column blocks of the target column bank.

The LRM algorithm is shown in the following pseudocode. It mainly consists of two procedures, i.e., bitmap construction (BC) and block allocation (BA). They are shown in the figure as BCFLRM() (BC for LRM) and BAFLRM() (BA for LRM), respectively. The local bitmap for each column is constructed by using BCFLRM(). Redundant row (column) blocks are allocated by using BAFLRM(). It is evident that the complexity of the ELRM algorithm is $O(N)$, where N denotes the4f number of memory cells.

```
**Procedure BCFLRM()**
$\text{LBM}_{ij} : 0 \leq i \leq p-1, 0 \leq j \leq q-1$ : an individual bit in
        the $i$th row and the $j$th column of the local bitmap
**(tag)** $ARB_i : 0 \leq i \leq b-1$: available row blocks in row bank $i$.
**(tag)** *ACB*: available column blocks of the target column bank
**(tag) LRA:** local row address tags to store the $m/a$ row addresses of the faulty cells that are recorded in the local bitmap.

  **(tag) LCA:** local column address tags to store the $n/b$ column addresses of the faulty cells that are recorded in the local bitmap.
**(counter) CounterLRA:** fault counters to count the faulty flags for LRA tags.
**(counter) CounterLCA:** fault counters to count the faulty flags for LCA tags.
**(counter) CounterR:** count the no. of faulty cells in a row of the bitmap.
**(counter) CounterC:** count the no. of faulty cells in a column of the bitmap
**Begin**
clear $\text{ARB}_i, \text{ACB}_j, \text{LBM}_{ij}$, repairFailFlag;
**Start:**
**for** each faulty cell detected () {
  set $\text{currentFaultycell} = [\text{local\_row} : \text{local\_col}][\text{RB}_j]$;
  **for** (ColumnBankIndex $i = 0; i < a; i++)\{$
    **for** all $\text{RB}_j, 0 \leq j \leq b-1\{$
      **for** $(\text{LCAIndex}y = 0; y < q/b; y++)\{$
        **if** $(\text{local\_col} = \text{LCA}y)\{$
          **if** $(\text{CounterLRA} < p)$
            store in bitmap [NextAvailableLRA$x$, LCA$y$];
            $\text{CounterLRA} = \text{CounterLRA} + 1$
          **else**
            **BAFLRM()**;//bitmap's row is full
          $\}$
        **else** {
            **for** (LRAIndex $x = 0; x < p/a; x++)\{$
            **if** $(\text{local\_row} = \text{LRA}x)\{$
              **if** $(\text{CounterLCA} < q)$
                store in bitmap [LRA$x$, NextAvailableLCA$y$];
                $\text{CounterLCA} = \text{CounterLCA} + 1$
              **else**
                **BAFLRM ();}}}}**//bitmap's column is full}
**BAFLRM ()**{
  **for** all $i, 0 \leq i \leq p-1\{$
    **if** $(\text{ARB}_i = 0$ and $\text{ACB} = 0)$
      set RepairFailFlag;      **elseif** $(\text{ACB} = 0)$
      **for** all RowBankIndex $j, 0 \leq j \leq b-1\{$
        **if** $(\text{ARB}_i < \text{CounterLRA})$//check early termination
          setRepairFailFlag;
        **else** AllocateSpareRow()};
      }
    **elseif** $(\text{ARB}_i = 0)\{$
        **if** $(\text{ACB} < \text{CounterLCA})$//check early termination
          setRepairFlag;
        **else** AllocateSpareColumn();}
    **else**{
        **if** FinalFaultyCell
          RepairAll();
        **else**
          RepairMost ();
          go to **Start**; }}
```

V. Example

We now consider an example of the ELRM algorithm. The memory array with defects is shown in Fig. 11. The total number of faults in this 16×16 memory array is 16, where $\text{SR} = 2$ and $\text{SC} = 3$. We can see that there are four DAs. Fig. 12 shows the intermediate contents of the bitmap by applying the ELRM algorithm. In the bitmap, only the local addresses of faulty cells are considered instead of the global addresses. Once the BIST circuit detects the faulty cells, the procedure BCFLRM() will be performed immediately and the LRM algorithm will be evaluated sequentially for each column bank. Since the BIRA and the BISR procedures are executed in parallel with the BIST procedure, we should consider the address order of the test sequence. In our experiment, we consider the test sequence by the way of the *row-wise* direction [24], [25], which is also denoted as the *fast-y up addressing*.

The coordinate of a faulty cell in the memory array can be expressed as F_{ij}^{kl}, where $k(l)$ represents the local row (column) address and ij represents the index number of the divided array.

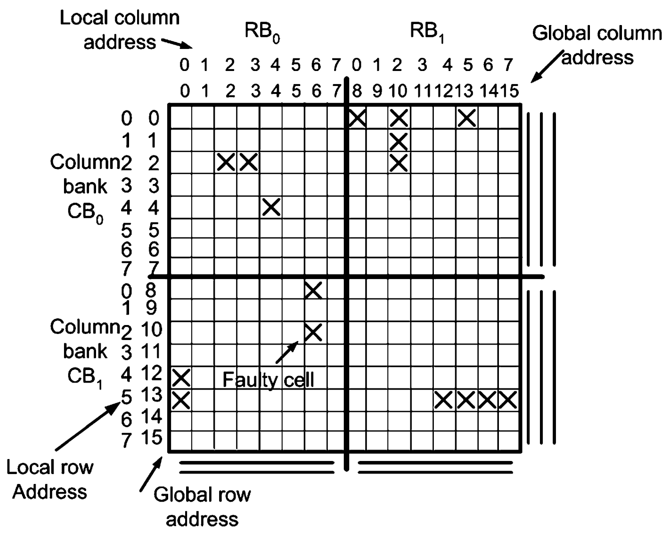

Fig. 11.Example memory array with 2-D redundancy.

For example, in Fig. 11, the first faulty cell detected by the BIST circuit is denoted as F^{00}_{01}, which means that the faulty cell is located at local address (0, 0) in the divided array DA_{01}. After executing the BCFLRM() procedure, this faulty cell will be recorded in the bitmap, as shown in Fig. 12(a). The next detected faulty cell is F^{02}_{01}, which is stored in the bitmap as shown in Fig. 12(b). The cell F^{05}_{01} is the third detected faulty cell during the test process. Since there is no available LCA tag to store the local column address of the third faulty cell, the repair-most (greedy) algorithm is performed for Fig. 12(b). A spare row block in RB_1 is used for replacing the first row block in DA_{01}. The bitmap will then be cleared, and the fourth detected faulty cell F^{12}_{01} can be stored in the bitmap, as shown in Fig. 12(c). Therefore, after storing the fifth, sixth, and seventh detected faulty cells,F^{22}_{00}, F^{23}_{00} and F^{22}_{01}, the bitmap is full again. Since there is no available register to store the eighth faulty cell F^{44}_{00}, the second faulty row block in RB_0 is replaced by a spare row block in RB_0, as shown in Fig. 12(d). After faulty cell F^{44}_{00} is detected, there is no more detected faulty cell in CB_0. The repair-most algorithm is then performed for the remaining bitmap's records, as shown in Fig. 12(d), for the final repair of CB_0. The final allocation result for CB_0 is shown in Fig. 12(e).

In the ELRM approach, the repair procedure is done on the column banks sequentially. Therefore, the register ACB is reset to 3 after the three faulty cells in CB_0 are repaired. For CB_1, the first six faulty cells $F^{06}_{10}, F^{26}_{10}, F^{40}_{10}, F^{50}_{10}, F^{54}_{11}$, and F^{55}_{11} are detected as shown in Fig. 12(f). The seventh detected faulty cell F^{56}_{11} cannot be stored in the bitmap. Therefore, this faulty row block will be replaced by a spare row block, as shown in Fig. 12(f).

Since there is no further faulty cells detected after the repair process in Fig. 12(f), the remaining faulty addresses in the bitmap can be repaired by the repair-most algorithm. The final result is shown in Fig. 12(g). As a result, the final spare allocation is shown in Fig. 13. With our ELRM approach, the replacement of spare row blocks and column blocks can be arranged globally for better utilization. This will further improve the repair rate.

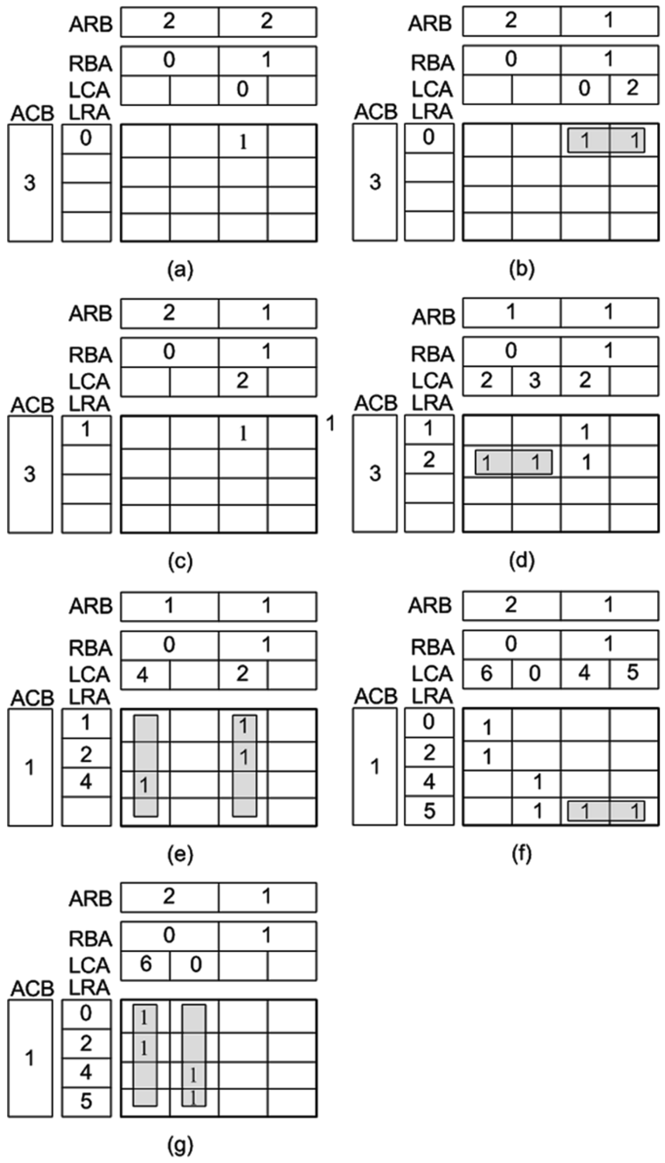

Fig. 12. Redundancy analysis of the ELRM algorithm.

For embedded memories, the proposed ELRM algorithm is executed by the corresponding BIST circuit to find the optimal allocation of redundancy elements. An address remapping circuit (usually a CAM) is used to store the repair information (addresses of faulty row/column blocks). The details about the remapping circuit can be found in [19]. In normal memory access, the accessed address is sent to the original address decoder and the address remapping circuit simultaneously. Since the address remapping circuit is implemented with a CAM, it is usually faster than the original address decoder. Therefore, the timing impact is almost negligible.

VI. Repair Rate and Hardware Overhead Analysis

Repair rate is defined as the probability of successful reconfigurations. In our simulation, we inject 17 random cell faults into a 1024×64 memory array. The types of the injected faults include stuck-at faults, a faulty cell, a faulty row, and a faulty column. During simulation, the probabilities of these fault types

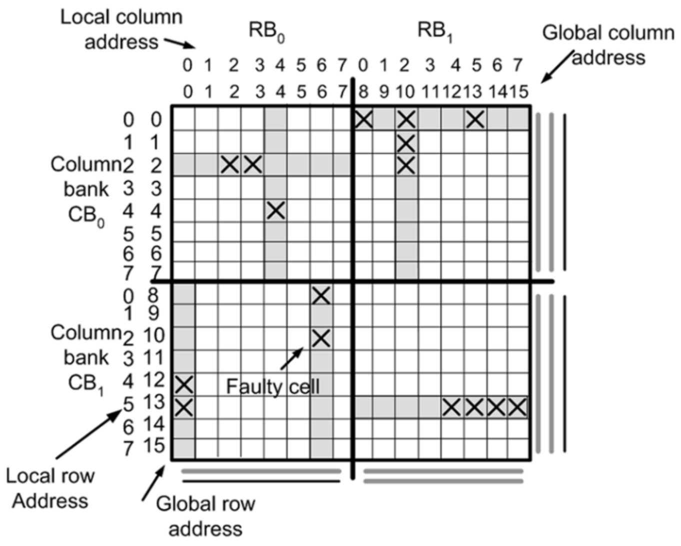

Fig. 13. Final allocation of the defective memory array.

TABLE I
REPAIR RATES FOR DIFFERENT SR, a, AND b VALUES

Repair Rates	# of Redundant Rows					
	4	5	6	7	8	9
a, b = 8	62.5%	71%	78.2%	82.5%	86%	91.3%
a, b = 4	60%	63.2%	71.8%	79.2%	83.2%	86.3%
a, b = 2	57.2%	60.1%	66.2%	72.1%	80%	83.2%

TABLE II
REPAIR RATES FOR DIFFERENT SIZES OF BITMAPS AND THE VALUES OF a AND b

Repair Rates	Size of Bitmap			
	8 × 8	8 × 16	16 × 16	32 × 32
a = b = 2	71.2%	73.4%	75.2%	78.0%
a = b = 4	76.2%	78.4%	79.5%	80.4%
a = b = 8	80.3%	81.8%	83.2%	86.6%

can be modified. Furthermore, more sophisticated fault types can also be injected by using our simulator. The injected faults must be first detected by the BIST session. The number of injected faults may be too large to be repaired by the added spare rows and columns. Table I shows the repair rates for different numbers of spare rows. The number of spare columns is assumed to be five $(\mathrm{SC} = 5)$. From this table, we can see that the repair rate increases if the number of redundant rows increases. Moreover, greater values of a and b will also result in greater repair rates. The repair rates for different sizes of bitmaps are shown in Table II, where $\mathrm{SR} = \mathrm{SC} = 6$. It is easy to see that a large bitmap will also result in a greater repair rate.

The *hardware overhead* required for implementing the ELRM algorithm is defined as the ratio between the transistor count of the extra components and the transistor count of the whole memory array. The extra components include the $p \times q$ bitmap, LCA and LRA registers, RBA registers, ARB and ACB registers, and the fault counters. These extra components can be found in Fig. 10 and the corresponding pseudocode. The transistor counts of the extra components can be easily estimated from their corresponding gate-level circuits. According

TABLE III
HARDWARE OVERHEAD FOR ELRM $(\mathrm{SC} = 5, \mathrm{U_{NIT}} : \%)$

SRAM Area Overhead	512K (512 × 1024)		1M (1024 × 1024)		2M (1024 × 2048)	
	a = 8 b= 8	a=16 b=16	a=8 b=8	a=16 b=16	a=8 b=8	a=16 b=16
SR = 2	0.565	0.579	0.289	0.293	0.148	0.152
SR = 4	0.572	0.593	0.293	0.304	0.150	0.155
SR = 8	0.579	0.607	0.296	0.311	0.152	0.159
SR = 16	0.586	0.622	0.300	0.318	0.153	0.162

TABLE IV
HARDWARE OVERHEAD FOR ELRM $(\mathrm{SC} = 5)$

DRAM Area Overhead	512K (512 × 1024)		1M (1024 × 1024)		2M (1024 × 2048)	
	a=8 b=8	a=16 b=16	a=8 b=8	a=16 b=6	a=8 b=8	a=16 b=16
SR = 2	3.391	3.476	1.738	1.781	0.890	0.919
SR = 4	3.434	3.562	1.759	1.823	0.901	0.923
SR = 8	3.476	3.647	1.781	1.866	0.911	0.954
SR = 16	3.519	3.733	1.802	1.909	0.922	0.976

to our evaluation, the results are shown in Tables III and IV for SRAMs and DRAMs, respectively. From these tables, we can find that the hardware overhead to implement the ELRM algorithm is almost negligible. For example, the hardware overhead for implementing this algorithm is below 0.17% for a 1024 × 2048-b SRAM.

Now we compare our work with the approach proposed in [27]. In [27], the spare columns are partitioned into spare column groups (SCGs) and segments. The spare rows are not partitioned into blocks. Therefore, it is evident that our approach will result in higher repair rates. In [27], an 8 K × 64-b SRAM is used to perform the simulation for repair rates. The number of injected random faults is from 1 to 10. The resulted repair rates are 86.09%, 96.10%, and 98.52%, respectively $(\mathrm{SC} = 4, \mathrm{SR} = 2, 3,$ and 4, respectively). If our redundant mechanism and the ELRM algorithm are used $(a = b = 8, p = q = 4)$, the achieved repair rates are 100%, 98.07%, and 96.31%, respectively. The hardware overhead to implement the BISR module of the repairable SRAM in [27] is about 4.6%. In our approach, the hardware overhead is about 5%. This increase is due to the feature that the spare rows are also partitioned into blocks.

VII. YIELD ANALYSIS

To simplify the yield analysis, three assumptions are made first. First, we assume that all failures on the memory chip are the result of *spot defects*. Spot defects are in contrast to global defects, which affect complete sections of a chip or wafer. Second, any single spot defect will result in the chip being inoperative unless some types of redundancy are included. Finally, spot defects are randomly distributed. We assume that each row block has an area A_{RB} and each column block has an area A_{CB}. The defect density function is denoted as $f(D)$, distributed between 0 and D_n. For ease of discussion, we use

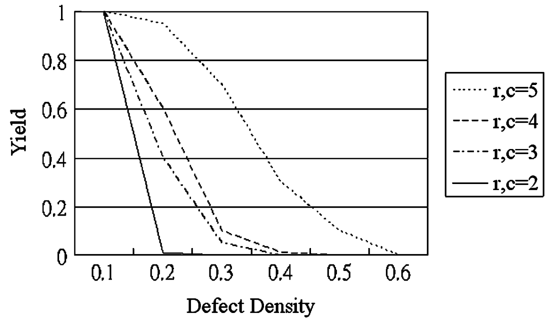

Fig. 14. Yield analysis for different r and c values.

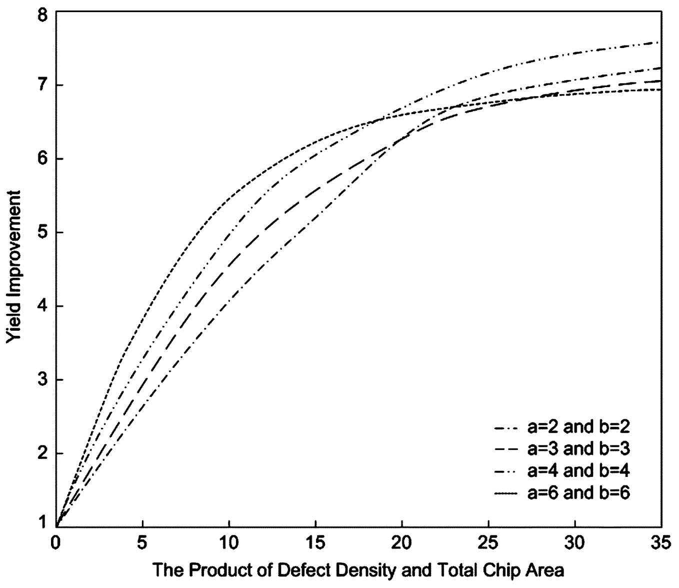

Fig. 15. Yield improvement of a 1024 × 64-b RAM, $\mathrm{SR} = 4$, and $\mathrm{SC} = 4$.

the Poisson distribution for yield modeling [21], [22]. The yield Y_0 of the nonredundant memory array is computed as

$$Y_0 = f(D) \int_0^{D_n} e^{-D(naA_{\mathrm{RB}}+mbA_{\mathrm{CB}})}\, dD.$$

Let P_{ELRM} be the probability of successful allocation (repair rate) with the proposed ELRM approach. Then, the yield Y_{ELRM} can be expressed as

$$Y_{\mathrm{ELRM}} = Y_0 + (1 - Y_0)P_{\mathrm{ELRM}}.$$

If more redundant elements are included, it will increase the repair rate. However, the extra redundant elements also increase the area, which may impact the yield. For this reason, a better measure for evaluating the benefit of redundancy is the effective yield improvement (YI) [24]

$$\mathrm{YI} = \frac{Y_{\text{with redundancy}}}{Y_{\text{without redundancy}}} \frac{A_{\text{without redundancy}}}{A_{\text{with redundancy}}}.$$

The effective yield improvement Y_{ELRM} with respect to the yield Y_0 are represented as

$$\mathrm{YI}_{\mathrm{ELRM}} = \frac{Y_{\mathrm{ELRM}}}{Y_{\text{without redundancy}}} \frac{A_{\text{without redundancy}}}{A_{\mathrm{ELRM}}}.$$

A 1024 × 64 b memory array is used for the yield simulation. Fig. 14 shows the yield with different numbers of redundancies under a certain defect density. It is shown that the yield is increased significantly as the number of redundancies grows. The yield improvements are plotted in Fig. 15. From this figure, if the product of the defect density and the total chip area is less than 17, then the memory array with more row (column) banks will have a higher yield improvement. However, if the product is higher than 17, the added BISR circuit will also incur more negative effects for the yield improvement. A higher number of banks does not necessarily achieve the maximum yield improvement. The optimal number of banks can be obtained from the simulation results.

VIII. Conclusion

We have proposed a novel redundancy repair mechanism based on the DWL and DBL techniques, which can be used in today's SOC designs. The reconfiguration is performed at the row (column) block level instead of the conventional row (column) level. Due to the efficient usage of redundancy, the repair rate and fabrication yield can be improved significantly. Experimental results also reveal this fact. Based on the proposed redundancy mechanism, we also have shown that the complexity of the optimal redundancy allocation problem is NP-complete, and a heuristic ELRM algorithm suitable for built-in implementation has also been proposed. According to the simulation results, the hardware overhead for implementing this algorithm is below 0.17% for a 1024×2048 SRAM, showing the cost-effectiveness of our approach as compared with the original RM algorithm.

References

[1] A. Allan *et al.*, "2001 technology roadmap for semiconductors," *Computer*, vol. 35, no. 1, pp. 42–53, Jan. 2002.

[2] D. K. Bhavsar, "An algorithm for row-column self-repair of RAM's and its implementation in the Alpha 21 264," in *Proc. Int. Test Conf.*, Sep. 1999, pp. 311–318.

[3] I. Kim, Y. Zorian, G. Komoriya, H. Pham, F. P. Higgins, and J. L. Lewandowski, "Built in self repair for embedded high density SRAM," in *Proc. Int. Test Conf.*, Oct. 1998, pp. 1112–1119.

[4] W. K. Huang, Y. H. Shen, and F. Lombrardi, "New approaches for the repairs of memories with redundancy by row/column deletion for yield enhancement," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 9, no. 3, pp. 323–328, Mar. 1990.

[5] M. Horiguchi, J. Etoh, M. Aoki, K. Itoh, and T. Matsumoto, "A flexible redundancy technique for high-density DRAM's," *IEEE J. Solid-State Circuits*, vol. 26, no. 1, pp. 12–17, Jan. 1991.

[6] 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. Comput-Aided Des. Integr. Circuits Syst.*, vol. 12, no. 1, pp. 24–36, Jan. 1993.

[7] H. C. Kim, D. S. Yi, J. Y. Park, and C. H. Cho, "A BISR (built-in self-repair) circuit for embedded memory with multiple redundancies," in *Proc. Int. Conf. VLSI CAD*, Oct. 1999, pp. 602–605.

[8] S.-Y. Kuo and W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays," *IEEE Design Test*, vol. 4, pp. 24–31, 1987.

[9] M. Yoshimoto, K. Anami, H. Shinohara, T. Yoshihara, H. Takagi, S. Nagao, S. Kayano, and T. Nakano, "A divided word-line structure in the static RAM and its application to a 64K full CMOS RAM," *IEEE J. Solid-State Circuits*, vol. SC-18, no. 5, pp. 479–485, Oct. 1983.

[10] A. Karandidar and K. K. Parhi, "Low power SRAM design using hierarchical divided bit-line approach," in *Proc. Int. Conf. Computer Design*, Oct. 1998, pp. 82–88.

[11] C.-H. Hsu and S.-K. Lu, "Novel fault-tolerant techniques for high capacity RAMs," in *Proc. Pacific Rim Int. Dependable Computing Symp.*, 2001, pp. 11–18.

[12] M. Tarr, D. Boudreau, and R. Murphy, "Defect analysis system speeds test and repair of redundant memories," *Electronics*, pp. 175–179, Jan. 1984.

[13] J. R. Day, "A fault-driven comprehensive redundancy algorithm for repair of dynamic RAM's," *IEEE Design Test*, vol. 2, no. 3, pp. 33–44, 1985.

[14] 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.*, Oct. 2000, pp. 567–574.

[15] C.-L. Wey and F. Lombardi, "On the repair of redundant RAM's," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. CAD-6, no. 2, pp. 222–231, Mar. 1987.

[16] 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. Comput.-Aided Des. Intergr. Circuits Syst.*, vol. 9, no. 3, pp. 323–328, Mar. 1990.

[17] C.-T. Huang, C.-F. Wu, J.-F. Li, and C.-W. Wu, "Built-in redundancy analysis for memory yield improvement," *IEEE Trans. Reliability*, vol. 52, pp. 386–399, Dec. 2003.

[18] S.-K. Lu, "Built-in self-repair techniques for embedded RAM's," in *Proc. IEE Computers Digit. Tech.*, vol. 150, Jul. 2003, pp. 201–208.

[19] S.-K. Lu, "A novel built-in self-repair approach for embedded RAMs," *J. Electron. Testing: Theory Applicat.*, vol. 19, pp. 315–324, Jun. 2003.

[20] J. A. Mchugh, *Algorithm Graph Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[21] C. H. Stapper, "Improved yield models for fault-tolerant memory chips," *IEEE Trans. Computers*, vol. 42, no. 7, pp. 872–881, Jul. 1993.

[22] W. Kuo and T. Kim, "An overview of manufacturing yield and reliability modeling for semiconductor products," *Proc. IEEE*, vol. 87, no. 8, pp. 1329–1344, Aug. 1999.

[23] I. Koren and Z. Koren, "Defect tolerance in VLSI circuits: Techniques and yield analysis," *Proc. IEEE*, vol. 86, no. 9, pp. 1819–1836, Sep. 1998.

[24] A. J. van de Goor, "Testing semiconductor memories—Theory and practice,", Apr. 1996.

[25] A. J. van de Goor and I. Schanstra, "Address and data scrambling: Causes and impact on memory tests," in *Proc. Int. Workshop Design, Test Applicat.*, Jan. 2002, pp. 128–136.

[26] J. Day, "A fault-driven comprehensive redundancy a algorithm," *IEEE Design Test Comput.*, vol. 2, no. 3, pp. 35–44, Jun. 1985.

[27] J.-F. Li, J.-C. Yeh, R.-F. Huang, and C. W. Wu, "A built-in self-repair scheme for semiconductor memories with 2-D redundancy," in *Proc. Int. Test Conf.*, vol. 1, 2003, pp. 393–402.

[28] S.-K. Lu and S.-C. Huang, "Built-in self-test and repair (BISTR) techniques for embedded RAMs," in *Proc. Int. Workshop Memory Technol., Design Testing*, Aug. 2004, pp. 60–64.

[29] S.-K. Lu and C.-H. Hsu, "Built-in self-repair for divided word line memory," in *Proc. Int. Symp. Circuits Syst.*, pp. IV13–IV16.

[30] C.-H. Hsu and S.-K. Lu, "Fault-tolerance design of memory systems based on DBL structures," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, Oct. 2002, pp. 221–224.

[31] S. Nakahara, K. Higeta, M. Kohno, T. Kawamura, and K. Kakitani, "Built-in self-test for GHz embedded SRAM's using flexible pattern generator and new repair algorithm," in *Proc. Int. Test Conf.*, Sep. 1999, pp. 301–310.

Shyue-Kung Lu received the M.S. degree from National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 1991 and the Ph.D. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1996, both in electrical engineering.

From 1995 to 1998, he was an Associate Professor with the Department of Electrical Engineering, Lunghwa Junior College of Technology and Commerce, Taoyuan, Taiwan. Since 1998, he has been with the Department of Electronics Engineering, Fu-Jen Catholic University, Taipei, Taiwan, where he is a Professor. His research interests include the areas of VLSI testing and fault-tolerant computing, video coding techniques, and architectures design.

Yu-Chen Tsai received the B.S.E.E. degree from National Central University, Jhongli, Taiwan, R.O.C., in1998 and the M.S. degree in electronic engineering from Fu-Jen Catholic University, Taipei, Taiwan, R.O.C., in 2004.

His research interests include memory testing, VLSI design, and BIST techniques. He is currently an Engineer with Socle Technology Corporation, Hsinchu, Taiwan, R.O.C., investigating resuable IP design and verification.

Chih-Hsien Hsu received the B.S. and M.S. degrees from Fu-Jen Catholic University, Taipei, Taiwan, R.O.C., in 1999 and 2001, respectively, and is currently working toward the Ph.D. degree at National Tsing Hua University, Hsinchu, Taiwan, R.O.C.

His research interests include VLSI design and testing, as well as the testing and self-repair of semiconductor memory. Since 2002, he has been an Engineer with National Chip Implementation Center (CIC), Hsinchu.

Kuo-Hua Wang received the B.S. and M.S. degrees, both in computer engineering, and the Ph.D. degree in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1986, 1988, and 1994, respectively.

He is currently an Associate Professor with the Department of Computer Science and Information Engineering, Fu-Jen Catholic University, Taipei, Taiwan, R.O.C. His teaching and research interests include design automation of VLSI, logic synthesis and optimization, logic verification, and high-level synthesis.

Cheng-Wen Wu received the B.S.E.E. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1981 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of California, Santa Barbara (UCSB), in 1985 and 1987, respectively.

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 Researcher with the Electrical and Computer Engineering Department, UCSB. He then served as the Chair of the Electrical Engineering Department, NTHU, from 2000 to 2003, and the Director of the IC Design Technology Center from 2000 to 2005. He is currently the Dean of the College of Electrical Engineering and Computer Science. He is the Editor-in-Chief for the *International Journal of Electrical Engineering* and an Editor for the *Journal of Electronic Testing: Theory and Applications*. He is interested in design and test of high performance VLSI circuits and systems.

Dr. Wu is a Life Member of the Chinese Institute of Electrical Engineers (CIEE) and the Taiwan IC Design Society. Dr. Wu was the Technical Program Chair of the IEEE Fifth Asian Test Symposium (ATS'96), the General Chair of ATS'00, and the General Chair of 2005 IEEE International Workshop on Memory Technology, Design, and Test (MTDT). He was the recipient of the Distinguished Teaching Award from NTHU in 1996, the Outstanding Electrical Engineering Professor Award from the Chinese Institute of Electrical Engineers in 1997, the Distinguished Research Awards from National Science Council in 2000 and 2002, the Industrial Collaboration Awards from the Ministry of Education in 2001 and 2003, the Best Paper Award of the 2002 IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems, the Best Paper Award of the 2003 IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), and the Special Feature Award of the 2003 ASP-DAC University LSI Design Contest.