Missouri University of Science and Technology

Scholars' Mine

Electrical and Computer Engineering Faculty Research & Creative Works

Electrical and Computer Engineering

01 Jul 2006

Cost-Driven Repair of a Nanowire Crossbar Architecture

Yadunandana Yellambalase

Shanrui Zhang

Minsu Choi
Missouri University of Science and Technology, choim@mst.edu

Nohpill Park

et. al. For a complete list of authors, see https://scholarsmine.mst.edu/ele_comeng_facwork/1518

Recommended Citation

Cost-Driven Repair of a Nanowire Crossbar Architecture

Yadunandana Yellambalase[1], Shanrui Zhang[1], Minsu Choi[1], Nohpill Park[2] and Fabrizio Lombardi[3]

[1] Dept of ECE, University of Missouri-Rolla, Rolla, MO 65409-0040, USA

{ypymy9, sz2k2, choim}@umr.edu

[2] Dept of CS, Oklahoma State University, Stillwater, OK 74078, USA,

npark@cs.okstate.edu

[3]Dept of ECE, Northeastern University, Boston, MA 02115,

lombardi@ece.neu.edu

Abstract— The recent development of nanoscale materials and assembly techniques has resulting in the manufacturing of high-density computational systems. These systems consist of nanometer-scale elements and are likely to have many manufacturing imperfections (defects); thus, defect-tolerance is considered as one of the most some algorithms for repairing defective crosspoints in a nanoscale crossbar architecture; furthermore we estimate the efficiency and cost-effectiveness of each algorithm. Also, for a given design and manufacturing environment, we propose a cost-driven method to find a balanced solution by which figures of merit such as area, repair time and reconfiguration cost can be taken into account. Probabilistic parameters are utilized in the proposed cost-driven method for added flexibility.

I. INTRODUCTION

Recently, logic devices have been proposed based on nanoscale components such as carbon nanotubes (CNTs) and silicon nanowires (SiNWs); computing architectures have also been proposed using these devices as primitive building blocks. Unlike CMOS, chemically-assembled nanoscale devices (such as CNTs and SiNWs) are unlikely to be used to construct complex aperiodic (non homogeneous) structures [1], [2].

One of the most promising computational nanotechnologies is the so-called crossbar-based architecture [1], [2], [3], [4], [5], [6]; this is a two-dimensional array (nanoarray) formed by the intersection of two orthogonal sets of parallel and uniformly-spaced nanometer-sized wires, such as carbon nanotubes (CNTs) and silicon nanowires (SiNWs). Experiments have shown that such nanoscale wires can be aligned to construct an array with nanometer-scale spacing using directed self-assembly.

Defects and faults are significant problems for nanoscale integration. Although high density (such as 10^{12} devices per cm^2 [3]) well beyond the capabilities of scaled CMOS can be realized in nanoscale technology, these systems (with nanometer-scale elements) are likely to have many imperfections in diameter, pitch, length, alignment, etc,. Thus, computing or storage systems designed based on conventional defect & fault models are not practical [1]. It is anticipated that CAEN (Chemically-Assembled Electronic Nanotechnology) such as NanoFabric will have a significantly higher defect density than CMOS, as high as 10% [3], [4], [5]. Fabrication could potentially be very inexpensive provided an efficient chemical self-assembly could be attained; however this will still require laborious testing, diagnosis, repair and reconfiguration with significant costs. In this paper, repair algorithms are investigated and their performance is evaluated in terms of reconfiguration time, area and overall utilization of programmable crosspoints. Subsequently, these performance metrics are combined with various cost parameters; a method for finding the best combination of repair and overall cost for a given parameter set is proposed and validated.

II. REPAIRING A CROSSBAR ARCHITECTURE

A nanowire crossbar architecture supports flexible utilization of crosspoint devices through reconfiguration for defect rates as high as 10%. The internal lines in the crossbars are completely interchangeable and the switch block can provide connections between input and output signals of adjacent crossbar blocks. It is then possible to utilize this flexibility and reconfigurability to tolerate defective crosspoints.

Few papers have discussed testing of a nanowire crossbar architecture [2] [5]. An external tester is usually employed. The crossbar is programmed using an internal tester that can then be used to test the remaining parts of the architecture. Except for the external tester, this process can be basically viewed as a built-in self test method. Also, testing can be performed in parallel such that the test time can be reduced. For nanometer devices, the assumed defect model is substantially different from CMOS. Defects can be categorized as follows: 1) Defects in programmable crosspoints, 2) Defects in nanowires [6]. Nanowires with a short or a break can be easily detected and screened. Physically, defects in programmable crosspoints are due to the structure of the junctions, that are bistable molecules between two layers of nanowires. Reprogrammability of a crosspoint originates from the bistable property of the molecules located in the crosspoint area. If there are not enough molecules at a crosspoint, then the junction may not be able to be programmed to a "closed" state, or the "closed" state may have a higher resistance than the threshold to enable the whole crossbar to operate properly. If a crosspoint cannot

be programmed to an "open" state (i.e. the two crossing nanowires are always connected, as equivalent to a short), then a nanowire defect rather than a junction defect is said to occur. Those crosspoints that cannot be programmed into a "closed" state, but they can be programmed into an "open" state are referred to as *non-programmable crosspoints*. Although non-programmable crosspoints are defective, they do not affect the other crosspoints on the same rows and columns. So, when functions are mapped to the crossbar devices, the non-programmable crosspoints are placed to the unused locations, thus providing flexibility in logic mapping.

To realize a function set I on the crossbar, a logic synthesizer generates a netlist that allocates some of the nanowires as inputs, and some as outputs; it also indicates the crosspoints that must be set to the "close" state. A $M \times M$ matrix I is used to represent the netlist. Each row (column) represents one of the logic rows (columns). If the node value is 1, this means the corresponding crosspoint must be programmed to a "closed" state; if the node value is 0, the crosspoint is not used. Due to the reconfigurability of the crossbar architecture, the order of the rows (and columns) can be rearranged. After testing, a defect map that indicates the locations of the defective crosspoints, is constructed. A $N \times N$ matrix D is used to represent the defect map. Both I and D are assumed to be symmetric matrices (only for simplicity in simulation). If there is a non-programmable crosspoint at a location, 1 is used for the corresponding node, otherwise a 0 is used. For a row or column, a successful matching from a function set I to the physical array D can be generated as follows:

1) Every node that needs to be programmed to a "closed" state, must fall into a non-defective crosspoint.
2) Every node that is "unused" can fall into either a non-defective crosspoint or a non-programmable crosspoint.

If matrices are used for modeling purposes, repair consists of finding an algorithm that successfully assigns all the 1s from matrix I to fall into the 0 nodes of the matrix D. The 0s from I are not important as far as which node in D must be matched. Therefore, the 1 nodes in I cannot overlap with the 1 nodes in D. This is accomplishing by $ANDing$ one row or column from I and one row or column from D. If the result is all 0s, then a successful matching is said to have occurred. Otherwise, another row must be selected to find a successful matching. Also, it is desirable to achieve minimal area and time overheads in the execution of the repair algorithm.

III. DEFECT MATRIX GENERATION PROCEDURE

The defect layout of a nanowire crossbar can be represented as a matrix. A non-programmable cross-point at a node location is identified by a 1, while a programmable cross-point is marked by a 0. In this paper, the defect maps were randomly generated as clusters with a negative binomial distribution [9]. The probability of having a defect at a cross-point during a time interval Δt in the manufacturing process is given by $p(\Delta t | k, l_1, l_2 \cdots l_n) = c(x, y) + bk + \sum_{l=0}^{n} b_i l_i$, where $c(x, y)$ is the susceptibility function, k is the number of defects already present in the area, the index i pertains

to the adjacent and neighboring cross-points, n indicates the number of neighboring cross-points considered, b is the global cluster parameter, b_i is the local cluster parameter and l_i is the number of defects that occur in the neighboring area. A constant susceptibility parameter C has been used throughout this paper.

The values of b, b_i, l_i, n and k were set to obtain the desired density and distribution. The resulting defect pattern was generated with a 150×150 matrix, for $C = 0.005$, $b = 0.00001$, $b_i = 0.02$, and $l_i =$ number of defects already present in the surrounding 3×3 matrix of a cross-point.

IV. REPAIR ALGORITHMS

For repair, a row-wise or column-wise algorithm (1-D) is trivial, because a greedy algorithm always results in the optimal solution. However, for the two-dimensional (2-D) case (that is similar to a two-dimensional memory), only a brute-force algorithm can always guarantee an optimal solution as this problem is NP-complete [8]. So, algorithms that find sub-optimal solutions with a reasonable execution time, are usually pursued. For repairing a crossbar architecture, the following three algorithms have been evaluated: 1) 1-D greedy repair, 2) 2-D sequential shuffle algorithm (i.e., search row-wise then column-wise, or vice versa), 3) 2-D repair with redundant input columns (if available). Simulation of these algorithms was performed using Matlab. The following assumptions were used throughout the simulation: 1) A defect rate of 10% is used, 2) A so-called function usage rate P_f is used. When a $M \times M$ matrix I is generated, each crosspoint has a probability P_f probability to be 1, 3) It is also assumed that $N \geq M$. Two metrics are defined for the repair algorithms: 1) *Utilization*: The number of columns or rows that have been successfully matched to the physical array divided by the physical array size N, 2) *Coverage*: The number of columns or rows that have been successfully matched to the physical array divided by the given function size M.

V. COST-DRIVEN REPAIR

In this section, three figures relating to overhead as for repair time, area and switch block reconfiguration are analyzed and discussed as result of the crossbar repair process.

A. Repair Time

The advantages of 2-D over 1-D repair algorithms are in both Utilization and Coverage. However, 2-D algorithms are computationally more intensive than 1-D algorithms and result in a larger repair time. Repair time accounts for a significant cost parameter in manufacturing (other cost parameters are introduced and evaluated in the following section). They can be combined to establish a balanced repair solution.

The execution time of each algorithm was measured by $Matlab$ and used for time overhead. A Pentium M 1.9Ghz processor and a 1GB RAM were utilized. A method to calculate the time overhead will be discussed in Section VI.
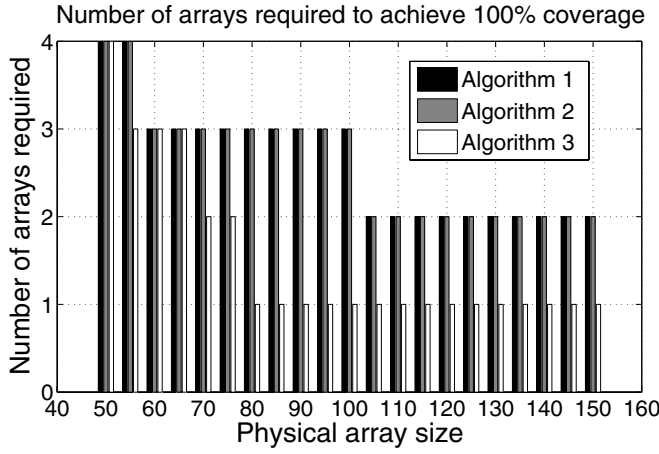
Fig. 1. Number of logic blocks needed to implement a function set I.

Fig. 2. Plots of the repair cost for $M = 50$ and $P_f = 70\%$ case.

B. Area

After completing the design phase, the size of the function set M can be determined as the manufacturing process defines the real array size N. Each array size N will result in different overhead for $Utilization$ and area. Initially, the area to implement a function set for a size M and a physical array size N must be found. From the simulation results shown in Figure 1, to achieve 100% $Coverage$, multiple crossbar blocks are required for implementing the function set. First, the number of arrays required for a combination of M and N must be found, i.e. the average number of required arrays is given by $N_a = \lceil Coverage^{-1}\rceil$. N_a is used as a starting value and increased or decreased until 100% $Coverage$ (for the smallest value N_a) is achieved. The detailed process can be described as follows: initially, try to match I to N_a physical arrays. If 100% coverage is achieved, then decrease N_a by 1 otherwise increase N_a by 1 until 100% coverage is obtained. Then, obtain N_f as the minimal number of arrays for implementing I. $Utilization$ is calculated again after achieving 100% coverage and the unused area as $N_f \times N^2 - M^2$. Simulation was conducted and, among the three considered algorithms, *algorithm 3* had the best performance in terms of area.

C. Switch Block Reconfiguration

Crossbar blocks are interconnected via switch blocks. Programmable crosspoints in switch blocks provide flexible routing to the crossbar architecture. The programming process of the crosspoints also involves a reconfiguration time. So the switch block programming cost should be also taken into consideration. It is assumed that switch blocks have programmed crosspoints initially arranged in a diagonal pattern. After completing the repair process by matching and reconfiguring the crossbar block, the connectivity between two crossbar blocks may be changed. So, switch blocks should be reprogrammed. According to [6], the programming process takes place one crosspoint at a time. Within a switch block, programming cannot be executed in parallel. Thus, the number
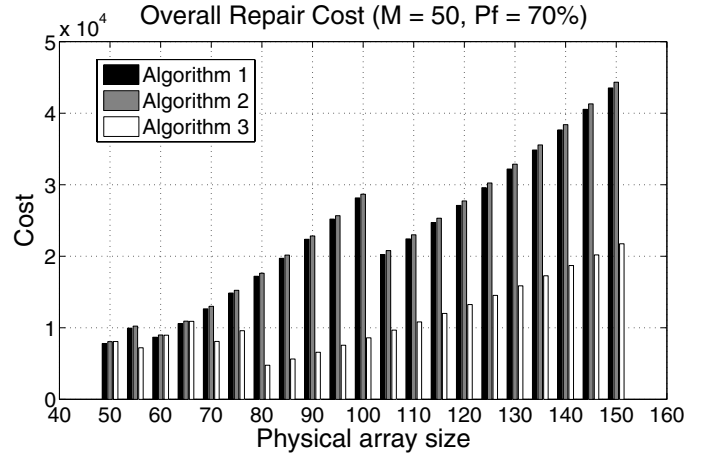
of crosspoints not falling on the initial diagonal pattern (i.e. the number of crosspoints that must be reconfigured) determines the time overhead of the programming process. The number of crosspoints that are required by the three repair algorithms was calculated and considered in the overall cost-driven technique as described in the next section.

VI. Cost-Driven Repair

After analyzing the time overhead, the area and the switch block reconfiguration overheads (as described in a previous section) have been considered as cost parameters for finding the solution that has a balanced repair performance. The approach proposed in [7] can be utilized to balance performance and overall cost of the repair process. The cost of the area overhead can be calculated as follows: obtain the normalized manufacturing cost value (e.g., in dollar) per one device. For a mature fabrication plant, this cost can be empirically estimated. Then divide the fabrication cost for a single device by the total number of crosspoints in the crossbar blocks, such that the unit cost per crosspoint is found. Let α denote the unit area cost-parameter. Finally, multiply the unit cost for a crosspoint α by the number of unutilized crosspoints so that it is possible to find the cost of the area overhead in the normalized cost value. The time overhead cost can be calculated as follows: find the ratio η for the implementation time and the simulation time using the proposed simulator. This requires to know the operational frequency of the testers, the machine touching down time, the number of I/O pins on the interface device, etc. Then for a manufacturing process, find the cost of machine usage, power consumption and related features per unit time. Let β denote the cost-parameter per unit manufacturing time multiplied by η. So, multiply the simulation time by β to get the normalized cost of the time overhead.

For the cost of reprogramming a switch block, an approach similar to the time overhead analysis is applicable. Moreover it is required to find the cost per unit time for manufacturing and the elapsed time to reconfigure one crosspoint (that is also
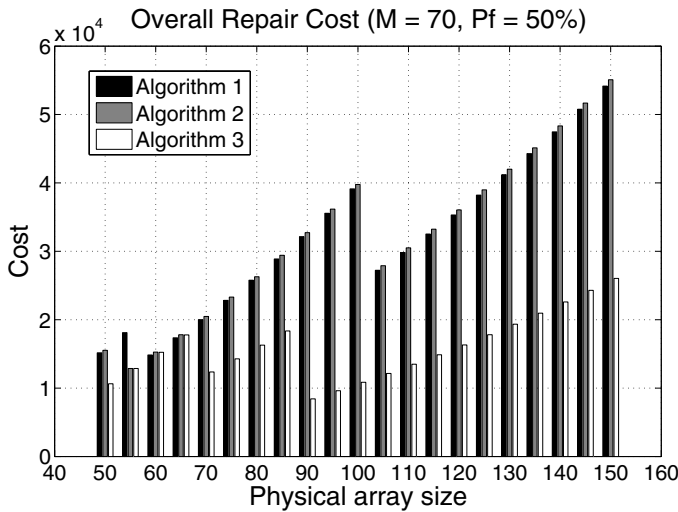
Fig. 3. Plots of overall repair cost for $M = 70$ and $P_f = 50\%$ case.

determined by the specifications of the tester). Let γ denote the cost for programming a crosspoint. For each parameter, as long as the ratios among the costs (α, β and γ) are the same, the plot for the total cost will be the same. It is then possible to add the parameters to obtain the overall repair cost as: $Cost_{overall} = \alpha \times$ *# of unused crosspoints* $+ \beta \times$ *execution time of repair algorithm* $+ \gamma \times$ *# of reconfigured switching points*.

Simulation was performed to validate the proposed cost-driven model, in which the production parameters were given by $\alpha = 1$, $\beta = 500$ and $\gamma = 5$. The results are shown in Figures 2 and 3. From Figure 2, it can be observed that by *algorithm 3* and the physical array size of 80×80, the lowest overall repair cost can be achieved, while successfully implementing the function set I. For *algorithm 1* and *2*, a 50×50 crossbar size can achieve the least costs. To simulate different environments, various values for α, β and γ were chosen. Besides these parameters, the function set I affects the overall cost. The simulation results shown in Figure 2 are based on a function set I with $P_f = 70\%$. Note that in general, other physical designs may not need such a high usage rate for the crosspoints. It is easier to find a matching for a lower row or column usage provided the defect rate is kept the same. The function usage rate P_f is also a significant parameter in determining the overall cost.

Consider the impact on the overall cost with different values of M. $M = 70$, $P_f = 50\%$ and the same cost parameters $\alpha = 1$, $\beta = 500$ and $\gamma = 5$ are assumed for simulation purposes. The results are shown in Figure 3; *algorithm 3* finds an physical array size of 90×90 while *algorithm 1* and *2* find a physical array size of 50×50 and 55×55, respectively. Compared with the result shown in Figure 2, the cost in Figure 3 is much higher, thus demonstrating the importance of M.

VII. CONCLUSION

For nanowire crossbar-based systems, higher defect densities are anticipated due to the unique nature of a bottom-up assembly process. Novel methods are required for tolerating defects. As defects in nanoscale wires can be screened out by testing, this paper focuses on avoiding defective crosspoints in an cost-effective manner. Area, time and reconfiguration have been shown to be important parameters in determining the cost of repair in a crossbar architecture. A model that includes the above parameters, has been proposed and analyzed in detail. The proposed cost-driven repair method is flexible and versatile to provide an estimate of the total cost for the repair solution. Extensive simulation results have been provided.

REFERENCES

[1] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams, "A defect-tolerant computer architecture: Opportunities for nanotechnology," Science, Vol. 280, pp. 1716-1721, 1998.

[2] M. Mishra and S. Goldstein, "Scalable defect tolerance for molecular electronics", Workshop Non-Silicon Computation (NSC-1), pp. 78, 2002.

[3] J. Huang, M. B. Tahoori and F. Lombardi, "On the defect tolerance of nano-scale two-dimensional crossbars," IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 96-104, Oct 2004.

[4] M. Jacome, C. He, G. de Veciana, and S. Bijansky, "Defect tolerant probabilistic design paradigm for nanotechnologies," IEEE/ACM Design Automation Conference (DAC), pp. 1-6, 2004.

[5] M. Tehranipoor, "Defect Tolerance for Molecular Electronics-Based NanoFabrics Using Built-In Self-Test Procedure," IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 305-313, Oct 2005.

[6] H. Naeimi and A. DeHon, "A greedy algorithm for tolerating defective cross points in nanoPLA design," IEEE International Conference on Field-Programmable Technology, pp. 49-56, 2004.

[7] S. Zhang, M. Choi, N. Park and F. Lombardi "Cost-Driven Optimization of Fault Coverage in Combined Built-In Self-Test/Automated Test Equipment Testing," IMTC 04, 2004

[8] M. Choi and N. Park, "Dynamic yield analysis and enhancement of FPGA reconfigurable memory systems", IEEE Transactions on Instrumentation and Measurement, Vol. 51, No. 6, pp. 1300 - 1311, December 2002.

[9] Stapper, C.H. "Simulation of spatial fault distributions for integrated circuit yield estimations," IEEE Transaction on Computer -Aided Design, Vol. 8, No. 12, pp. 1314-1318, 1989.