Defect Tolerance for Nanoscale Crossbar-Based Devices

Mohammad Tehranipoor
University of Connecticut, Storrs

Reza M.P. Rad
University of Maryland, Baltimore County

The need for defect maps and per-chip placement and routing limits the efficiency of test and defect tolerance techniques in nanoscale crossbar-based devices. The authors propose a method using two simulation programs that circumvents these difficulties to find fault-free implementations of logic functions on defective crossbars.

According to the 2005 *International Technology Roadmap for Semiconductors* (http://www.itrs.net/Links/2005ITRS/Home2005.htm), scaling of CMOS technology will face several practical and theoretical difficulties within the next few years. To enhance the domain of information-processing applications beyond CMOS capabilities and keep pace with Moore's law, researchers are investigating several technologies for future computing devices. These include quantum, optical, biologically inspired, molecular, and nanowire and carbon-nanotube-based devices. A high defect rate is expected to be the common problem for these emerging technologies. Their defect density will be considerably higher than that of CMOS devices because of nondeterministic fabrication processes and the dominance of quantum effects at such scales. Managing such high-defect densities will require new test and defect tolerance techniques with low area overhead.

We propose a defect tolerance method for nanoscale devices that does not depend on a defect map. It can also avoid per-chip placement and routing, because crossbar blocks have considerable redundancy to ensure defect-free implementation of the logic functions. Our method takes advantage of such redundancy in each block and tries to configure the functions on defect-free sections of the block. It also tests the mapped function at the same time. Thus, available (inherent) redundancy in the configurable blocks provides defect tolerance, and the additional redundancy required for acceptable yields is small. We have evaluated the proposed method for different defect probabilities that could occur in defect-prone crossbar-based devices. We developed two different implementations of the proposed method and two simulation programs to examine the method for different MCNC (Microelectronics Center of North Carolina) benchmarks implemented on such devices. The first is a C++ program that exhaustively searches for all possible implementations of logic functions on defective crossbars and evaluates the upper limit of defect tolerance in crossbar-based devices. The second simulation program is based on a Verilog model written for crossbars. It performs a greedy search to find a fault-free implementation of logic functions on the crossbars. Analysis and simulation results both confirm that inherent redundancy inside crossbars can effectively provide defect tolerance and achieve high yield.

Limitations of previously proposed strategies

Among the proposed technologies, devices based on crossbars of nanowires or nanotubes are the most widely discussed (http://www.itrs.net/Links/2005ITRS/Home2005.htm). These crossbars can serve to implement memory[1] or memory-based logic devices—for example, programmable logic arrays (PLAs) or lookup tables (LUTs).[2,3] Researchers have proposed several test methods and solutions for defect and fault tolerance for devices built using crossbars and molecular switches. Most of the proposed dynamic fault tolerance techniques are based on traditional techniques used in CMOS devices, such as triple-modular redundancy (TMR), *N*-modular redundancy (NMR), cascaded TMR (CTMR), and coding-based fault tolerance methods.[4,5] However, redundancy-based methods such as TMR and NMR result in high area overhead for devices with very high defect rates. Coding-based methods also suffer from high area overhead because of the need to implement coder and decoder circuits. Moreover, these circuits entail delay overhead. As Kuekes et al. suggest, coding methods can provide good fault tolerance for parts of the device, such as memory-based logic or address decoders in a memory.[5]

Defect-map strategy

Proposed static-defect-tolerance techniques for nanoscale crossbars are based mostly on defect avoidance through reconfiguration and defect mapping. Such techniques originated from Teramac experiments.[6] Teramac, a custom computer built on reconfigurable blocks (FPGAs), can tolerate defects in its components through reconfiguration and avoidance of defective components. This defect avoidance scheme required that the location of defective components be identified during a preconfiguration test process and stored in a defect database called a *defect map*. Similar defect avoidance methods were proposed for nanoscale devices.[7,8] Test methods have also been proposed to identify the locations of faulty components in crossbar-based circuits and store them in a defect map.[9,10] However, using a defect map in the mass production of nanoscale devices poses major challenges. For instance, the random nature of defects means that there will be different defect maps for different chips. Even if we assume that it's possible to find defect locations through a test process, it will be a very time-consuming task for such high-density devices.

Another issue related to defect-map-based strategies is that a defect map for a nanodevice will be extremely large and will be practically impossible to store on chip. Storing a defect map on chip will be prohibitively expensive because, for reliability, a CMOS-scale memory should be used to store it. Shipping a defect map of each chip on a separate storage device along with the chip doesn't seem to be a practical solution either.

Another important issue associated with the defect map strategy is the need to perform a separate place-and-route procedure for each chip, because each chip will have a unique defect map. This too is prohibitively expensive. Also, the configured devices could have varying performance levels because their different defect maps will require the blocks in each nanochip to be placed and routed differently.

Simultaneous configuration and test

SCT is a defect tolerance technique proposed for nanoscale devices that avoids identifying exact fault locations. Instead, this technique combines test and configuration processes to avoid the need for a defect map.[11] The SCT method uses conventional partitioning and place-and-route tools for configurable devices. A given application is partitioned into small logic functions (f_i) that can be implemented on programmable nanoscale logic blocks. The place-and-route tool then assigns one of the device's logic blocks to each function and determines the required routing.

Once the place-and-route tool has specified the logic function assigned to each programmable nanoscale logic block and the required routing, the SCT method configures each function (f_i) of the circuit into its assigned nanoblock (b_j) in the device. The function will also be configured onto an LUT implemented as part of a small BIST circuit in the device's CMOS support layer, as Figure 1 shows. The function, configured on a nanoscale block, is then tested exhaustively. Test patterns applied to its inputs are sent back, along with the responses, to the BIST circuit for comparison with the results obtained from the CMOS LUT, which contains a fault-free copy of the function. If function f_i fails the test, the corresponding block (b_j) will be avoided during the next place-and-route phase. Thus, the SCT method removes the requirement to store defect information in a defect map, but it still requires per-chip placement and routing.

Defect avoidance method

We can best describe the main objective of our proposed method through an example. Figure 2 shows two implementations of function $f = ab + b'c$ on a defective crossbar-based PLA. The crossbar in the figure is a combination of an AND plane and an OR plane (similar to traditional PLAs) and can be implemented on the basis of diode logic by using a crossbar of nanowires and configurable molecular switches on the crosspoints of the wires.[8] The crossbar

PLA can also be implemented using FET switching properties created on carbon nanotubes. As the figure shows, even in the presence of many defects in the crossbar, there are several alternatives for fault-free implementation of function *f*. This is because of the extensive inherent redundancy in this crossbar and its many additional resources (switches and wires) available for implementing the function. In general, we consider the case of configuring a function of size (K_f, M_f, 1)—that is, K_f inputs, M_f product terms, and one output—on a crossbar of size (K, M, N), where K, M, and N are the number of input lines, product lines, and output lines, respectively. We then calculate the number of possible ways to implement a function on the crossbar as

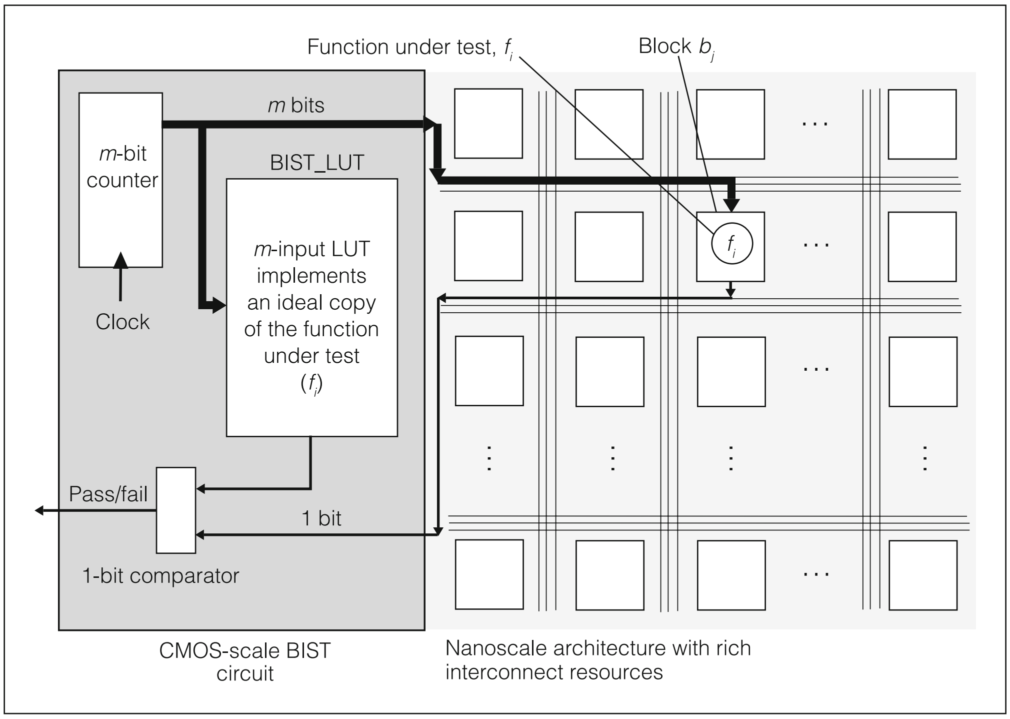

Function under test, f_i
Block b_j
m bits
m-bit counter
BIST_LUT
m-input LUT implements an ideal copy of the function under test (f_i)
Clock
f_i
Pass/fail
1 bit
1-bit comparator
CMOS-scale BIST circuit
Nanoscale architecture with rich interconnect resources

Figure 1. Proposed simultaneous configuration and test (SCT) method.[11] The CMOS BIST circuit is used to functionally test nanoscale blocks. (LUT: lookup table.)

$$P^M_{M_f} \times P^K_{K_f} \times P^N_1$$

where $P^M_{M_f}$ represents all possible permutations of M_f product terms among M product lines ($M_f \le M$ and $K_f \le K$). This relation holds true for a defect-free crossbar whose resources are all usable for implementing a function (a defect-free block). For example, to implement a function of size (5, 6, 1) on a crossbar of size (7, 8, 3), there are 7,065 possible configurations. This is a large number for configuring only one block. Efficient algorithms must be devised to find a fault-free mapping of the function onto the crossbar.

Redundancy is also available in interconnects formed using crossbars of the nanowires existing between blocks of a device. It's possible to find several paths from one block to another without changing the structure of the path between them; hence, the path's timing characteristics will not change. In other words, each segment of the route between two blocks contains several usable tracks. The amount of redundancy in a device's routing resources is a design parameter (as is the block size) that should be determined for the device's architecture based on the targeted yield. Traditional FPGA design includes consideration of an optimum value for the number of available tracks in each routing channel so as to reach an acceptable trade-off between area and performance. For nanowire-based reconfigurable devices, designers should select the number of tracks in each routing channel carefully so that routing resources can provide the required defect tolerance while maintaining an acceptable area range.

Configuration and test procedure

The first part of the procedure, conventional placement and routing, occurs in the design phase. In addition to placement and routing of blocks, this phase determines the device's boundary and the connections between the blocks and the boundary. An external tester or a reliable on-chip CMOS-scale test generator can apply test patterns to internal blocks and read back the test results (see Figure 1), but a fault-free path must be identified and configured between the tester and the block. The way to detect and configure paths depends on a device's routing architecture. We suggest that starting from the edges of the array of configurable blocks and testing each block for a pass-

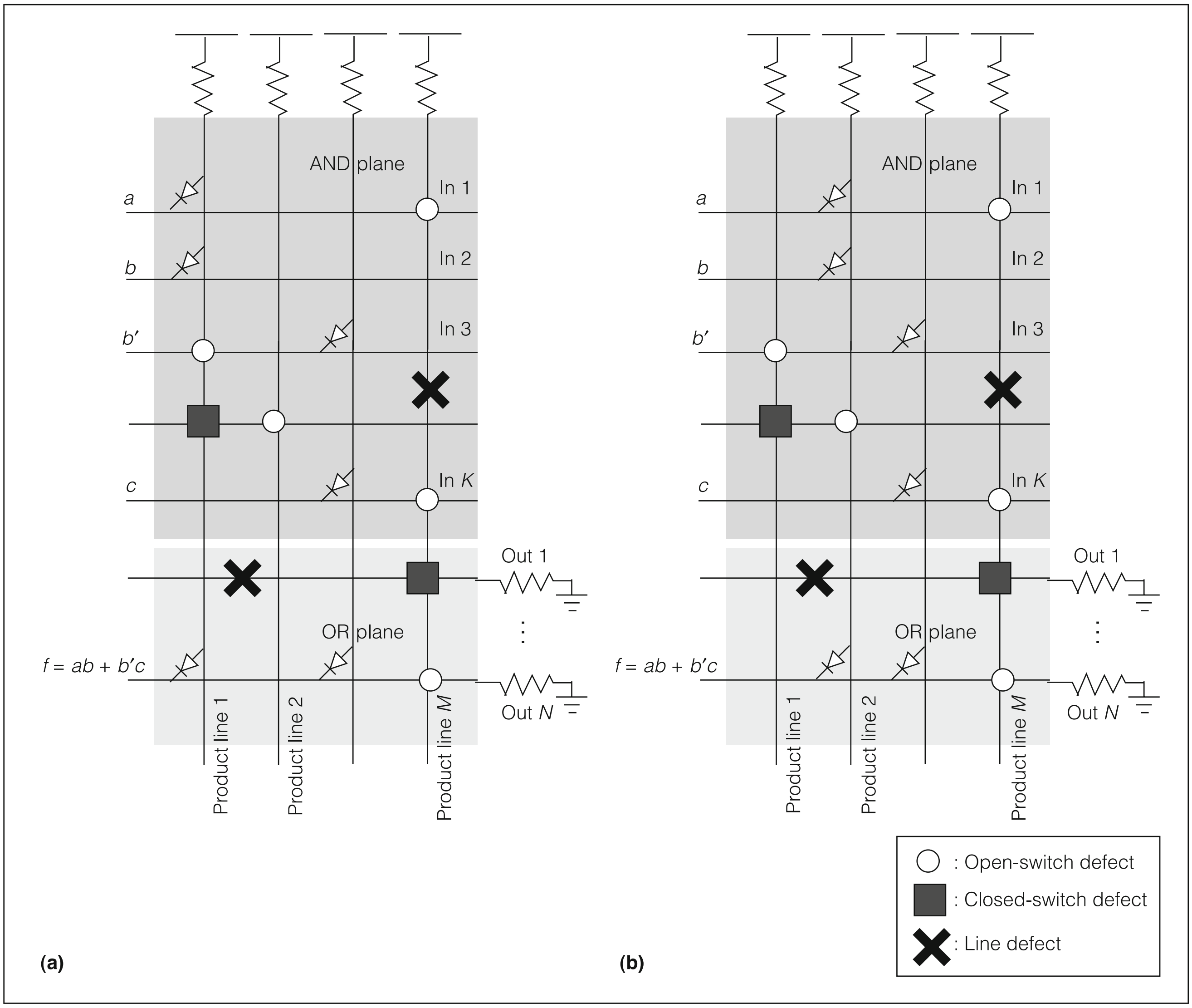


Figure 2. Two different implementations of function *f* = *ab* + *b*′*c* on a defective crossbar: *ab* and *b*′*c* are generated using product lines 1 and 3 (a), and *ab* and *b*′*c* are generated using product lines 2 and 3 (b).

through function can identify fault-free paths to the targeted block in the nanodevice. Because both routing resources and blocks are based on crossbars, similar defect tolerance capabilities likely exist in the routing section of nanodevices. Therefore, using the proposed method for interconnects will require similar test and configuration efforts and will result in similar defect tolerance and yield values.

With fault-free paths to the target blocks identified, the procedure goes on to configure and test each function on the block specified in the placement phase. This can be implemented through two different approaches, discussed in greater detail later. Simulation programs developed in this article focus on identifying a fault-free implementation of the functions on their associated blocks.

Limitations in lithographic resolution can make electrical contacts between CMOS metal wires and nanowires difficult to achieve, leading to difficulties such as nonohmic contacts. These problems make it more difficult to send test patterns to the blocks and receive the results. However, researchers in emerging technologies are addressing these challenges, and the possibility of using nanowire and molecular-switch circuits will depend on finding effective interfacing solutions for CMOS and nanowires. We assume here that future fabricated nanochips will provide such an interface mechanism.

Identifying fault-free configuration of logic functions

We have implemented our proposed test and configuration method in two ways. The first is based on an exhaustive search that examines all possible configurations of each logic function f_i on block b_j. In this approach, for each configuration of the function, the crossbar switches are configured and test patterns are applied through selected paths from device boundaries to the block. If the test results for the function show fault-free behavior, then the desired configuration has been found. This repeats for all functions of the application. Exhaustive searching of all possible configurations of functions on the crossbar blocks of each nanodevice is not practical, because of the time such searching requires. However, this simulation can provide an upper limit of defects that can be tolerated in crossbar-based devices.

Our second algorithm implements function f_i on block b_j through a greedy-search method. First, the algorithm applies simple test patterns to find K_f fault-free nanowires among K inputs of block b_j and one fault-free nanowire from this block's N outputs. Then, each product term of f_i is configured on one of the available product lines of b_j, and test vectors are applied to test the product term. If the product line shows faulty behavior, the product term is configured on the next available product line and tested again. This repeats until all product terms of f_i are configured and tested on product lines of b_j. Compared with the exhaustive method, this greedy approach reduces the required search time, but it doesn't cover all possible combinations of inputs, product lines, and outputs; therefore, it could be less successful in implementing f_i on defective crossbars. However, our simulation results demonstrate that the technique can still achieve high yield.

Probabilistic analysis

Performing a probabilistic analysis evaluates the feasibility of the proposed method when using the greedy-search approach. In our analysis, P_l represents the probability of a line (horizontal or vertical) defect in the crossbar nanowires, P_o denotes the probability of a stuck-open defect in a molecular switch, and P_c represents the probability of a stuck-closed defect on a molecular switch.

Defects in crossbar nanowires can appear as bridging or broken nanowires. Molecular layers between horizontal and vertical nanowires can have inhomogeneities that cause a direct connection between a horizontal and a vertical nanowire (stuck-closed), or there can be impurities at the crosspoint of two nanowires, causing a stuck-open defect between them. In the literature, these are considered the major defect sources for nanowire crossbars.

In our probabilistic analysis, we first calculate the probability of finding K_f defect-free input lines (P_{in}) among all K inputs of the crossbar, and the probability of finding one defect-free output line (P_{out}) among N outputs of the crossbar. Once K_f defect-free inputs and one defect-free output are found for implementing the function, M_f defect-free product terms should be found among M product lines of the crossbar. For a product line of the crossbar to be usable as a product term of the function, the product line itself should be defect free, and the switches on the crosspoints of the line with K_f input lines and one output line should also be defect free. At least M_f of these defect-free product lines are necessary to implement the function on the crossbar. We denote the probability of having at least M_f defect-free product lines as $P_{\text{min\_lines}}$. Finally, the probability of successful implementation of the function f_i on the crossbar $b_j(P_{\text{success}})$ can be calculated on the basis of P_{in}, P_{out}, and $P_{\text{min\_lines}}$:

$$P_{\text{min\_lines}} \geq 1 - \sum_{i=M-M_f+1}^{M} \left(1 - P^{i}_{\text{pterm}}\right)$$

$$P_{\text{success}} \geq P_{\text{in}} \times P_{\text{out}} \times P_{\text{min\_lines}}$$

Figure 3 shows the probability of successful implementation of a function on a crossbar for different defect densities and different amounts of redundancy provided in the crossbar. Area ratio (A_r) is defined as the ratio of crossbar area to the minimum area required by the function and is calculated as

$$A_r = \frac{M(K+N)}{M_f(K_f+1)}$$

A_r serves as a measure for the amount of redundancy, and the probability calculations are performed for different values of A_r.

As Figure 3 shows, having only a moderate amount of redundancy in the crossbar results in a very high probability of successful implementation (P_{success}). Note that for the results shown in Figure 3, we have set $P_c = P_l = P_o$. Therefore, the probabilities shown in the

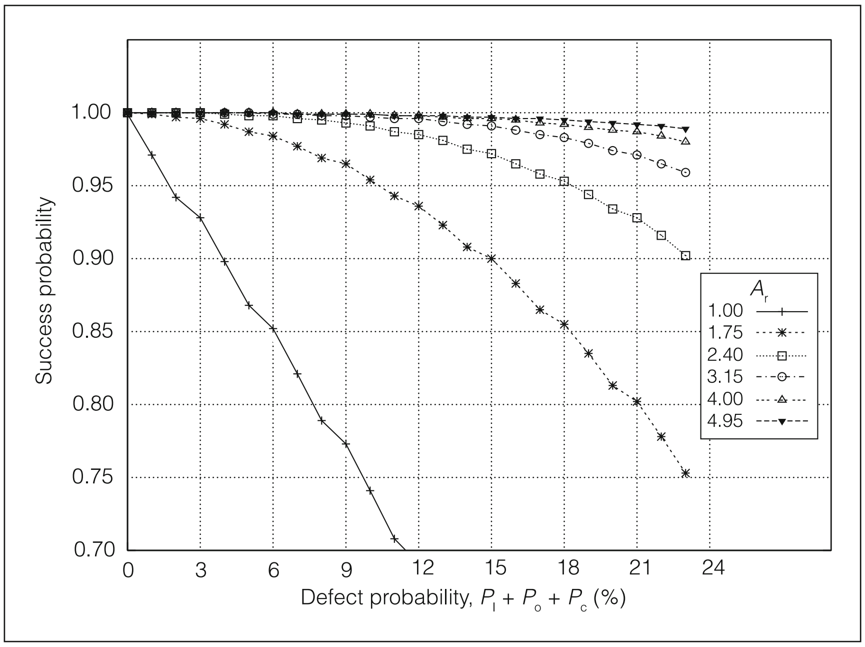


Figure 3. Probability of successful implementation ($P_{success}$) of a function on a crossbar programmable logic array (PLA) for different defect probabilities and different amounts of redundancy. (A_r: area ratio; P_c: probability of stuck-closed defect on each molecular switch; P_l: probability of line defect in crossbar nanowires; P_o: probability of stuck-open defect in a molecular switch.)

Exhaustive-search approach

The first simulation program performs an exhaustive search to implement the logic functions on defective crossbars. A crossbar is modeled as a 2D array, and each array element specifies the status of one of the crosspoints of vertical and horizontal nanowires in the crossbar. Defects are randomly generated and injected into this array. As discussed previously, we consider stuck-open defects, stuck-closed defects, and nanowire line defects with probabilities of P_o, P_c, and P_l. In these experiments, the program iteratively tries to implement each function on a defective crossbar and measure the average rate of success in those implementations. The results of these experiments provide the success rate for implementing functions on defective crossbars under different defect rates through exhaustive search. Figure 4a shows a diagram of the steps performed by the program.

horizontal axis of Figure 3 are the total probabilities of stuck-open, stuck-closed, and line defects. In fact, this is a pessimistic assumption, because we expect stuck-closed and line defects to have a far lower probability than stuck-open defects. Therefore, we can expect higher $P_{success}$ in implementing functions on the crossbar.

Simulation programs

To evaluate the proposed method in terms of the ability to find a defect-free implementation of functions on crossbars and at the same time to test the configured functions, we implemented the method in two different ways and developed two simulation programs to run on MCNC benchmarks. Both implementations use Sequential Interactive Synthesis (SIS) Flowmap and Flowpack outputs. SIS is a synthesis package that can map the benchmarks into partitions (in this case, each function f_i) of a specified size. Each of these partitions can be implemented on a crossbar PLA. In our simulations, SIS provides the required partitioning of benchmarks into small functions of size K_f. In both simulation programs, the SCT method is the main engine for on-chip test generation and application.[11]

Greedy-search approach

The second simulation tool also consists of three parts: the SIS package, a Perl script, and a Verilog model for the nanowire crossbar. Figure 4b shows a high-level flow of the developed simulation tool. The Perl part of the simulation program reads SIS outputs and extracts the functions from a benchmark, thereby obtaining the product terms of each function. Then, the program seeks different possible configurations for each function on the crossbar. For each of these configurations, the program generates input and configuration files and applies them to the Verilog model of the crossbar. The Verilog simulator then simulates that configuration. The process repeats for all product terms for each function in the benchmark. A defect file, created by random assignment of defects to different crossbar components, helps in implementing each function on the crossbar model. Probabilities of line defects, stuck-open switches, and stuck-closed switches are inputs to the program, and they can be tuned to simulate different defect densities. We also apply the generated defect file to the Verilog model of the crossbar and use it in all steps of implementing a function. We created a new random defect file to

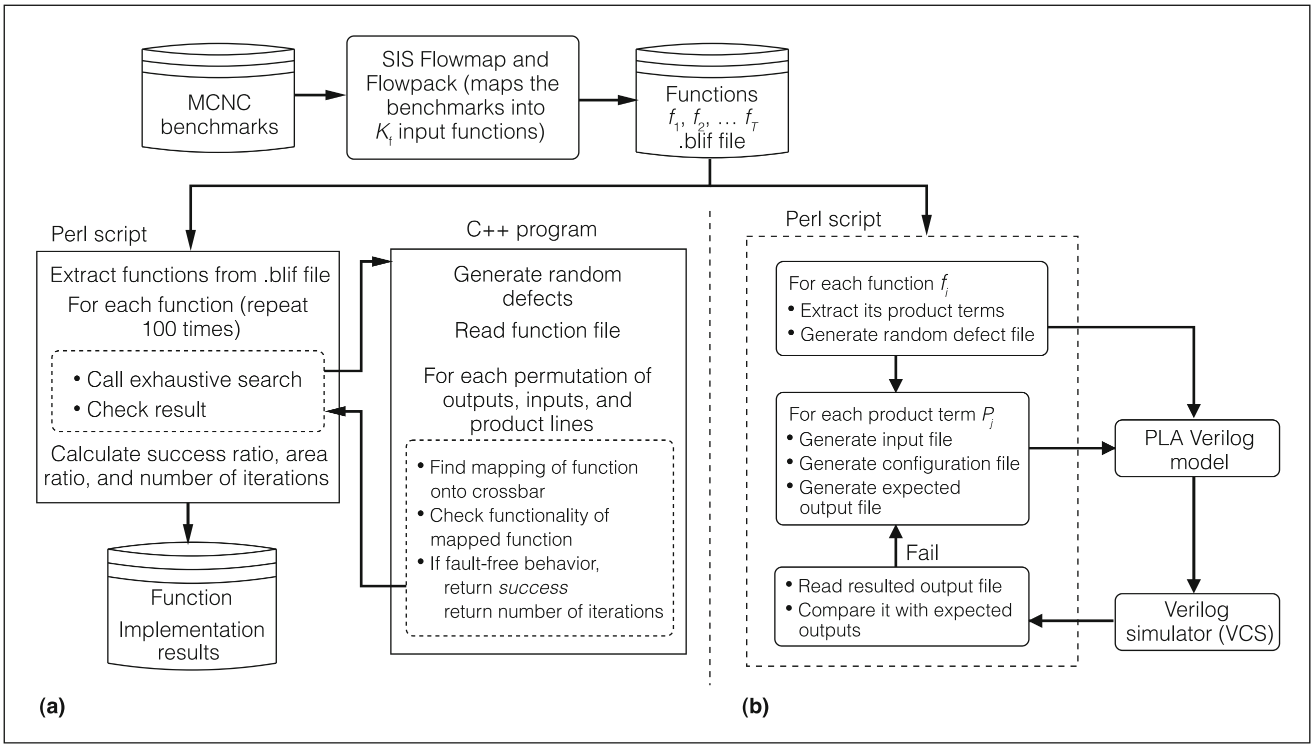

MCNC benchmarks
SIS Flowmap and Flowpack (maps the benchmarks into K_f input functions)
Functions $f_1, f_2, \ldots f_T$ .blif file
Perl script
Extract functions from .blif file
For each function (repeat 100 times)
• Call exhaustive search
• Check result
Calculate success ratio, area ratio, and number of iterations
Function Implementation results
C++ program
Generate random defects
Read function file
For each permutation of outputs, inputs, and product lines
• Find mapping of function onto crossbar
• Check functionality of mapped function
• If fault-free behavior, return success return number of iterations
Perl script
For each function f_i
• Extract its product terms
• Generate random defect file
For each product term P_j
• Generate input file
• Generate configuration file
• Generate expected output file
Fail
• Read resulted output file
• Compare it with expected outputs
PLA Verilog model
Verilog simulator (VCS)
(a)
(b)

Figure 4. Flow of mapping functions onto defective crossbars using the exhaustive-search approach (a) and the greedy-search approach (b). (MCNC: Microelectronics Center of North Carolina; SIS: Sequential Interactive Synthesis.)

represent different defects on the new crossbar, and apply this as well to each implementation of a function.

The Verilog crossbar model includes behavioral models for each component in the crossbar: the product lines, the input lines, the output lines, and the molecular switches between input and output lines and product lines. For each of these components, we consider appropriate defect types, and describe component behavior under each of these defects. Finally, all these components are combined in an array to create the crossbar module. This module can read the configuration information file, the defect information file, and the input file. Once these three files are prepared and applied to the model, the Verilog simulator can generate and store the crossbar's outputs in a file. Our developed Perl program analyzes these outputs to verify whether the defective crossbar is working correctly under the current configuration.

To find a defect-free implementation of function f_i on the crossbar, the simulation program looks for appropriate positions for each product term (pt^i_k) of function f_i in the crossbar. The program maps the product term pt^i_k into a product line of the crossbar, then calls the Verilog simulator to analyze the simulation output file. If the Verilog simulation result is correct for the specific product term, the process is repeated for another product term in the function. Otherwise, the program selects a new product line, a new configuration file, and a new input file, and the Verilog simulation is repeated.

Once all product terms of the function f_i have been placed on product lines of the crossbar model and their functionality has been verified, the task of implementing the function is complete. The program performs this task for all functions of the benchmark. If the program cannot find a defect-free implementation of function f_i on a crossbar, we assume that the chip will be discarded.

For our simulations, we specified the size of the functions in the benchmarks—that is, the number of their inputs (K_f)—through the mapping phase in the SIS package. The number of crossbar model inputs can be specified as a parameter in the Verilog code. So, we can calculate the amount of redundancy available in the model. Also, we can specify defect densities as parameters in the program. The simulation program calculates and reports the overall yield resulting from the process of testing, configuring, and searching for defect-free implementations of benchmarks.

Simulation results

We ran the simulation programs on 10 MCNC benchmarks (i_1 through i_{10}). In our simulations, defects were randomly injected into the crossbar Verilog models. We assumed a uniform defect distribution and conducted the experiments under different defect probabilities.

Results from the exhaustive-search approach

We performed the exhaustive search for very high defect rates. In these experiments, $P_o = 10\%$ and 15%; $P_c = 10\%$, 12%, and 15%; and $P_l = 5\%$ and 10%. These defect conditions are beyond the expected rate of actual defects for nanowire crossbars. However, we assumed these extremely high defect rates to evaluate the upper limits of defect tolerance on a crossbar if an exhaustive search is to be used to find fault-free implementations of crossbar functions. The number of inputs of the functions (K_f) was set to be fewer than four. For each of these functions, the simulation program was repeated to obtain the number of successful mappings of the function onto defective crossbars. We also obtained the average number of iterations the exhaustive-search program required to find a fault-free implementation of each function. In these experiments, we considered a crossbar of size $M = 7$, $K = 7$, and $N = 3$. Because the benchmark functions are bounded to fewer than four inputs, area ratios differ on the basis of the function size (number of its inputs and product terms) and the crossbar size.

Another parameter that must be considered in these experiments is the amount of redundancy provided in crossbars. We can measure this redundancy as the area ratio (A_r) between the crossbar of size $M \times K \times N$ and the functions of size $M_f \times K_f \times 1$ implemented on the crossbar. We performed the experiments for different amounts of redundancy to evaluate the effect of redundancy on the method's efficiency.

The success rate, S, is defined as the ratio of the number of times the implementation of a function on the crossbar is successful, F_S, to the total number of functions in the benchmarks, F_T. That is, $S = F_S/F_T$. This definition of success rate helps us understand how successful the method has been at providing defect tolerance in a nanoscale device.

Figure 5a shows the average success rate (by percentage) when implementing a function onto the crossbar under different defect rates. Note that very high defect rates can be tolerated in crossbars when there is enough time to search for a fault-free implementation. The observed tolerance against high defect rates encourages developing faster algorithms for finding a fault-free implementation of the crossbar functions.

Figure 5b shows the average number of iterations required to find a fault-free implementation of functions on crossbars. As the figure shows, increasing the area ratio between the crossbar and the function to be mapped on it (that is, increasing redundancy) reduces the number of required iterations significantly.

Results from greedy-search approach

In these experiments, we assume the stuck-open probability for molecular switches ranges from 0% to 10% and that the stuck-closed and line defect probabilities range from 0% to 6%. Therefore, in the worst case for these experiments, there will be, in total, a 10% probability of stuck-open defects plus a 6% probability of stuck-closed defects and a 6% probability of nanowire (line) defects in the crossbar model, which is still higher than the expected defect rate for nanowire crossbars.

The first set of results, shown in Figures 6a and 6b, are the average success rates obtained from running the simulation program on the 10 MCNC benchmarks. We obtained the rates using two different amounts for redundancy and two amounts for defect probability.

Although these results are only for the device blocks and not the routing, a high success rate is possible for both blocks and routing. For example, as Figure 6b shows, when $P_o = 6\%$ and $P_l = P_c = 2\%$, an 80% success rate is achievable. As reported in the literature, for nanoscale crossbar fabrication methods, defect probabilities of up to 10% for stuck-open faults are expected, whereas considerably lower probabilities are predicted for stuck-closed and line defects.[5] As Figures 6a and 6b show, the success rate decreases considerably for higher values of stuck-closed and line defects, causing parts of the crossbar (a defective line or two closed lines) to function incorrectly. Therefore, these defects' adverse effect on the success rate is greater than that of stuck-open defects.

Figure 7a compares achieved success rates for four different redundancy conditions. These results are shown for a 2% probability of stuck-closed and line defects ($P_l = P_c = 2\%$) and different values of stuck-open probability (P_o). As Figure 7a shows, a relatively high success rate is achievable through the proposed test and configuration method. Figure 7b shows the average number of switch configurations for different redundancy values. As redundancy increases, a crossbar's size—and hence the number of its molecular switches—also increases. Therefore, the number of switch configurations increases, as does the overall test process time. Figure 7c shows the average number of test cycles for different values of redundancy. Enlarging the crossbars used in a device (that is, increasing redundancy) results in additional test cycles because there are more product lines in each crossbar that need to be searched to find a defect-free implementation of functions. Therefore, higher redundancy results in more test cycles, more switch configurations, and longer runtime.

Defect map generation remains a challenge when testing nanoscale crossbars. By simultaneously configuring and testing nanoblocks, our proposed method can avoid defect maps while reducing total test cost. In the future, we would like to further reduce test time by using a parallel SCT architecture. We will also incorporate interconnect redundancy into our algorithms and analyze its impact on the overall yield and test time. ■

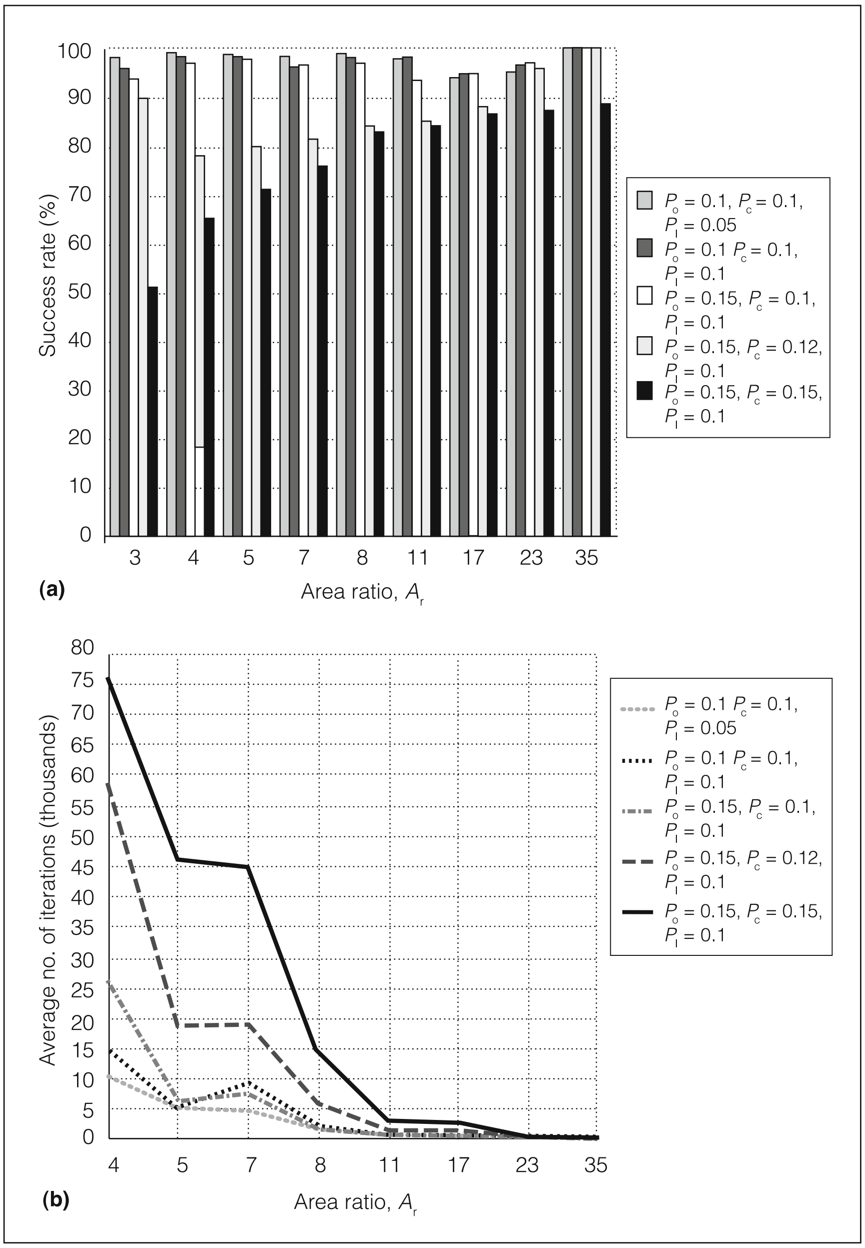


Figure 5. Results for implementing benchmark logic functions on defective crossbars using the exhaustive-search approach: average percentage of successful implementations of benchmark functions (a) and average number of required iterations to find a fault-free implementation (b).

■ References

1. A. DeHon et al., "Nonphotolithographic Nanoscale Memory Density Prospects," *IEEE Trans. Nanotechnology*, vol. 4, no. 2, Mar. 2005, pp. 215-228.
2. A. DeHon and M.J. Wilson, "Nanowire-Based Sublithographic Programmable Logic Arrays," *Proc. Int'l Symp. Field Programmable Gate Arrays* (FPGA 04), ACM Press, 2004, pp. 123-132.
3. R.M.P. Rad and M. Tehranipoor, "A New Hybrid FPGA with Nanoscale Clusters and CMOS Routing," *Proc. 43rd Design Automation Conf.* (DAC 06), ACM Press, 2006, pp. 727-730.
4. D. Bhaduri and S. Shukla, "NANOLAB—A Tool for Evaluating Reliability of Defect-Tolerant Nanoarchitectures," *IEEE Trans. Nanotechnology*, vol. 4, no. 4, July 2005, pp. 381-394.
5. P. Kuekes et al., "Defect-Tolerant Interconnect to Nanoelectronic Circuits: Internally Redundant Demultiplexers Based on Error-Correcting Codes,"

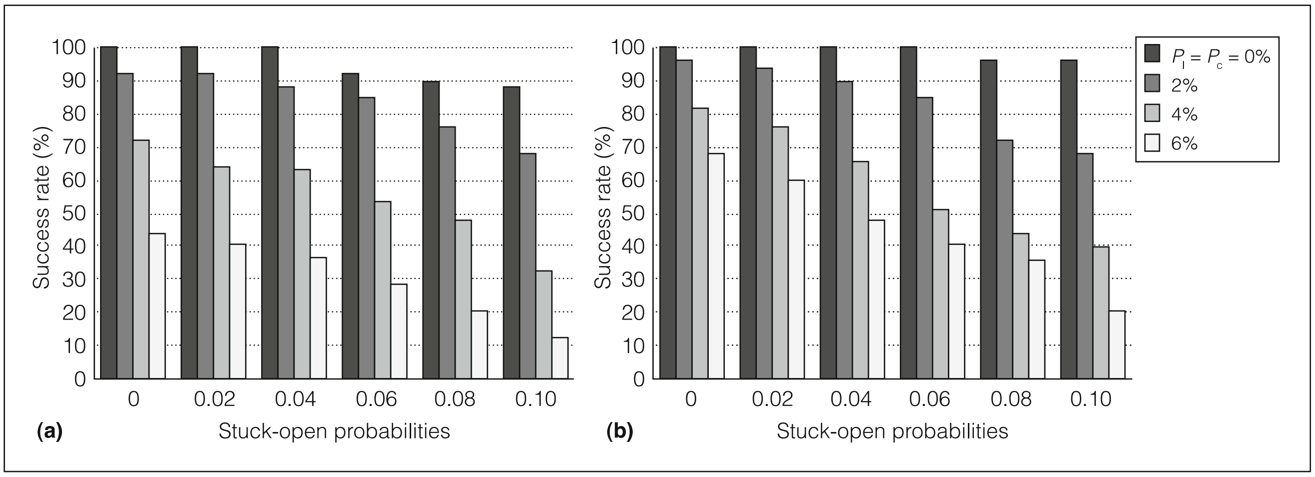
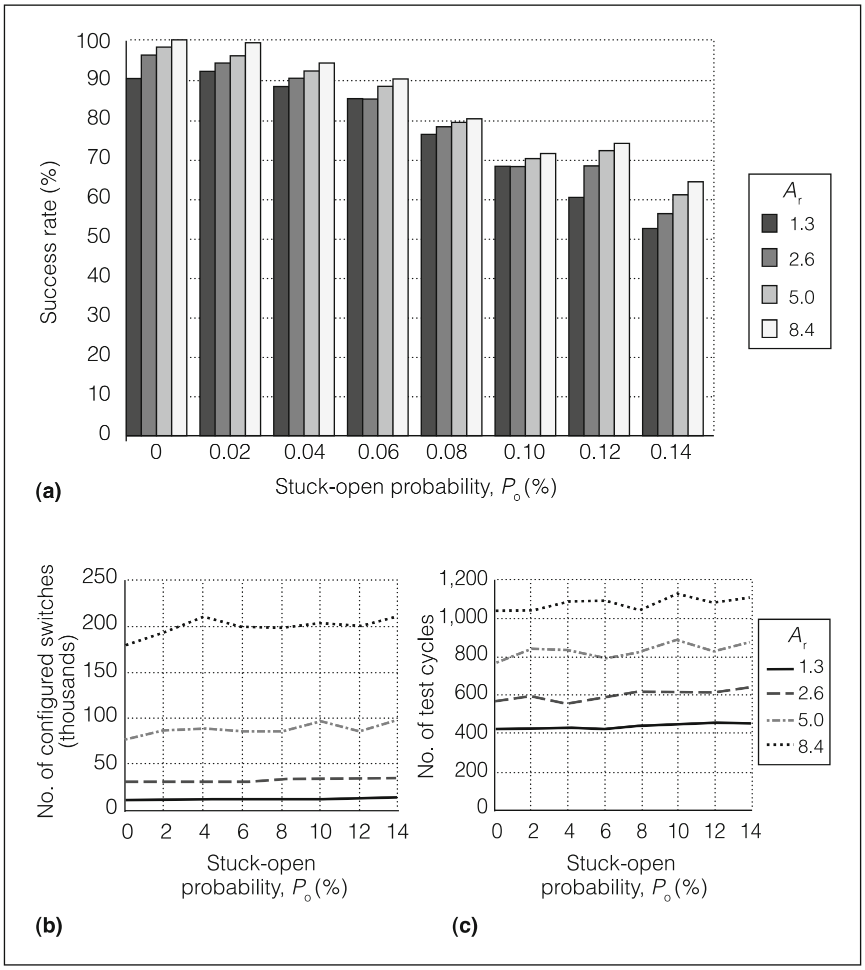


Figure 6. Success rate for implementing benchmarks on a defective crossbar using the greedy-search approach: $A_r = 1.3$ (a) and $A_r = 2.6$ (b).

Nanotechnology, vol. 16, no. 6, June 2005, pp. 869-882, http://www.iop.org/EJ/abstract/0957-4484/16/6/043.

6. B. Culbertson et al., "Defect Tolerance on the Teramac Custom Computer," *Proc. 5th IEEE Symp. FPGA-Based Custom Computing Machines* (FCCM 97), IEEE CS Press, 1997, pp. 116-123.

7. A. DeHon and H. Naeimi, "Seven Strategies for Tolerating Highly Defective Fabrication," *IEEE Design*



←

Figure 7. Experimental results using the greedy-search approach: average success rate for implementing benchmarks on defective crossbars with different redundancy values (a); average required number of switch configurations for implementing benchmarks with different redundancy values (b); and average required number of test cycles for implementing benchmarks with different redundancy values (c). (For all three parts of the figure, $P_l = P_c$ = 2%.)

& Test, vol. 22, no. 4, July-Aug. 2005, pp. 306-315.

8. S.C. Goldstein and M. Budiu, "NanoFabrics: Spatial Computing Using Molecular Electronics," *Proc. 28th Int'l Symp. Computer Architecture* (ISCA 01), IEEE CS Press, 2001, pp. 178-189.
9. M. Tehranipoor, "Defect Tolerance for Molecular Electronics-Based NanoFabrics Using Built-in Self-Test Procedure," *Proc. 20th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems* (DFT 05), IEEE CS Press, 2005, pp. 305-313.
10. Z. Wang and K. Chakrabarty, "Using Built-in Self-Test and Adaptive Recovery for Defect Tolerance in Molecular Electronics-Based Nanofabrics," *Proc. Int'l Test Conf.* (ITC 05), IEEE CS Press, 2005, pp. 486-495.
11. R.M.P. Rad and M. Tehranipoor, "SCT: An Approach for Testing and Configuring Nanoscale Devices," *Proc. 24th IEEE VLSI Test Symp.* (VTS 06), IEEE CS Press, 2006, pp. 370-377.

Mohammad Tehranipoor is an assistant professor of electrical and computer engineering at the University of Connecticut, Storrs. His research interests include computer-aided design and test for CMOS VLSI designs and emerging nanoscale devices, DFT, at-speed test, secure design, and IC trust. He has a BSc from Amirkabir University of Technology, an MSc from the University of Tehran, and a PhD from the University of Texas at Dallas, all in electrical engineering. He is a senior member of the IEEE and a member of the ACM and ACM SIGDA.

Reza M.P. Rad is a research assistant pursuing a PhD in computer engineering at the University of Maryland, Baltimore County. His research interests include design, testing, and defect and fault tolerance issues in digital systems. He has a BSc in electrical engineering and an MSc in computer engineering from the University of Tehran, Iran. He is a student member of the IEEE and the IEEE Computer Society.

■ Direct questions and comments about this article to Mohammad Tehranipoor, Dept. of ECE, University of Connecticut, 371 Fairfield Way, Unit 2157, Storrs, CT 06269-2157; tehrani@engr.uconn.edu.