# On the Use of Bloom Filters for Defect Maps in Nanocomputing

Gang Wang Wenrui Gong Ryan Kastner
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560
{wanggang, gong, kastner}@ece.ucsb.edu

## ABSTRACT

While the exact manufacturing process for nanoscale computing devices is uncertain, it is abundantly clear that future technology nodes will see an increase in defect rates. Therefore, it is of paramount importance to construct new architectures and design methodologies that can tolerate large numbers of defects. Defect maps are a necessity in the future design flows, and research on their practical construction is essential. In this work, we study the use of Bloom filters as a data structure for defect maps. We show that Bloom filters provide the right tradeoff between accuracy and space-efficiency. In particular, they can help simplify the nanosystem design flow by embedding defect information within the nanosystem delivered by the manufacturers. We develop a novel nanoscale memory design that uses this concept. It does not rely on a voting strategy, and utilizes the device redundancy more effectively than existing approaches.

## Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-Aided Design (CAD)

## General Terms

Algorithms, Design, Theory

## Keywords

Nanotechnology, Defect tolerant, Bloom filter, Defect map

## 1. INTRODUCTION

In research laboratories around the world, prototype nanoscale devices have been constructed using bottom-up chemical assembly instead of top-down lithography that has dominated VLSI manufacturing in the past decades. This progress brings the hope that in the not too distant future, we will be able to create computing devices with unprecedented density ($10^{10}$ devices/cm$^2$) that operate in the THz frequency domain [3].

There are fundamental differences between traditional fabrication methods and the proposed nanoscale methods. Nanoscale manufacturing techniques cannot replicate the complex structures seen in today's ASIC designs. Instead, future techniques favor regular, periodic and programable structures. This makes it natural for nanoscale architectures to have a computation structure similar to those found in current reconfigurable architectures. For instance, the crossbar structure is a simple reconfigurable realization that is prevalent in many proposed nanoscale architectures [2, 4, 8, 16].

It is predicted that nanoscale devices will have high defect rates. These defects can be roughly divided into two classes: (i) permanent defects caused by inherent physics uncertainties in the manufacturing process, and (ii) transient faults due to lower noise tolerance or charge injection at reduced voltage and current levels. Nobody knows the eventual manufacturing failure rate for the nanoscale devices. It is reported that self-assembly nanometer devices could have defect rates as high as 10% [12]. This is much worse compared to the one per billion defect rate found in current CMOS technology. In the near future, it is extremely unlikely that we will be able to produce defect-free nanocomputing devices; current manufacturing practices will not work since high defect rates and the massive number of devices on a chip will result in a yield close to 0%. Therefore, any practical nanoscale computing system must be able to cope with the possibilities of defects.

To address this problem, many defect-tolerant design methods and architectures have been proposed [14]. These methods can be classified into two categories. In the first approach [6, 11, 12], defect tolerance is built into the system utilizing redundancy, e.g. R-fold Module Redundancy (RMR) [13, 15]. Such approaches can handle both permanent defects and transient faults; however, they suffer from low reliability. Alternatively, reconfigurable methods [7, 9, 18] provide an attractive way to address defect tolerance [5]. The basic idea is to use reconfigurable techniques during post-manufacturing design to avoid the defects. It is reported that reconfiguration is the most effective technique, and is able to cope with manufacturing defect rates of the order of 0.01 to 0.1 [13]. It does not, however, effectively handle transient faults.

One of the key concepts in the reconfigurable approach is a *defect map* [12] – a data structure that stores the locations of the defective devices. Effective and efficient defect maps have not been well studied, and are often treated as intermediate component in the design flow [9, 12, 17]. In this work, we show that Bloom filters are an ideal data structure for defect maps. We propose a novel defect-tolerant nanoscale memory architecture by exploiting the Bloom filters. This approach is more effective in using device redundancy compared to traditional majority voting approaches, and can be used effectively with the reconfigurable approaches.

We confine our discussion on permanent defect tolerance only. Though it is possible to extend our method to handle transient faults, it is outside the scope of this work. To our best knowledge, this is the first attempt to utilize Bloom filters in defect-tolerant computing for nanoscale devices.

The paper is organized as follows. In order to better motivate our study, the next section highlights the role of defect maps in the lifecycle of a nanoscale computing system. We discuss the potential impact of the Bloom filter on the nanoscale computing design


*ICCAD'06*, November 5-9, 2006, San Jose, California, USA.

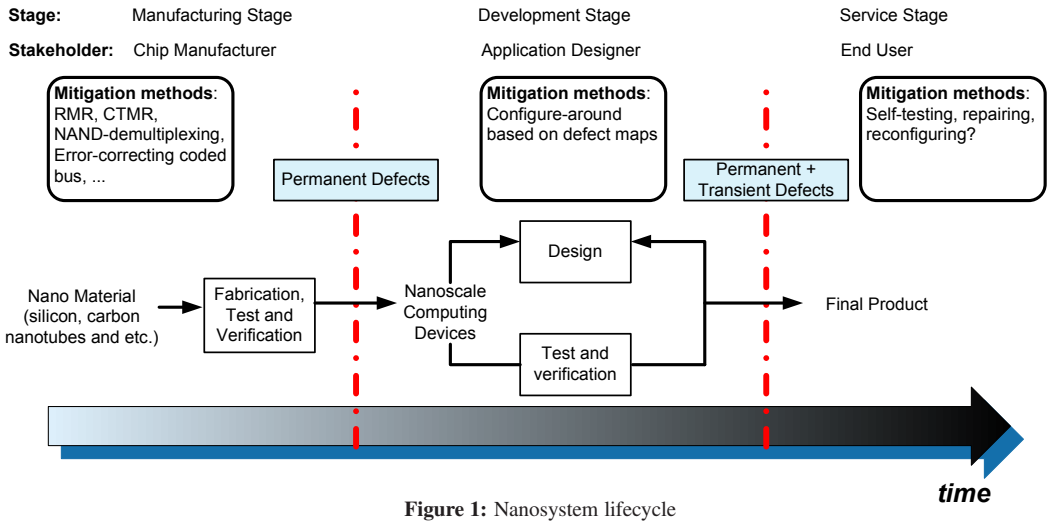


**Figure 1:** Nanosystem lifecycle
(what mitigation methods can be used and how to apply them are greatly impacted by the lifecycle stages).

flow, particularly by embedding defect maps within a computing device. Section 3 describes the fundamentals of Bloom filters, and how they can be used as defect maps. Following that, we present a novel nanoscale memory design using defect maps implemented with Bloom filters. We conclude with Section 5.

## 2. NANOSYSTEM LIFECYCLE AND DEFECT MAPS

A computing device goes through different stages from being manufactured to providing service. These stages greatly constrain the defect-tolerant paradigms that can be practically realized. To understand the related issues, we provide a view of nanosystem lifecycle in Figure 1. The lifecycle is roughly partitioned into three stages: manufacturing, development, and service; each of the stages has a major stakeholder associated with it, the chip manufacturer, the application designer, and the end user, respectively.

Defect mitigation methods available at each stage are limited. In the manufacturing stage, only methods such as RMR and NAND-demultiplexing [6] are feasible as application information is not available then. On the other hand, at the development stage the application designer typically goes through an iterative design, test and verification process to finalize the product. Most likely, the designer will purchase a fabricated programmable nanoscale computing device from the manufacturer, and will be limited from performing circuit level mitigation. Here defect tolerance based on configuration is the primary approach. The biggest limitation at this stage is perhaps the time and money needed to perform full scale defect detection. This is caused by:

- The prohibitive cost of buying and operating test equipment;
- The immense amount of time required to perform defect detection.
- The limitation on expertise. Application designers deal with the underlying computing devices using higher level abstract models, and they may not have the device level knowledge;

Because of these limitations, ideally, the manufacturer shall provide defect information to the designer. For the designer, it is also desired to have a system architecture and a set of design tools that mask these defects as much as possible and handles them transparently in the design process. This allows the designer to focus on application specific issues.

With this view, some existing defect-tolerant methods based on reconfiguration may be difficult to apply in future practice. For example, the framework proposed in [7, 9] requires a deeply coupled approach of nanoscale defect handling and a high level behavior description of the application represented in a control data flow graph. This requires close collaboration and knowledge sharing between application designers and manufacturers.

Alternatively, we believe a more practical approach is to use a defect map as the bridge between the manufacturing and the design stages in order to address defect tolerance problem. The locations of defective devices are recorded in the defect map, which is generated once during the manufacturer's testing and validation process. Providing this map helps to enable a transparent design stage where the defects can be handled automatically by the design tools.

More attractive, with a space-efficient defect map, it is possible to embed the defect map within a nanoscale computing system and deliver them together. We envision that such systems are realized with hybrid architectures similar to the ones developed in [4, 16], where defect maps are stored with reliable CMOS technology.

Such on-chip defect maps are especially useful during the development stage. With system specific defect maps on the same chip, the most important benefit introduced by embedding the defect maps is to enable a better defect tolerant design flow. Without providing such maps, the final mapping of a design is proceeded by a defect test and diagnosis process on each computing device in order to obtain the critical defect map information, and configure the design around these defects using a mapping tool. This is prohibitive in practice, as we described previously. One could argue that the defect map could be provided as off-the-chip files by the manufacturers. This is definitely a possible solution in labs with sophisticated database support to keep track of the individual defect map for each device. However, it is hardly practical for systems that require field reconfigurability after product deployment.

On the other hand, by using a space-efficient defect map embedded on the same chip, the application design flow could be greatly simplified. It provides a system specific bridge between the manufacturing and the development stage. Existing design tools enhanced with querying capability of the defect map can be applied directly by the application designer. This saves the trouble for the designer to examine defects in a system-by-system manner, which is costly, time consuming and distractive. Also it eliminates the requirement to purchase expensive setups in order to perform such tests. It is foreseeable such an enhancement will greatly improve the productivity of the design process and reduce cost incurred for managing this information through the lifecycle of the nanosystem.

Moreover, having a embedded defect map in the deployed sys-

tem is beneficial for self-testing and reconfiguration in the field, which makes field reconfigurability more feasible. For example, the reconfiguration approaches suggested in [12] and [9] both need to locate a valid area on the nanosystem and configure the nanoscale devices in this area as the initial testing circuitry. With an accompanying defect map, we can speed up this process and also guide the testing process with previously obtained defect information.

In order for defect maps to be an integral part of the design flow, its implementation must meet certain requirements. We identify the following criteria for a practical implementation of defect maps:

1. It must be space efficient for a huge number of devices.
2. It must support random access to be efficient for any real use.
3. It must be accurate;it should not allow false negatives, which will wrongfully report a defective device to be functional.

One straight forward way is to use a flat data structure such as a two-dimensional matrix for the crossbar architecture, where each bit indicates whether a specific nanoscale device is defective or not. This approach satisfies the 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! One may use a compression technique such as the Run-Length Encoding to reduce the space usage. However, this does not meet the random access requirement.

Unfortunately, the implementation of defect maps has received little attention in previous literature. Some possible reasons for the lack of study in this area are: 1) it is commonly believed that the size of the defect maps for the nanoscale fabric would be prohibitively large; and 2) it is commonly believed that the entire design flow has to be customized per nanochip, which results in a radical shift from conventional design methods where all logic and physical design tools have to redeveloped [17]. In the following, we attempt to demonstrate that these are not necessarily true and it is possible to construct space-efficient defect maps for high density and high defect rate (a billion devices, 10% defect rate) nanoscale chips with only modest changes in existing tool chains.

# 3. DEFECT MAPS USING BLOOM FILTERS

A Bloom filter is a simple space-efficient randomized data structure for representing a static set in order to support approximate membership queries [1]. The main idea is to achieve space efficiency by allowing false positives with a low probability in the membership query. That is, for a query on element $x$, when the Bloom filter says that $x$ is not in the set, the answer is always correct. However, when it says $x$ belongs to the set, it may be wrong with a small probability. This is extremely useful when most elements are *not* in the set of interest. More importantly, it offers flexible ways to adjust the tradeoff between the space and the rate of false positives. Bloom filters have been widely used in database software, and have recently started receiving intensive attention in networking applications.

The implementation of a Bloom filter is quite simple. It represents a set $A = a_1, a_2, \ldots, a_n$ of $n$ stored elements as a vector $v$ of $m$ bits, where initially all the bits are set to 0. Associated with the Bloom filter are $k$ independent hash functions $H_1, H_2, \ldots, H_k$, each with a range of $1, \ldots, m$. For every element $a$ to be stored, the bits at positions $H_1(a), H_2(a), \ldots, H_k(a)$ in $v$ are set to 1. A particular bit might be set to 1 multiple times but only the first one is effective. Later, to determine whether $x$ is in the set, the bits at positions $H_1(x), H_2(x), \ldots, H_k(x)$ are checked. If any of them is 0, then certainly $x$ is not in the set $A$. This guarantees that no false negative will be returned. However, a false positive is possible when the bits of $x$ happen to be all set by previously stored elements. For a fixed $n$, the parameters $k$ and $m$ can be adjusted to tradeoff between the size of the filter and the false positive probability. Generally, the false positive probability decreases as $k$ increases or $m/n$ increases.

We can use a Bloom filter to store all the addresses of the defective devices. Clearly it meets the random access requirement by simply performing hashing on the incoming locations. It achieves great space-efficiency with a small one-side (false positive) error rate. The error rate is justified by the abundance of devices in nanoscale computing systems, which simply result in functional devices to be marked as defective. Moreover, we can deterministically make tradeoffs by adjusting the number of hash functions, the bit size derived from the defect rate of the specific nanoscale realization, and the application requirements.

| 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% |
| 0.03 | 10 | 5 | 37.5 | 0.009 | 70.0% |
| 0.04 | 10 | 5 | 50.0 | 0.009 | 60.0% |
| 0.05 | 8 | 5 | 50.0 | 0.022 | 60.0% |
| 0.06 | 8 | 5 | 60.0 | 0.022 | 52.0% |
| 0.07 | 7 | 5 | 61.2 | 0.035 | 51.0% |
| 0.08 | 6 | 4 | 60.0 | 0.056 | 52.0% |
| 0.09 | 6 | 4 | 67.5 | 0.056 | 46.0% |
| 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. )

Consider a nanosystem that contains $10^9$ devices with a defect rate of 5%. This implies that $5 \times 10^7$ devices are defective. With a $m/n$ ratio of 8 bits, and 5 hash functions, we can construct a defect map using 50 Mbytes with a false positive error of 2.17%. This is a 60% reduction on the storage space required compared with the flat implementation. 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, flat implementation. This makes it more feasible to be embedded within a nanosystem.

One may question the effectiveness of using multiple hashing functions and whether it is practical. This concern is justified as most hash functions involve modulation. Furthermore, hashing is the only non-trivial operation that the Bloom filter requires, therefore any optimization on this step will translate into great gains. Recently, Kirsch *et al.* reported [10] progress on this topic. Their key idea is borrowed from the classic hashing literature that uses two hash functions $h_1(x)$ and $h_2(x)$ to simulate additional hash functions $g_i$ in the form $g_i(x) = h_1(x) + ih_2(x)$, where $i$ is hash function index.

By leveraging this method, they have shown that a Bloom filter can be implemented using only two pseudorandom hash functions $h_1$ and $h_2$ without any noticeable increase in the asymptotic false positive probability for Bloom filters of fixed size with reasonable parameters. This is a significant result. It indicates that for a single Bloom filter with $k$ hash functions, we only need to compute two modulations and the remaining hashes can be quickly calculated using addition/subtraction recursively on these two values. This result may greatly simplify hardware implementations.

# 4. REDUNDANT NANOSCALE MEMORY

A nanoscale memory architecture based on defect maps using Bloom filters is illustrated in Figure 2. In this design, we use the

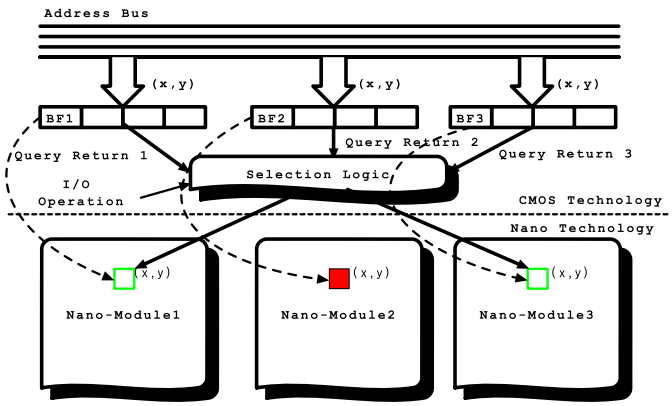


**Figure 2:** Defect tolerant memory using defect maps based on Bloom filters. (BFs are defect maps associated with nanoscale modules.)

popular crossbar realization [2, 16, 4] for the nanoscale implementation. Each memory module has its own defect map, denoted as BFs in Figure 2, implemented using a Bloom filter with reliable CMOS technology. They store the location information of defective devices in each of the nanoscale modules. The nanoscale memory cells in each module are organized as a two-dimensional matrix in the similar way as FPGA. Each cell can be addressed using an address tuple $(x, y)$ with indices of 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 2, three such modules are included.

The address bus provides a transparent view for the external application. It hides the nanoscale redundancy. Assuming that the each modules has $2^n$ memory cells, the bus will have $n$ address lines to provide full access to any cell in every nanoscale module. To the application, the internal redundancy is hidden and the whole architecture is treated as a memory chip with $2^n$ cells.

For each read/write operation on address $(x, y)$, we first query the Bloom filters of all the nanoscale modules using the incoming address. It will provide a true/false report (with false positive) for each module on whether the cell $(x, y)$ is defective. Then, simple selection circuitry is used to allow or disallow the read/write operation on a specific module. For example, in Figure 2, the nanoscale device for location $(x, y)$ in *Nano-Module2* is reported to be defective (indicated with a solid box), while the other two with the same address are normal. For a write operation, the selection logic will perform memory write at location $(x, y)$ for both *Nano-Module1* and *Nano-Module3* so to keep maximal redundancy. For a read operation, the content of one of the working cells, either from *Nano-Module1* or *Nano-Module3*, is picked randomly as the output.

The proposed design differs in principle from tradition approaches using NAND-demutiplexing or RMR, though they may have the same number of nanoscale cells. Instead of using majority voting or cascaded defect control, our design 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, if we use two chip or any other number of modules in Figure 2, the same process can be carried without modification. This makes it flexible and more suitable in practice.

Moreover, the proposed architecture compensates 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 the all the modules would have defects on exactly same locations. Since the system would work if any chip works for the requested address, it has better chance to provide a higher successful rate than using one chip with single Bloom filter. For example, compare the above design with a classic triple redundancy approach using reliable majority voting gates, for a defect rate of 5%, we have a failure rate for a given address at $(0.05)^3 \approx 0.01\%$ while the later is at: $1 - \binom{2}{3} (0.95)^2(0.05) - (0.95)^3 \approx 0.73\%$. In this case, our design is about two orders of magnitude better in operation failure rate.

# 5. CONCLUSION

By connecting existing defect tolerant methods with the lifecycle of nanosystems, we suggest that the defect map is an important concept for bridging manufacturing and development stages. We demonstrate that it is possible to construct space-efficient, randomly accessible defect maps for high density and high defect rate nanosystems with very small sacrifice in accuracy using Bloom filters.To our best knowledge, this is the first time that anyone has suggested using the Bloom filter in nanosystem architecture. Furthermore, we propose to embed a nanocomputing fabric and it's associated defect map on the same chip, and illustrate its advantage over the traditional design flow. Moreover, we present a new defect tolerant memory architecture combining defect maps with module level redundancy, and show that it is advantageous over traditional majority voting approaches.

# 6. REFERENCES


[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[2] Y. Chen, G.-Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14:462–468, 2003.

[3] R. Compano, L. Molenkamp, and D. J. P. (eds). Technology roadmap for nanoelectronics. *European Commission IST Programme - Future and Emerging Technologies*, 2000.

[4] A. DeHon and K. K. Likharev. Hybrid CMOS/Nanoelectronic digital circuits: Devices, architectures, and design automation. In *ICCAD'05*, pages 375–382, November 2005.

[5] S. C. Goldstein. The impact of the nanoscale on computing systems. In *ICCAD'05*, pages 655–661, November 2005.

[6] J. Han and P. Jonker. A defect- and fault-tolerant architecture for nanocomputers. *Nanotechnology*, 14:224–230, 2003.

[7] C. He, M. F. Jacome, and G. de Veciana. Scalable defect mapping and configuration of memory-based nanofabrics. In *IEEE HLDVT Workshop*, 2005.

[8] T. Hogg and G. Snider. Defect-tolerant logic with nanoscale crossbar circuits. Technical report, HP Labs, 2004.

[9] M. Jacome, C. He, G. de Veciana, and S. Bijansky. Defect tolerant probabilistic design paradigm for nanotechnologies. In *DAC '04*, pages 596–601, New York, NY, USA, 2004. ACM Press.

[10] A. Kirsch and M. Mitzenmacher. Building a better bloom filter. Technical Report TR-02-05, Division of Engineering and Applied Sciences, Harvard University, 2005.

[11] P. J. Kuekes, W. Robinett, G. Seroussi, and R. S. Williams. Defect-tolerant interconnect to nanoelectronic circuits: internally redundant demultiplexers based on error-correcting codes. *Nanotechnology*, 16:869–882, 2005.

[12] M. Mishra and S. C. Goldstein. Defect tolerance at the end of the roadmap. In *ITC*, pages 1201–1211, 2003.

[13] K. Nikolic, A. Sadek, and M. Forshaw. Fault-tolerant techniques for nanocomputers. *Nanotechnology*, 13:357–362, 2002.

[14] G. Norman, D. Parker, M. Z. Kwiatkowska, and S. K. Shukla. Evaluating the reliability of defect-tolerant architectures for nanotechnology with probabilistic model checking. In *VLSI Design*, pages 907–912, 2004.

[15] D. Siewiorek and R. Swarz. *Reliable Computer Systems Design and Evaluation*. Digital Press, Bedford, MA, 1992.

[16] G. Snider, P. Kuekes, and R. S. Williams. CMOS-like logic in defective, nanoscale crossbars. *Nanotechnology*, 15:881–891, 2004.

[17] M. Tahoori. A mapping algorithm for defect tolerance of reconfigurable nano-architectures. In *ICCAD'05*, pages 668–672, 2005.

[18] M. B. Tahoori. Defects, yield, and design in sublithographic nano-electronics. In *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems.*, pages 3–11, 2005.