An Efficient Algorithm for Reconfiguring Shared Spare RRAM

Hung-Yau Lin[1], Hong-Zu Chou[1], Fu-Min Yeh[2], Ing-Yi Chen[3], and Sy-Yen Kuo[1]
[1] *Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan.*
[2] *Chung-Shan Institute of Science and Technology, Taoyuan, Taiwan.*
[3] *Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei, Taiwan.*
hylin@lion.ee.ntu.edu.tw *sykuo@cc.ee.ntu.edu.tw*

Abstract

Redundant rows and columns have been used for years to improve the yield of DRAM fabrication. However, finding a memory repair solution has been proved to be an NP-complete problem. This paper presents an efficient algorithm which is able to find a repair solution for shared spare memory arrays if a solution exists. The remarkable performance of the algorithm can be demonstrated by experimental results.

1. Introduction

Nowadays billions of memory dies are fabricated worldwide each year. Even a small percentage of yield improvement can be quite beneficial. One of the techniques used to improve the yield is to employ redundant rows and columns to replace the rows and/or columns where faulty cells (faults or defects in short) lie [1-4]. A row or a column is called a *line*. It has been proved that memory repair by line deletion is an NP-complete problem [2]. An algorithm which can find a repair solution whenever one exists is called a *perfect* algorithm. A few perfect algorithms have been presented [1-4] but they are not efficient enough.

Spare lines can be cut into segments [4-5]. This allows more flexible use of spare lines. With spare cutting, different segments of a spare line can be used to replace segments of different faulty lines in the original array. The yield may then be improved. Cutting spare lines into segments can make a memory repair problem more complex if spare lines are shared. In this paper, the term SSRRAM denotes shared spare RRAM (Redundant RAM).

Approaches for improving the throughput have been proposed [6]. Many other heuristic algorithms are also proposed to reduce the time of searching for a repair solution [2-4, 7-11]. Though these heuristic algorithms are very efficient, they have a common drawback: they cannot guarantee a solution to be found even if one exists.

This paper presents an algorithm that finds a repair solution for SSRRAM by using BDD (Binary Decision Diagram) [12]. The algorithm is not only perfect but also highly efficient if a good variable ordering is chosen. The remarkable performance of the algorithm can be demonstrated by experimental results.

2. Traditional Algorithms

Previously published perfect algorithms are based on the concept of exhaustive search. A partial solution is chosen for branching into two partial solutions: one uses spare rows and the other uses spare columns. All partial solutions are sorted according to a user-defined cost function. The partial solution with the least cost is chosen for branching. The previous steps repeat until a repair solution is found or no repair solution can be found. The branch-and-bound (B&B) algorithm [2] is a typical representation. The algorithm incorporating the improvements in [3] is called an improved B&B (IB&B) in this paper. IB&B can find a repair solution with less generated records. It is believed that IB&B is faster than B&B.

A huge number of partial solutions will be generated for complex problems. Partial solutions have to be sorted or compared over and over again during the search process. This can be quite time-consuming. The more complex the problem is, the more partial solutions will be generated and the more data will be in each partial solution. The performance of these comparison-based perfect algorithms degrades severely as the problems become more complex.

Memory repair problems can become more complex if spare lines are cut into segments and shared among sub-arrays. A heuristic algorithm is proposed in [4] to speed up the search process. The heuristic algorithm searches each sub-array for the minimal

Proceedings of the IEEE International Conference on Computer Design (ICCD'04)
1063-6404/04 $ 20.00 IEEE

coverage in turn. A minimal coverage is a repair solution that uses the minimal number of spare lines. Consider the problem shown in Figure 1. White rectangles represent normal cells while grayed cells are faulty cells. Spare rows or columns are drawn as thick black lines. The original array is cut into two sub-arrays and the three spare rows are cut into 6 spare row segments. Two spare columns are shared between sub-array A and B. In Figure 1, the minimal coverage for sub-array A is (R4, C3, C4). The minimal coverage for sub-array B is (R8, C7, C8) which also requires two columns. Four spare columns are required but only two of them are provided. Thus, this heuristic algorithm cannot always find a solution even if one exists. In fact, there are 4 solutions to this problem as shown below.

3. The Repair_SSRRAM algorithm

The major difference between our algorithm and other perfect algorithms is that we transform a memory repair problems into Boolean function operations and use BDD to manipulate these Boolean functions. The motivation and goal is to improve the performance by avoiding the huge number of comparison and copy operations inherent in every exhaustive search algorithm. The algorithm is shown in Table 1.

Since improving the variable ordering has been proved to be a NP-complete problem [13], a heuristic algorithm MaxRelated is used to find a relatively good variable ordering. The basic idea of MaxRelated is to keep more related variables closer. Two variables are related if one variable depends on the other to determine the value of the Boolean function. Though MaxRelated seems to be able to find a good variable ordering for most of test cases in this paper, it does not always do.

The defect function in step 2 is the Boolean function that encodes all faults. It can be easily built with the following equation.

$$DF = \prod_{for\_each\_fault\_i} (R_i + C_i)$$

The symbol $\prod$ and + denote the Boolean **AND** and **OR** operations respectively.

The CF_m in step 3 can be built with the following equation and it has to be constructed for each set of spare lines.

$$CF_m = \sum_{i=0}^{i=y_m} C_i^{x_m}$$

, where $C_i^{x_m}$ is a combinatorial function for spare line set m, y_m is the number of spare lines in set m, and x_m is the number of faulty lines in the arrays that spare line set m should repair.

However, it is not practical to enumerate all terms in the combinatorial function. Another method is to put

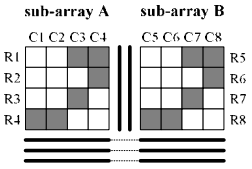

Figure 1. A memory array with shared spare rows.

Table 1. The repair algorithm for SSRRAM.

```
Repair_SSRAM()
{
  1. Choose a variable ordering using MaxRelated
     heuristic algorithm.
  2. Construct the BDD of the defect function DF;
  3. Construct the BDD of the constraint function for
     each set of spare lines. CF_m denotes the constraint
     function for spare line set m.
  4. Construct the final constraint function CF by
     performing Boolean AND operations on all CF_m.
     CF = ∏_{for_each_CF_m} CF_m
  5. Construct the repair function RF by performing the
     Boolean AND operation on DF and CF, that is,
     RF = DF AND CF;
  6. Traverse RF;
  if (RF contains only one BDD node bdd_zero)
       no repair solution can be found;
  else
       report a repair solution;
}
```

all variables of the faulty lines that should be replaced by a spare line set into an array L. The CF_m can then be constructed by setting array index j to 0 in the following placement function PF.

$$PF(L_j, x_m, y_m) = \overline{L_j} \bullet PF(L_{j+1}, x_m - 1, y_m) + L_j \bullet PF(L_{j+1}, x_m - 1, y_m - 1)$$

PF runs recursively until (1) x_m is smaller or equal to y_m or (2) y_m is equal to 0. The following two properties of PF can be applied to these two cases.

$$PF(L, x, y) = true \quad if \quad x \le y.$$
$$PF(L_j, x, 0) = \overline{L_j} \bullet \overline{L_{j+1}} \bullet \overline{L_{j+2}} \bullet \cdots \bullet \overline{L_{x-1}}$$

Step 4 and 5 just perform normal Boolean **AND** operations. After step 5, the repair function RF has been built and it encodes *all* solutions of a memory repair problem. A path from the top variable to the terminal node *bdd_one* is called a *solution path*. All variables in a solution path taking the 1-edge form a solution to the memory repair problem.

Though the graph of a repair function does not need to be generated to find a repair solution, a BDD graph of the RF of Figure 1 is generated and shown in Figure 2. Note that a variable ordering is deliberately chosen to make Figure 2 look better and this variable ordering

is different from the one generated by the MaxRelated heuristic algorithm. There are 4 solution paths in Figure 2:

$$\left\langle \begin{matrix} C3, R2 \\ C4, R3 \end{matrix} \right\rangle, \left\langle \begin{matrix} C7, R6 \\ C8, R7 \end{matrix} \right\rangle, R1, R4, R5, R8.$$

4. Experimental Results

The proposed algorithm is compared to IB&B because IB&B is one of the most efficient perfect algorithms. The CMU BDD library was used to handle the BDD operations. All other files were written in C++ and compiled with g++ 3.3.2. The hardware system had 512 MB of memory and a single Pentium III processor running at 1 GHz. The operating system was a Linux with kernel version 2.4.22.

Figure 3 shows the arrangement of the 4-blocks test cases. Assume (1) S_1 and S_2 have the same number of spare line segments and (2) spare lines segments are shared by adjacent blocks. Table 2 shows some experimental results of 4-blocks test cases. The running time is in seconds. All cases in Table 2 are repairable.

5. Conclusions

The proposed algorithm encodes all repair solutions in a BDD and it is much faster than IB&B if a good variable ordering is chosen. The performance of the proposed algorithm highly depends on the variable ordering. The heuristic algorithm that we use to find a variable ordering seems to be able to find a good variable ordering for most of the test cases. But it is not guaranteed to find a good one. Finding the optimal variable ordering remains an open question.

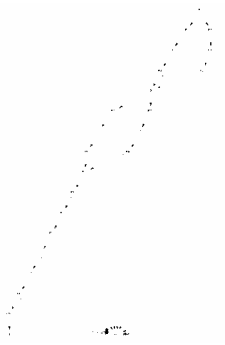

Figure 2. An example BDD of RF.

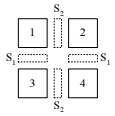

Figure 3. Layout of 4-blocks test cases.

Table 2. Performance comparison.

Case	Sub-array Size	S_1 (S_2)	Faults	IB&B	Ours
1	256 x 256	7	60	0.05	0.01
2	512 x 512	8	80	90.85	0.04
3	512 x 512	12	120	> 3600	0.11
4	512 x 512	14	200	> 3600	0.26
5	512 x 512	16	250	> 3600	4.84
6	512 x 512	18	250	> 3600	8.75
7	512 x 512	20	300	> 3600	10.22
8	512 x 512	22	300	> 3600	2.64
9	512 x 512	22	330	> 3600	1.37
10	512 x 512	22	360	> 3600	2.20
11	512 x 512	25	400	> 3600	6.18
12	1024 x 1024	25	400	> 3600	13.79
13	1024 x 1024	25	400	> 3600	3.42

References

[1] J. R. Day, “A fault-driven comprehensive redundancy algorithm for repair of dynamic RAMs”, *IEEE Design & Test*, Vol. 2, No. 3, 1985, pp. 35-44.

[2] S. Y. Kuo and W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays," *IEEE Design & Test*, Feb. 1987, pp. 24-31.

[3] W. K. Huang, Y. N. Shen, and F. Lombardi, “New approaches for the repairs of memories with redundancy by row/column deletion for yield enhancement”, *IEEE Trans. on Computer-Aided Design*, March 1990, pp. 323-328.

[4] N. Hasan and C. L. Liu, "Minimum fault coverage in reconfigurable arrays," *IEEE Symposium on Fault-Tolerant Computing*, June 1988, pp. 348-353.

[5] Y. N. Shen, N. Park, and F. Lombardi, “Spare cutting approaches for repairing memories”, *IEEE Conference on Computer Design*, Oct. 1996, pp. 106-111.

[6] R. W. Haddad, A. T. Dahbura, and A. B. Sharma, “Increased throughput for the testing and repair of RAMs with redundancy”, *IEEE Trans. on Computers*, Feb. 1991, pp. 154-166.

[7] D. M. Blough and A. Pelc, “A clustered failure model for the memory array reconfiguration problem”, *IEEE Trans. on Computers*, May 1993, pp. 518-528.

[8] C. P. Low and H. W. Leong, “A new class of efficient algorithms for reconfiguration of memory arrays”, *IEEE Trans. on Computers*, May 1996, pp. 614-618.

[9] D. M. Blough, “Performance evaluation of a reconfiguration algorithm for memory arrays containing clustered faults”, *IEEE Trans. on Reliability*, June 1996, pp. 274-284.

[10] C. P. Low and H. W. Leong, “Minimum fault coverage in memory arrays: a fast algorithm and probabilistic analysis”, *IEEE Trans. on Computer-Aided Design*, June 1996, pp. 681-690.

[11] W. Shi and W. K Fuchs, “Probabilistic analysis and algorithms for reconfiguration of memory arrays”, *IEEE Trans. on Computer-Aided Design*, Sep. 1992, pp. 1153-1160.

[12] R. E. Bryant, “Graph-based algorithms for Boolean function manipulation”, *IEEE Trans. on Computers*, Aug. 1986, pp. 677-691.

[13] B. Bollig and I. Wegener, “Improving the variable ordering of OBDDs is NP-complete”, IEEE Trans. on Computers, Sep. 1996, pp. 993-1002.