

Defect-Tolerant Nanocomputing Using Bloom Filters

Gang Wang Wenrui Gong Ryan Kastner
Dept. of ECE, University of California, Santa Barbara
wanggang{gong,kastner}@ece.ucsb.edu

Abstract

We propose a novel defect-tolerant design methodology using Bloom filters for defect mapping for nanoscale computing devices. It is a general approach that can be used for any permanent defects incurred during the manufacturing process. Our redundant design methodology does not rely on a voting strategy, thus it utilizes the device redundancy more effectively than existing approaches. Additionally, our method does not have false-positive in defect identification, i.e. it will not report a defective device as functional. Moreover, it is very space economic and can be programmed to fit different scales and characteristics of the underlying specific nanoscale devices used in the system.

1 Introduction

In order to realize the promise of nanoscale computing, new design methodologies are needed to deal with the high defect rate as we scale down to the nanometer level. It is reported that the defect rate for the self-assembly nanometer devices could be as high as 10% [2], which is much worse compared to the one per billion defect rate found in today's CMOS technology. To tolerate such high degree of faulty devices, many defect-tolerant design methods and architectures have been proposed in past years. Among these solutions, reconfigurable fabrics provide an attractive approach for addressing defect tolerance. It is reported that the reconfiguration is the most effective technique and is able to cope with manufacturing defect rates of the order of 0.01 to 0.1 [3]. The basic idea is to use reconfigurable techniques in a post-manufacturing step to place-and-route the design around the defects. One of the key concepts in this process is a *defect map* [2], in which the locations of the defective devices are stored. However, how to effectively construct defect maps have received very little attention in nanoscale defect-tolerant architectures.

We believe defect maps are fundamental for defect-tolerant computing. This is especially true if we take a practical view of the lifecycle of nanosystems, where de-

fect maps perhaps are the most viable solution to bridge the manufacturing and system design stages. We identify the following criteria for practical implementation of defect maps: 1) It has to be space efficient for a huge number of devices; 2) It has to support random access to be efficient for any real use; and 3) It has to be accurate; at least it should not have false negative, which will wrongfully report a defective device to be functional.

One straight forward way to implement a defect map is to use a flat data structure such as a two-dimensional matrix for the crossbar architecture, where each bit indicate whether a specific nanoscale device is defective or not. This approach satisfies random access and accuracy requirements, however, it is not space efficient. For one billion (10^9) devices, assuming no extra bits are needed to maintain such a structure, it requires 125 MBytes!

2 Defect Tolerance Using Bloom Filters

A Bloom filter is a simple space-efficient randomized data structure for concisely representing a static set in order to support approximate membership queries. The main idea is to achieve space efficiency by allowing false positives with a low probability in the membership query. The Bloom filter has been widely used in database applications, and only recently started receiving wide attention in networking applications. A good survey can be found in [1]. The implementation of Bloom filter is very simple. For any fixed set with size n (in our case, this is the number of nanoscale devices), a Bloom filter makes space-accuracy tradeoffs by adjusting the number of hash functions k , and the total number of bits m for storing the entire table.

Based on the criteria identified in the previous section for a desired defect map implementation, the Bloom filter seems to be a promising solution for nanoscale devices. Clearly it meets the random access requirement by simply performing hashing on the incoming addressing data. As discussed previously, it achieves great space-efficiency with a small one-side (false positive) error rate. Such error rate is justified by the abundance of devices in nanosystems, which will simply result extra devices to be marked as defective.

Moreover, we can deterministically make tradeoffs by adjusting the number of hash functions and the bit size based on the underlying defect rate of the nanosystem and the application requirements. In Table 1, we provide a list of possible tradeoffs for defect rates from 0.01 to 0.1. It can be seen, with modest false positive rates, we are able to shrink the size of the defect maps by 40% to 90% compared with the trivial implementation.

defect rate	m/n	k	map size	false positive	savings
0.01	10	5	12.5	0.009	90.0%
0.02	10	5	25.0	0.009	80.0%
0.04	10	5	50.0	0.009	60.0%
0.06	8	5	60.0	0.022	52.0%
0.08	6	4	60.0	0.056	52.0%
0.10	6	4	75.0	0.056	40.0%

Table 1: Tradeoffs on using Bloom filters as defect maps. (Computation is done for 10^9 devices. Map size is in MBytes.)

3 Defect-Tolerant Design Methodology

Using defect maps based on the Bloom filter, we propose a new defect tolerant design approach as illustrated in Figure 1. In this design, we use the popular crossbar architecture for the nanosystem implementation. Each nanosystem has its own defect map, denoted as BFs in Figure 1, implemented with a reliable memory using a Bloom filter. They save the information regarding the locations of defective devices for each of the nanoscale modules. The nanoscale devices are organized in each module as a two-dimensional matrix in the similar way as FPGA. Each device can be addressed using an address tuple (x, y) with indices to the row and column, respectively. The defect tolerance in the proposed architecture is achieved by providing multiple nanoscale modules that share the same address space. In Figure 1, three such modules are included.

The proposed design for defect tolerant computing differs in principle from tradition approaches such as RMR or NAND-demultiplexing. Instead of using majority voting or cascaded defect control, our arbitrator works as a filter by using information obtained from the defect maps. There is no need to have odd number module duplications in our design. Actually, the same process can be carried without modification with any number of modules in Figure 1. This makes it flexible and more suitable in practice.

Moreover, one benefit of the proposed architecture is that it compensates for the false positive inherited from Bloom filter. One just needs notice that even though for each module the false positive rate is the same p , it is much less likely

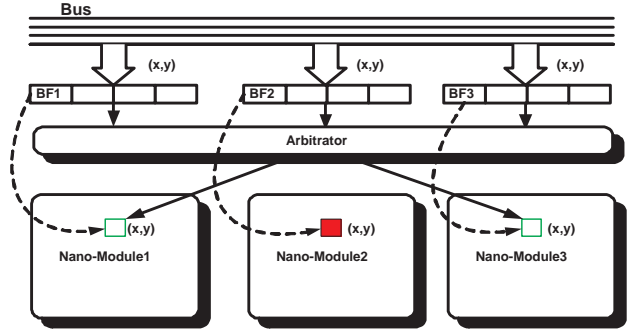


Figure 1: Defect tolerant memory using defect maps based on Bloom filters. (BFs are defect maps associated with corresponding nanoscale modules.)

the all the modules would have defects on exactly same locations. Since the system would work if any chip works for the requested address, the system will have better chance to provide a higher successful rate than using one chip with single Bloom filter.

Furthermore, the proposed design provides a clean interface to existing architectural enhancements. For example, memory caching can be integrated with the above design easily. The cache will remember the most recent visits thus it can speed up the operation by bypassing the query and arbitration steps.

4 Conclusion

We demonstrate that by cleverly applying Bloom filters, we can achieve space efficient defect maps for high density, defective nanosystems with only a small one-sided loss in mapping accuracy. Moreover, we propose a novel defect-tolerant architecture by exploiting the proposed defect map. This approach is more effective in using device redundancy compared to traditional majority voting approaches. To our best knowledge, our work is the first attempt to utilize Bloom filters in defect-tolerant computing for nanoscale systems.

References

- [1] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.
- [2] M. Mishra and S. C. Goldstein. Defect tolerance at the end of the roadmap. In *ITC*, pages 1201–1211, 2003.
- [3] K. Nikolic, A. Sadek, and M. Forshaw. Fault-tolerant techniques for nanocomputers. *Nanotechnology*, 13:357–362, 2002.