High Repair Efficiency BIRA Algorithm with a Line Fault Scheme

Taewoo Han, Woosik Jeong, Youngkyu Park, and Sungho Kang

With the rapid increase occurring in both the capacity and density of memory products, test and repair issues have become highly challenging. Memory repair is an effective and essential methodology for improving memory yield. An SoC utilizes built-in redundancy analysis (BIRA) with built-in self-test for improving memory yield and reliability. This letter proposes a new heuristic algorithm and new hardware architecture for the BIRA scheme. Experimental results indicate that the proposed algorithm shows near-optimal repair efficiency in combination with low area and time overheads.

Keywords: Built-in redundancy analysis (BIRA), memory yield, repair efficiency.

I. Introduction

Previous studies of built-in redundancy analysis (BIRA) methods can be classified into three categories: heuristic algorithms, parallel hardware architectures (used for analyzing all cases of a fault), and search-tree-based algorithms. Among these algorithms, must-repair (MUST) and repair-most (RM) [1] are widely used as simple heuristic algorithms in finding the repair solution. Local RM (LRM) [2] uses a small local bitmap, instead of the existing large fault bitmap, and its RA algorithm is RM. Recently, a heuristic BIRA method with a very small area overhead [3] was researched, but its repair efficiency is lower than RM. Some papers propose all cases of the fault search algorithm to obtain optimal repair efficiency. Comprehensive real-time exhaustive search test and analysis (CRESTA) [4] uses parallel architecture to obtain optimal solutions with a short redundancy-analysis (RA) time. Selected fail count comparison (SFCC) [5] uses a pruned search-tree algorithm for its optimal repair efficiency and low storage requirements. When there are few redundant memories, the area-overhead of CRESTA and the required RA time of SFCC are tolerable. However, when the memory redundancy increases, both of their costs rapidly increase. The heuristic method is a good way to avoid serious cost problems, but the repair efficiency of the existing heuristic RA algorithms is not sufficient. This letter presents a BIRA algorithm with a near-optimal repair efficiency and low area and time costs. The proposed RA algorithm is an improved heuristic based on RM. It can achieve a near-optimal solution in just one step without any search tree or parallel hardware. In addition, high repair efficiency and low area overhead can be achieved. The experimental results illustrate more than 99% repair efficiency with less than half of the area overhead and analysis time of the LRM.

Manuscript received Mar. 24, 2010; revised May 17, 2010; accepted June 3, 2010.

Taewoo Han (email: twhan@soc.yonsei.ac.kr, phone: +82 2 2123 2775), Woosik Jeong (email: woosik.jeong@soc.yonsei.ac.kr), Youngkyu Park (email: hipyk@soc.yonsei.ac.kr), and Sungho Kang (corresponding author, email: shkang@yonsei.ac.kr) are with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Rep. of Korea

doi:10.4218/etrij.10.0210.0097

II. Proposed RA Algorithm

RM is an instinctive heuristic RA algorithm which has high repair efficiency but cannot guarantee an optimal solution for RA. In the RM process, it decides a max fault line for repairing first. When a row line and a column line have the same number of faults, it cannot determine the priority of the row or column first. A defective memory example is shown in Fig. 1. A row-first RM can repair all faults. However, a column-first RM repairs c2 with sc2, and the remaining faults cannot be repaired by the remaining redundant memories. This means that the repair efficiency of RM changes with the row- or column-first repair policy.

This letter uses the cross-point fault character for solving this priority problem. The cross-point fault is defined as a fault in the point that a row fault line and a column fault line are crossed. In Fig. 1, the faults (3, 2) and (4, 2) are the cross-points.

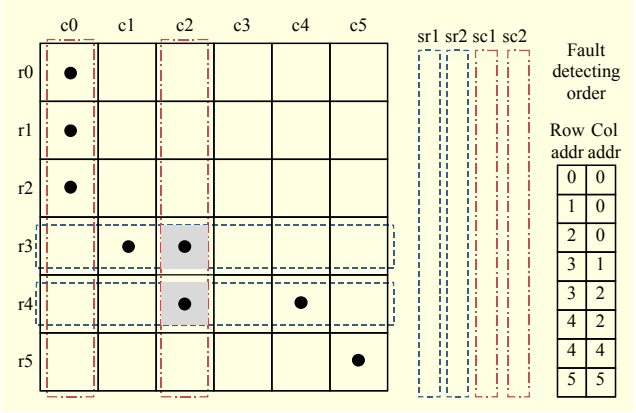

Fig. 1. Example of a memory block with faults.

```
CRM(){
 CRM_FC(){                                  CRM_RA(){
  MUST_CHECK();                               for all fault buffer{
  if(no_must){                                  if(row_flag&column_flag is 1)
   FAULT_BUFFER_WRITE();                          CROSS_POINT_CHECK();}
   for all fault buffer{                      for all fault lines repair{
    if(row address same){                       FIND_MAX();
     row_flag = 1;                              if(count–cross max > 0)
     LOCAL_ROW_TAG_WRITE();}                      REPAIR_LINE(count–cross max );
    if(column address same){                    else if(count max > 0)
     col_flag = 1;                                REPAIR_LINE(count max);}
     LOCAL_COL_TAG_WRITE();}}} }                REPAIR_POINT();              }
                                              }
```

Fig. 2. Pseudo-code of CRM algorithm.

These points can be repaired by either spare row memory or spare column memory. Therefore, we excluded the cross-point from the fault line counting process, which sets the priority of the repair order in RM. After this process, the number of faults in the r3 is 1, c2 is 0, and r4 is 1. The proposed algorithm can repair all faults by substituting r1 and r2 with two redundant row memories. In this letter, this proposed algorithm is called the cross RM (CRM).

Figure 2 shows the pseudo-code of the CRM. A BIRA collects the fault information from the test logic in a dynamic RA mode, and after the memory test process terminates, it analyzes the redundancy within a static RA mode. CRM_FC() represents the fault collection process of CRM in dynamic mode, and CRM_RA() represents the redundancy analysis process of CRM in static mode.

Figure 3 is the new hardware architecture for efficiently executing this algorithm. The proposed hardware architecture is implemented with a fault buffer for collecting faults and local tag registers for storing line fault information. The depth of the fault buffer is $2\times Rs\times Cs$, local row tag is $Rs+Rs\times(Cs/2)$, and the local column tag is $Cs+Cs\times(Rs/2)$. If any three storage areas overflow due to numerous faults, then it is considered an unrepairable case. Therefore, these storage limits automatically offer early termination conditions and reduce the CRM

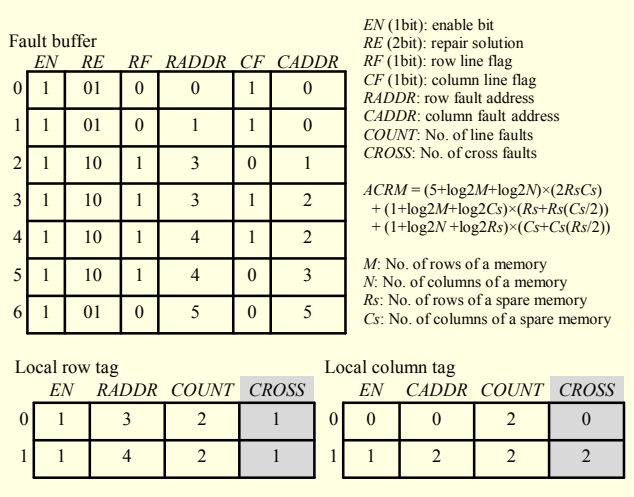

Fig. 3. Hardware architecture of CRM algorithm.

analysis time.

Figure 3 also shows the result of the CRM in the example of Fig. 1. When the fault (0, 0) is detected, it is stored in *fault buffer*$_0$ at FAULT_BUFFER_WRITE. When the fault (1, 0) is detected, it is stored in *fault buffer*$_1$. The column address is the same as the fault buffer $CADDR_0$, so the fault buffers CF_0 and CF_1 are set to 1, and the c0 is stored to the local column tag and $COUNT_0$ is set to 2 at LOCAL_COL_TAG_WRITE. When the fault (2, 0) is detected, the column address is the same as the *fault buffer*$_0$, but the local column tag $COUNT_0$ is already the same as Rs. It is in the MUST condition, and c0 is repaired by the spare column memory. The local column tag EN_0 is reset, and the fault buffers RE_0 and RE_1 are set to 01 at MUST_CHECK. After all faults are detected, if the fault buffer RE_i is 00 and both the RF_i and CF_i are 1, the i-th fault in the fault buffer is a cross-point fault, and local row tag $CROSS_i$ and local column tag $CROSS_i$ are increased by 1 at CROSS_POINT_CHECK. Also, the max $COUNT_i$–$CROSS_i$ value is found in the local tags at FIND_MAX. The *local row tag*$_0$ has the maximum $COUNT_i$–$CROSS_i$ value, so r3 is repaired by the spare row memory at REPAIR_LINE. The local row tag EN_0 is reset, and the fault buffer RE_2, RE_3 is set to 10. After all line faults in the local tags are repaired, the point fault (5, 5) in the fault buffer is repaired by the remaining spare column memory at REPAIR_POINT.

III. Experimental Results

Many previous papers scattered faults randomly, so most faults are positioned as point faults, and most of the repair efficiencies of the heuristic algorithms are shown to be high. In this letter, a Gaussian distribution is used for generating various memory fault cases. The simulations are repeated for 100,000

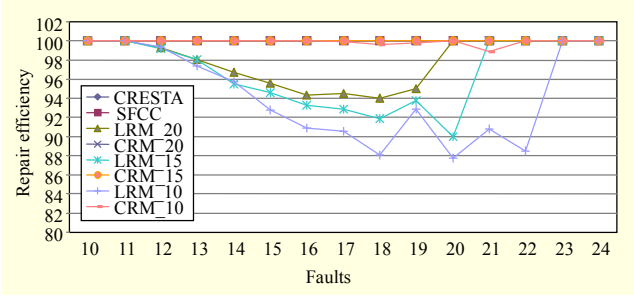

Fig. 4. Repair efficiency (M=1024, N=64, Rs=4, Cs=6).

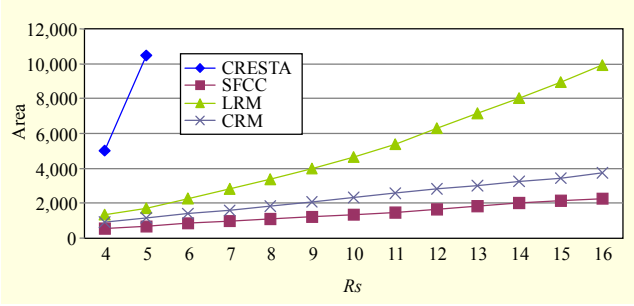

Fig. 5. Area overhead (M=1024, N=64, 4<Rs<16, Cs=4).

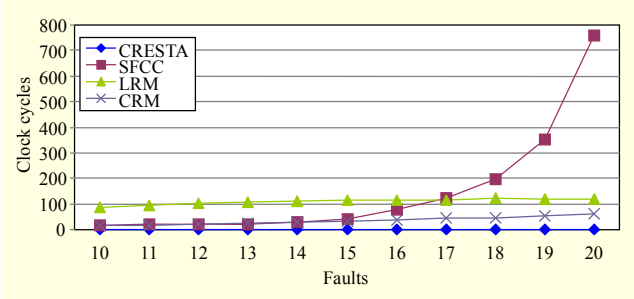

Fig. 6. Clock cycles (M=1024, N=64, Rs=4, Cs=6).

cases, and the same target memories, as described in [2], [5], are used. In Fig. 4, the repair efficiency indicates the number of repaired memories divided by the number of repairable memories. In CRM_15, the 15 refers to the variance in the Gaussian distribution. When the faults are generated massively from 15 to 10, LRM repair efficiency decreases to 88%, but CRM has a repair efficiency of more than 99%. In Fig. 5, the area overhead is estimated to be the required storage for an element as shown in previous papers [2], [5]. The area overhead of CRM is much lower than LRM and increases similarly in comparison to the SFCC. Figure 6 shows the clock cycles for estimating RA times in the static RA process. The clock cycles of heuristic algorithms are increased gradually, but SFCC increases rapidly due to the back-trace within the search-tree processes. Therefore, it is able to confirm that the analysis speed of CRM is much faster than SFCC for many faults and complex cases. CRESTA has optimal repair efficiency with no additional analysis time during the static RA process, but the area overhead increases sharply with highly redundant memories. SFCC guarantees optimal repair efficiency and the lowest area overhead, but the RA time grows rapidly in the case of many faults. Within the heuristic algorithm cases, these have no critical costs, and CRM has lower costs and overhead than LRM. This is because the area overhead and the RA time are part of the 2D local bitmap in LRM, but CRM has only local tag registers and no bitmap. CRM needs additional RA time for collecting cross-point information during the static RA process, but its overall RA time is lower in comparison to the LRMs for the case of small storage sizes with aggressive line fault information usage.

IV. Conclusion

A new RA algorithm, CRM with specialized hardware architecture is proposed in this letter. CRM can achieve higher repair efficiency than RM due to excluding cross-points in the counting process. It shows near-optimal repair efficiency in a one-step process with a low area overhead and a short RA time. The performance of the new RA algorithm is better than the performance of LRM due to its use of line tag information. In comparison with optimal repair efficiency BIRA algorithms (CRESTA, SFCC), the new algorithm has a similar repair efficiency with no serious hardware cost or time cost for those complex cases that have many redundancies or faults. We showed a design for BIRA hardware at a minimum size to be built into embedded memories in SoCs or commodity memories. The existing RM algorithms may be substituted by the proposed heuristic algorithm to achieve higher repair efficiency, a lower area cost, and a faster analysis speed.

References

[1] S.Y. Kuo and W. Kent Fuchs, "Efficient Spare Allocation for Reconfigurable Arrays," *IEEE Des. Test*, vol. 4, no. 1, 1987, pp. 24-31.

[2] C.H. Huang et al., "Built-In Redundancy Analysis for Memory Yield Improvement," *IEEE Trans. Reliab.*, vol. 52, no. 4, Dec. 2003, pp. 386-399.

[3] M. Yang et al., "A Novel BIRA Method with High Repair Efficiency and Small Hardware Overhead," *ETRI J.*, vol. 31, no. 3, June 2009, pp. 339-341.

[4] T. Kawagoe et al., "A Built-In Self-Repair Analyzer (CRESTA) for Embedded DRAMs," *Proc. Int. Test Conf.*, Oct. 2000, pp. 567-574.

[5] W. Jeong et al., "A Fast Built-in Redundancy Analysis for Memories with Optimal Repair Rate Using a Line-Based Search Tree," *IEEE Trans. Very Large Scale Integration*, vol. 17, no. 12, Dec. 2009, pp. 1665-1678.