A Built-In Redundancy-Analysis Scheme for RAMs with 2D Redundancy Using 1D Local Bitmap

Tsu-Wei Tseng, Jin-Fu Li, and Da-Ming Chang
Advanced Reliable Systems (ARES) Laboratory
Department of Electrical Engineering
National Central University
Jhongli, Taiwan 320

Abstract

Built-in self-repair (BISR) technique is gaining popular for repairing embedded memory cores in system-on-chips (SOCs). To increase the utilization of memory redundancy, the BISR technique usually needs to perform built-in redundancy-analysis (BIRA) algorithm for redundancy allocation. This paper presents an ef£cient BIRA scheme for embedded memory repair. The BIRA scheme executes the 2D redundancy allocation based on the 1D local bitmap. This enables that the BIRA circuitry can be implemented with low area cost. Also, the BIRA algorithm can provide good repair rate (i.e., the ratio of the number of repaired memories to the number of defective memories). Experimental results show that the repair rate of the proposed BIRA scheme approximates to that of the optimal scheme for the memories with different fault distributions. Also, the ratio of the analysis time to the test time is small.

1 Introduction

Embedded memories occupy more than 60% chip area of most today's system on chip (SOC) designs. Keeping the memory cores at a reasonable yield level is thus vital for SOC products. Built-in self-repair (BISR) techniques have been shown to improve the memory yield from 5% to 20%, such that the net SOC yield increase can range from 2% to 10% [1]. Therefore the BISR technique is gaining popularity [2]. For a BISR design, however, the redundancy allocation for a memory with 2D (i.e., spare rows and spare columns) redundancy is very dif£cult since it is an NP-hard problem [3].

Many redundancy analysis algorithms for performing redundancy allocation at automatic test equipment (ATE) have been proposed, e.g., [3–6]. However, these algorithms are not adopted to be realized in built-in circuitries. Thus ef£cient built-in redundancy-analysis (BIRA) algorithms which can cost-effectively be realized with built-in circuitries are required for BISR schemes. In [7], the authors presented a BISR design with a comprehensive real-time exhaustive search test and analysis (CRESTA) scheme for bit-oriented memories. This BISR design can achieve the optimal repair rate (i.e., the ratio of the number of repaired memories to the number of defective memories). However, this scheme is only for bit-oriented memories and the hardware cost for implementing the CRESTA scheme is very high. In [8], the authors extend the CRESTA scheme to support the repair of word-oriented memories. A column repair vector is used to store information for column repair temporarily, such that the BIRA design can be executed at-speed. Again, the hardware cost of CRESTA increases drastically with the number of spare elements. Although one sequential BIRA scheme was also proposed to cope with this problem, the test time is increased drastically. In [9], a BISR scheme using redundant words is presented. However, redundancy analysis is not required in this scheme since only 1D redundancy is used. In [10], an ef£cient BIRA algorithm for word-oriented memories is reported. In this work, if a faulty word has multiple faulty subwords, it is repaired with an available spare row. This can reduce the area cost of the corresponding BIRA implementation. In [11], two ef£cient BIRA algorithms using 2D local bitmap are proposed. However, only the applications of the two algorithms for bit-oriented memories are discussed. If the user wants to extend the algorithms to repair word-oriented memories, each bit of 2D local bitmap needs to be replaced by a word. This results that the area cost for implementing the local bitmap is large.

This paper presents an ef£cient BIRA scheme for word-oriented memories with 2D redundancy. The BIRA algorithm can allocate the 2D redundancy based on 1D local bitmap. This results that the area cost for implementing the BIRA algorithm is low. Also, the BIRA can provide high repair rate for the defective word-oriented memories. The ratio of the analysis time to the test time is also analyzed. Experimental results show that proposed BIRA scheme can provide good repair rate for memories with different fault distributions. The repair rate of the proposed BIRA scheme approximates to that of the optimal analysis algorithm. Also, the ratio of the analysis time to the test time is only about 3.3% for a 4096×128-bit memory tested by the March-CW algorithm.

2 Preliminary

Assume that a bit-oriented memory has r spare rows and c spare columns. If we design a BIRA algorithm based on a 2D local bitmap, the maximum size of the 2D local bitmap for the memory can be limited to $(r(c+1)+r) \times (c(r+1)+c)$ [11] as shown in Fig. 1. As Fig. 1 shows, in addition, each bit of the bitmap needs one 1-bit register to store the error information, $(r(c+1)+r)$ row address registers

3-9810801-0-6/DATE06 © 2006 EDAA

(RARs) and $(c(r+1)+c)$ column address registers (CARs) are needed. The width of each RAR and CAR is equal to the number of row address bits and column address bits, respectively. Also, the RARs and CARs must be able to perform the function of parallel comparison. For brevity, if we estimate the area of a local bitmap only in terms of registers, $(r(c+1)+r) \times (c(r+1)+c) + m(c(r+1)+c) + n(r(c+1)+r)$ registers are needed to implement a local bitmap for a bit-oriented memory with n-bit row address and m-bit column address.

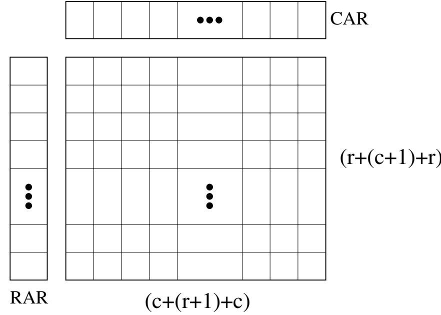

Figure 1: Local bitmap with maximum size.

In a similar way, a local bitmap for bit-oriented memories can be extended to a local bitmap for word-oriented memories by replacing every bit of the local bitmap for bit-oriented memories with one word. Consider an $N\times W$-bit memory. For brevity, we assume that N=2^u and u=n+m, where n and m represent the number of row address bits and column address bits, respectively. If the word-oriented memory has the same redundancy organization as the case described above, the number of registers for implementing the local bitmap is $W \times (r(c+1)+r) \times (c(r+1)+c) + m(c(r+1)+c)+n(r(c+1)+r)$. Therefore, the area cost of the local bitmap is dominated by the number of spare elements and word width. Note that the area cost is increased with the number of spare rows and columns in quadratic relation. Thus if only a 1D local bitmap is used for 2D redundancy allocation, the area cost can be reduced drastically. However, the problem is how to execute a BIRA algorithm for 2D redundancy allocation in a 1D local bitmap. In the next section, we present a BIRA scheme for word-oriented memories with 2D redundancy using 1D local bitmap.

3 Proposed BIRA Scheme

3.1 Block Diagram of the Proposed BIRA

Figure 2 shows the targeted memory architecture. The memory array is composed of two subarrays and has two kinds of spare elements: spare rows and spare columns. The spare row is global redundancy, but the spare column is local redundancy. That is, a spare row can repair the corresponding defective rows in the right and left subarrays. However, a defective column in right (left) subarray only can be repaired by the corresponding spare column in the right (left) subarray.

Figure 3 shows the block diagram of the proposed BIRA scheme. The BIRA scheme consists of four major blocks: an FSM, a Bitmap, a C_{MF} Detector, and a Remapping Data Register. The FSM executes the analysis procedure of the proposed BIRA algorithm. The Bitmap stores the faulty information from the BIST circuitry. Also, it can perform comparison function, which compares the stored information with the incoming information such that the information does not be stored repeatedly. The B_{MF} Detector can detect the column with maximal number of faulty cells within the Bitmap. The Remapping Data Register stores the repaired addresses, and these addresses are shifted to the fuse group of the repaired memory once the BIRA scheme is completed.

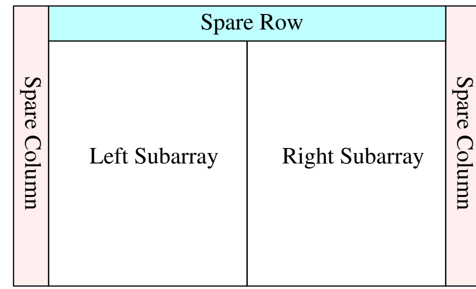

Figure 2: The targeted memory redundancy organization.

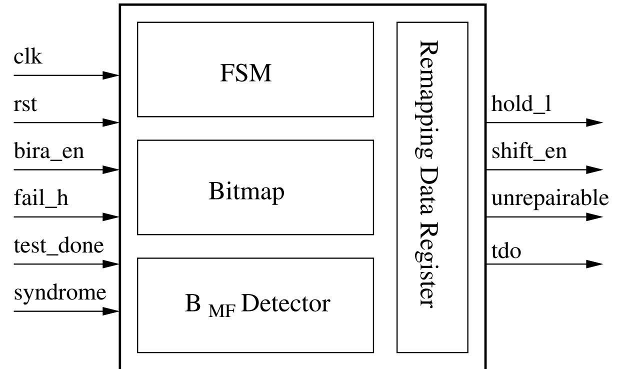

Figure 3: Block diagram of the proposed BIRA scheme.

In addition to the clock and reset signals (clk and rst), the BIRA has four input signals: bira_en, fail_h, test_done, and syndrome. The signals fail_h, test_done, and syndrome are from the BIST circuitry. As soon as the BIST detects a fault, the BIST is frozen. Also, the fail_h signal is asserted and the corresponding faulty information (faulty address and hamming syndrome) is exported to the BIRA through the bus—syndrome. The hamming syndrome is de£ned as the modulo-2 sum of the expected (fault-free) data output vector and the output vector from the memory under test, for word-oriented RAMs [12]. On the other hand, the BIRA has four output signals: hold_l, shift_en, unrepairable, and tdo. The hold_l is used to awake the BIST circuitry. If the BIRA completes the redundancy analysis of the current fault pattern in the Bitmap, the BIRA sets the hold_l to logic high and the BIST continues to perform the testing process. The unrepairable signal is asserted once the spare elements are exhausted and there is faulty information in the Bitmap. Otherwise, when the BIST and BIRA have been completed, the repaired information stored in the Remapping Data Register is shifted to the fuse group by asserting the shift_en to

high and exporting the data from tdo. In this paper we only focus on the discussion of the proposed BIRA algorithm and 1D bitmap.

3.2 1D Local Bitmap

Figure 4 shows the proposed 1D local bitmap. As the £gure shows, the bitmap has x rows and each row is composed of four £elds: a one-bit valid ¤ag (VF), a row address register (RAR), a column address register (CAR), and a hamming syndrome register (HSR). If a row stores valid information, the bitmap sets the corresponding VF to logic 1. The RAR and CAR store the faulty address of the detected faulty word. Also, the HSR stores the corresponding hamming syndrome of the faulty word. Therefore, the BIST circuitry exports the faulty information to the bitmap while it detects a fault. The faulty information includes a fail ¤ag, a faulty address (including row and column addresses), and a hamming syndrome. The size of the local bitmap can be limited to $(r + c) \times (1 + n + m + W)$. Since the memory under test is unrepairable if the maximal number of orthogonal faulty cells (a faulty cell which does not share any row or column with any other faulty cell is referred to as an orthogonal faulty cell [11]) is larger than the number of spare elements. Note that the selection of the value of x is dependent on the trade-off between the area cost and repair rate.

VF_0	RAR_0	CAR_0	HSR_0
VF_1	RAR_1	CAR_1	HSR_1
⋮			
VF_i	RAR_i	CAR_i	HSR_i
⋮			
VF_{x-1}	RAR_{x-1}	CAR_{x-1}	HSR_{x-1}

Figure 4: Organization of the 1D local bitmap.

3.3 BIRA Algorithm

Before the description of the proposed BIRA algorithm, we £rst de£ne the notations used in the BIRA algorithm as follows: (1) N_{ASR}—number of available spare rows; (2) LN_{ASC} (RN_{ASC})—number of available spare columns in left (right) subarray; (3) N_{FR}—number of faulty rows in the half Bitmap in which the number of available spare columns is exhausted; (4) G_{MC}—the group has maximal number of rows with the same column address in the Bitmap; (5) B_{MF}—the bit position in the HSRs within the G_{MC} has the largest number of faults; (6) RA—row address of the fault currently detected by the BIST; (7) CA—column address of the fault currently detected by the BIST; (8) <RAR>—valid addresses stored in RAR of the Bitmap; (9) <CAR>—valid addresses stored in CAR of the Bitmap; (10) N_{VRAR}—number of RARs storing valid row addresses; (11) N_{VCAR}—number of CARs storing valid column addresses.

Subsequently, examples are illustrated to further explain the notations: G_{MC}, B_{MF}, and N_{FR}. Consider a Bitmap with four entries as shown in Fig. 5. Assume that data width of the memory under test is 6. Also, let N_{ASR}=1, LN_{ASC}=1, and RN_{ASC}=1. As Fig. 5(a) shows, all entries of the Bitmap store valid addresses. Also, the column addresses stored in the £rst three entries all are the same. Therefore, the group of entry 0, entry 1, and entry 2 (E0, E1, and E2) is called G_{MC} (marked with the box in dash line). The B1 is called B_{MF} since the number of faults in it is the largest within the G_{MC}. If multiple bits within the G_{MC} have the same number of faults and the number of faults is the largest, the LSB is selected as the B_{MF}. Assume that LN_{ASC}=0 and the content of the Bitmap is shown as Fig. 5(b). That is, the spare column of the left subarray (i.e., the range of B0, B1, and B2) is exhausted. As Fig. 5(b) shows, therefore, the number of faulty rows in the left subarray is 2, i.e., N_{FR}=2.

	VF	RAR	CAR	HSR B0	B1	B2	B3	B4	B5
E0	1	1	2	0	1	0	0	0	0
E1	1	2	2	0	0	1	0	0	0
E2	1	3	2	0	1	0	0	0	0
E3	1	4	3	0	0	0	1	0	0

(a)

	VF	RAR	CAR	HSR B0	B1	B2	B3	B4	B5
E0	1	1	2	0	0	0	0	1	0
E1	1	2	2	0	0	1	0	0	0
E2	1	3	2	0	0	1	0	0	0
E3	1	4	3	0	0	0	1	0	0

(b)

Figure 5: (a) Example for G_{MC} and B_{MF}. (b) Example for N_{FR}.

Algorithm 1 summarizes the redundancy-analysis procedure of the proposed BIRA algorithm. Assume that the number of entries of the Bitmap is X. In Phase-1 analysis procedure, the BIRA performs redundancy allocation while the BIST detects a fault. If the address of the detected fault is the same as the ith faulty address stored in the Bitmap (i.e., RA∈<RAR> and CA∈<CAR>), the hamming syndrome of the detected fault is stored in HSR_i. Also, the content of the HSR_i is updated to the content which is obtained by performing the bit-wise OR operation of the original content and the hamming syndrome of the detected fault. If the row address of the detected fault is the same as one of row addresses stored in the Bitmap and the column address of the detected fault is different from all the column addresses stored in the Bitmap, the BIRA algorithm allocates an available spare row to repair the corresponding faulty row. Then the Bitmap is updated and the BIRA awakes the frozen BIST circuitry. However, if the number of available spare rows is exhausted, the memory is unrepairable. If the address of the detected fault does not belong to the two cases mentioned above, the BIRA stores the faulty address in an empty entry of the Bitmap. Then the BIRA checks whether N_{ARAR} or/and N_{ACAR} is full or not. If both the two registers are not full, the BIRA issues a commend to resume

the BIST circuitry. Otherwise, the BIRA goes to execute the analysis procedure of Subroutine.

The Phase-2 analysis procedure is executed while the memory BIST is completed. The BIRA £rst checks whether the Bitmap has valid information or not. If the Bitmap is not empty, the BIRA performs the analysis procedure of the Subroutine. Otherwise, the spare allocation is completed and the memory is repairable.

Algorithm 1 Pseudo code of the BIRA algorithm.

Phase-1: during BIST

```
 1: FOREACH (Detected fault){
 2:     IF (RA∈<RAR> and CA∈<CAR>){
 3:         HSR_i=HSR_i | hamming syndrome;
 4:         Update Bitmap and return to perform BIST;}
 5:     ELSE IF (RA∈<RAR> and CA∋<CAR>){
 6:         IF (N_ASR!=0){
 7:             Allocate a spare row to repair the faulty row;
 8:             Update Bitmap and return to perform BIST;}
 9:         ELSE{
10:             Memory is unrepairable;}}
11:     ELSE {
12:         Store the fault in an empty entry of the Bitmap;
13:         IF (N_ARAR<X and N_ACAR<X){
14:             Return to perform BIST;}
15:         ELSE {
16:             Go to Subroutine;}}}
```

Phase-2: BIST is completed

```
 1: IF (N_ARAR≠0 or N_ACAR≠0){
 2:     Go to Subroutine;}
 3: ELSE {
 4:     Memory is repairable;}
```

Subroutine

```
 1: IF (LN_ASC!=0 or RN_ASC!=0){
 2:     IF (LN_ASC=0/RN_ASC=0 and left/right bitmap isn't empty){
 3:         If (N_FR in the left or right subarray<N_ASR){
 4:             Repair all faulty rows with spare rows;
 5:             Update Bitmap and return to perform BIST;}
 6:         ELSE {
 7:             Memory is unrepairable;}}
 8:     ELSE{
 9:         Repair the column with B_MF with a spare column;
10:         Update Bitmap;
11:             IF (Bitmap is full){
12:                 Go to Subroutine;}
13:             ELSE {
14:                 Return to perform BIST;}}
15: ELSE {
16:     IF (N_ASR!=0){
17:         Allocate a spare row to repair one row in the Bitmap;
18:         Update Bitmap and return to perform BIST;}
19:     ELSE {
20:         Memory is unrepairable;}}
```

The analysis procedure of the Subroutine is performed when the Bitmap is full or the memory BIST is completed and the Bitmap is not empty. The Subroutine £rst checks whether the available spare columns in the left or/and right subarrays are exhausted or not. If either LN_{ASC} or RN_{ASC} is exhausted, the BIRA checks whether the corresponding left-half bitmap or right-half bitmap is empty or not. If LN_{ASC}=0 (RN_{ASC}) and the left-half (right-half) bitmap is not empty, the Subroutine checks whether the value of N_{FR} in the left subarray or the right subarray is larger than the number of available spare rows. If the value of N_{FR} is less than that of N_{ASR}, all the corresponding faulty rows in the left or the right subarray are replaced with the corresponding spare rows. On the contrary, if the N_{FR} is larger than the N_{ASR}, the memory is unrepairable. Otherwise, if both the BIRA identi£es the B_{MF} within a G_{MC} and the B_{MF} is replaced with an available corresponding spare column. Then the Bitmap is updated and the BIRA checks whether the Bitmap is full or not. If the Bitmap still is full, the BIRA executes the Subroutine again. Otherwise, the BIRA resumes the BIST circuitry. Finally, if both LN_{ASC} and RN_{ASC} are exhausted, the BIRA allocates an available spare row to repair one faulty row in the Bitmap. Then the Bitmap is updated and the BIRA resumes the BIST circuitry. However, if the available spare rows are exhausted, the memory is unrepairable.

An example is illustrated to explain the Algorithm 1 further. Consider a memory under repair has two spare rows and two spare columns (one is in the left subarray and the other is in the right subarray). Also, the word width of the memory is 6 and the number of entires of Bitmap is 4. Assume that the Bitmap is full and the fault pattern is shown as Fig. 6(a). Note that B_0, B_1, and B_2 (B_3, B_4, and B_5) belong to left (right) subarray region. As Fig. 6(a) shows, the four faults detected do not have the same row addresses. Thus no spare row allocation is done before the Bitmap is full. Once the Bitmap is full, the BIRA algorithm goes to execute the Subroutine. Because all the spare elements are not used, the Subroutine will allocate a spare column to replace the faulty bit with the largest number of faults, i.e., B_{MF}. In this case, the £rst three entries has the same column addresses. Thus they are the G_{MC}. Also, the B_0, B_1, and B_2 all have 1 faulty bit. Since the LSB bit position has the highest priority, the B_0 is selected as the B_{MF}. Thus B_0 is replaced by the spare column in the left subarray. After the B_0 is replaced, the left-half Bitmap is not empty and LN_{ASC}=0. Because N_{FR} in the left-half bitmap is 2, the BIRA allocates two spare rows to repair the row 1 and row 2. Then the Bitmap is updated and the BIRA informs the BIST to test the memory. Assume that no other fault is detected when BIST is completed, the BIRA executes the Subroutine again. Since $RN_{ARAR} \neq 0$ and right-half bitmap is not empty, the BIRA searches the B_{MF}. Since the B_3 is B_{MF}, the B_3 is replaced by the spare column at the right subarray. Finally, the memory is repairable and the redundancy allocation is shown in Fig. 6(b).

4 Experimental Results

We implement a simulator to evaluate the proposed BIRA algorithm. The simulator is implemented according to the approach reported in [13]. The repair ef£ciency is estimated by the parameter repair rate. Repair rate is de£ned as the ratio of the number of repaired memories to the number of defective memories [11]. In our analysis, we simulated a total of 500 memory cores with a core size of 8192×64 bits. Different types of faults are injected into each memory cores. In this paper, we show the simulation results for two different cases. For Case 1, only single-cell

	VF	RAR	CAR	HSR B0	B1	B2	B3	B4	B5
E0	1	1	2	1	0	1	0	0	0
E1	1	2	2	0	1	0	0	0	0
E2	1	3	2	0	0	0	1	0	0
E3	1	4	3	0	0	0	1	0	0

(a)

	VF	RAR	CAR	HSR B0	B1	B2	B3	B4	B5
E0	1	1	2	1	0	1	0	1	0
E1	1	2	2	0	1	0	0	0	0
E2	1	3	2	0	0	0	1	0	0
E3	1	4	3	0	0	0	1	0	0

(b)

Figure 6: An example of the redundancy analysis: (a)before analysis; (b)after analysis.

faults are injected into each memory core at random locations. Table 1 summarizes the repair rates of the optimal analysis algorithm and our BIRA algorithm for Case 1. In the table, the r and c denote the number of spare rows and spare columns. Note that the spare columns are evenly divided in left and right subarrays. That is, if c=2, one is for the left subarray and the other is for the right subarray. As Table 1 shows, the repair rate of the proposed BIRA algorithm can achieve very good repair rate. For all the simulated redundancy con£gurations, the repair rates of our algorithm approximate to those of the optimal algorithm. For Case 2, 40% single-cell faults, 20% row twin-bit faults (i.e., two adjacent bits in a row are faulty), 20% column twin-bit faults (two adjacent bits in a column are faulty), and 20% cluster faults (the cluster size is 2×2) are injected into each memory core at random locations. Table 2 summarizes the repair rates of the optimal analysis algorithm and our BIRA algorithm for Case 2. As the table shows, we see that the repair rate of the proposed BIRA algorithm is still very good for a memory with different fault types.

Table 1: Repair rates of the optimal algorithm and our BIRA algorithm for Case 1.

(r,c)	Opt.	Ours	(r,c)	Opt.	Ours
(0,2)	27.6%	27.6%	(2,4)	96.0%	96.0%
(0,4)	65.6%	65.4%	(3,0)	62.0%	62.0%
(1,0)	12.4%	12.4%	(3,2)	93.2%	93.2%
(1,2)	57.4%	57.4%	(3,4)	99.2%	99.2%
(1,4)	85.4%	83.4%	(4,0)	87.0%	87.0%
(2,0)	44.4%	44.4%	(4,2)	97.4%	97.2%
(2,2)	83.8%	83.8%	(4,4)	99.8%	99.8%

Subsequently, we describe the simulation results of the hardware implementation for the proposed BIRA algorithm. We used UMC 0.13μm technology to implement BIRA circuitries for different memory sizes with various redundancy con£gurations. Also, the number of entries of

Table 2: Repair rates of the optimal algorithm and our BIRA algorithm for Case 2.

(r,c)	Opt.	Ours	(r,c)	Opt.	Ours
(0,2)	27.6%	27.6%	(2,4)	95.4%	93.0%
(0,4)	66.4%	66.4%	(3,0)	62.0%	62.0%
(1,0)	12.6%	12.6%	(3,2)	92.2%	91.0%
(1,2)	57.8%	57.8%	(3,4)	98.4%	96.6%
(1,4)	87.4%	85.6%	(4,0)	86.8%	86.8%
(2,0)	44.2%	44.2%	(4,2)	95.6%	94.2%
(2,2)	84.4%	84.4%	(4,4)	99.8%	99.4%

Bitmap in the implemented BIRA circuities is four. Table 3 summarizes the simulation results of BIRA circuitries for three different memory sizes with 1 spare row and 2 spare columns. The second row of the table shows the area cost of the proposed BIRA scheme. Apparently, the area cost is heavily related to the word width since the width of the HSR of Bitmap is the same as the word width. The worst cycle time for the 2048×256-bit memory is only about 2.5ns. We also estimated the analysis time of the proposed BIRA scheme. We simulated 250 memory cores with injected faults for each memory con£gurations. The total analysis time (TAT) for the the 8192×64-bit, 4096×128-bit, and 2048×256-bit memories is 5882 cycles, 6071 cycles, and 6364 cycles, respectively. Assume that the memories are tested by the March-CW [14] test algorithm and the Read or Write operation of the memories consumes 1 clock cycles. The testing time for these three memories is 327680 cycles, 184320 cycles, and 102400 cycles. Therefore, the ratio of the analysis time to the test time for the three memories is about 1.8%, 3.3%, and 6.2%, respectively. Thus the test time overhead caused by the proposed BIRA scheme is small. The average analysis time (AAT) for each detected fault is also shown in the table. The AAT is obtained by the ratio of TAT to the number of total detected faults. The last row of the table shows the critical analysis time (CAT) for the proposed BIRA scheme. The CAT is obtained according to the longest path in the state diagram of the £nite state machine which is implemented to control the algorithm analysis procedure.

Table 3: Simulation results of the BIRA implementation for memories with 1 spare row and 2 spare columns.

Memory size	8192×64	4096×128	2048×256
Area	43169μm^2	71565μm^2	132709μm^2
Cycle Time	1.8ns	2.1ns	2.5ns
TAT	5882 cycles	6071 cycles	6364 cycles
AAT	25.57 cycles	26.4 cycles	27.67 cycles
CAT	39 cycles	39 cycles	39 cycles

Table 4 shows the simulation results of the BIRA implementation for memories with 2 spare rows and 2 spare columns. Compared with Table 3, the difference is that the memories are equipped with 2 spare rows and 2 spare columns. According to Table 4, we see that all the parameters are increased slightly. That is, the proposed BIRA scheme is slightly affected by the number of spare elements.

Finally, we compare the hardware cost for implementing the bitmap in terms of the number of required registers. As

Table 4: Simulation results of the BIRA implementation for memories with 2 spare rows and 2 spare columns.

Memory Size	8192×64	4096×128	2048×256
Area	$43523\mu m^2$	$74880\mu m^2$	$135425\mu m^2$
Cycle Time	1.8ns	2.2ns	2.5ns
TAT	6368 cycles	6242 cycles	6294 cycles
AAT	27.69 cycles	27.14 cycles	27.37 cycles
CAT	39 cycles	39 cycles	39 cycles

Sec. 2 and Sec. 3.2 show, the size of a 2D and a 1D bitmaps can be limited to $W\times(r(c+1)+r)\times(c(r+1)+c)+m(c(r+1)+c)+n(r(c+1)+r)$ and $(r+c)\times(1+n+m+W)$, where W, r, c, n, and m represent the word width, the number of spare rows, the number of spare columns, the number of row address bits, and the number of column address bits of the memory under test, respectively. Table 5 summarizes the comparison results of hardware cost between the 2D and 1D bitmaps for a memory with $n=7$, $m=6$, and $W=32$. As the table shows, the hardware cost of 1D bitmap is much smaller than that of 2D bitmap for various redundancy con£gurations. Note that the hardware cost of 2D bitmap is heavily affected by the redundancy con£gurations. However, the hardware cost of 1D bitmap is slightly affected by the redundancy con£gurations. The ratio of the hardware cost of 1D bitmap to the hardware cost of 2D bitmap is only about 8.6% for $(r,c)=(2,2)$.

Table 5: Comparison of hardware cost for 2D and 1D bitmaps.

$n=7$ and $m=6$ $W=32$	2D Bitmap (A)	1D Bitmap (B)	(B/A)%
(r, c)=(1,1)	327 regs.	92 regs.	28.1%
(r, c)=(1,2)	832 regs.	138 regs.	16.6%
(r, c)=(1,3)	1529 regs.	184 regs.	12.0%
(r, c)=(2,1)	834 regs.	138 regs.	16.6%
(r, c)=(2,2)	2152 regs.	184 regs.	8.6%
(r, c)=(2,3)	3982 regs.	230 regs.	5.8%
(r, c)=(3,1)	1533 regs.	184 regs.	12.0%
(r, c)=(3,2)	3984 regs.	230 regs.	5.8%
(r, c)=(3,3)	7395 regs.	276 regs.	3.7%

5 Conclusions

In this paper we have presented a BIRA scheme for allocating 2D redundancy using 1D local bitmap. The BIRA algorithm is designed based on the memory with global spare rows and local spare columns (i.e., spare columns in the left (right) subarray only can replace the faulty columns in the left (right) subarray). We also have realized the proposed BIRA scheme in hardware for different sizes of memories with various redundancy con£gurations. Experimental results show that the repair rate of the proposed BIRA scheme approximates to that of the optimal analysis algorithm. Also, the ratio of the analysis time to the test time is only about 3.3% for a 4096×128-bit memory tested by the March-CW test algorithm. Analysis results show that the hardware cost of 1D bitmap is much smaller than that of 2D bitmap.

Acknowledgement

This work was supported in part by National Science Council, R. O. C., under Contract NSC 94-2215-E-008-021. We also appreciate that Faraday Technology Corporation (FTC) provides partial support in design.

References

[1] R. Rajsuman, "Design and test of large embedded memories: an overview," *IEEE Design & Test of Computers*, vol. 18, no. 3, pp. 16–27, May 2001.

[2] Y. Zorian, "Embedded memory test & repair: infrastructure IP for SOC yield," in *Proc. Int. Test Conf. (ITC)*, Baltmore, Oct. 2002, pp. 340–349.

[3] S.-Y. Kuo and W. K. Fuchs, "Ef£cient spare allocation in recon£gurable arrays," *IEEE Design & Test of Computers*, vol. 4, no. 1, pp. 24–31, Feb. 1987.

[4] J. R. Day, "A fault-driven, comprehensive redundancy algorithm," in *IEEE Design & Test of Computers*, vol. 2, June 1985, pp. 35–44.

[5] C.-L. Wey and F. Lombardi, "On the repair of redundant RAM's," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 3, pp. 222–231, Mar. 1987.

[6] 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, no. 3, pp. 323–328, Mar. 1990.

[7] 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.

[8] D. Xiaogang, S. M. Reddy, W.-T. Cheng, J. Rayhawk, and N. Mukherjee, "At-speed built-in self-repair analyzer for embedded word-oriented memories," in *International Conference on VLSI Design*, 2004, pp. 895–900.

[9] V. Schober, S. Paul, and O. Picot, "Memory built-in self-repair using redundant words," in *Proc. Int. Test Conf. (ITC)*, Baltimore, Oct. 2001, pp. 995–1001.

[10] J.-F. Li, J.-C. Yeh, R.-F. Huang, and C.-W. Wu, "A built-in self-repair design for RAMs with 2-D redundancies," *IEEE Trans. VLSI Systems*, vol. 13, no. 6, pp. 742–745, June 2005.

[11] 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, no. 4, pp. 386–399, Dec. 2003.

[12] J.-F. Li and C.-W. Wu, "Memory fault diagnosis by syndrome compression," in *Proc. Design, Automation and Test in Europe (DATE)*, Munich, Mar. 2001, pp. 97–101.

[13] R.-F. Huang, J.-F. Li, J.-C. Yeh, and C.-W. Wu, "A simulator for evaluating redundancy analysis algorithms of repairable embedded memories," in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, Isle of Bendor, France, July 2002, pp. 68–73.

[14] C.-F. Wu, C.-T. Huang, and C.-W. Wu, "RAMSES: a fast memory fault simulator," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT)*, Albuquerque, Nov. 1999, pp. 165–173.