BISM: Built-in Self Map for Crossbar Nano-Architectures

Mehdi Tahoori
Northeastern University
Boston, MA 02115
mtahoori@ece.neu.edu

Abstract

Bottom-up self-assembly process used in the fabrication of nanoscale devices yields significantly more defects compared to conventional top-down lithography used in CMOS fabrication. Therefore, applying defect tolerant design schemes is inevitable to be able to map the design to these programmable fabrics around defects. However, defect-aware design mapping could be very time consuming and complicated both at design time and post-fabrication configuration time. In this paper, we present an alternative approach for defect tolerant mapping in which the mapping phase is built into the programmable fabric and on-chip resources are used for defect-free mapping. This built-in self map *(BISM) scheme significantly reduces the complexity of defect tolerance both at design time and post-fabrication configuration time.*

1 Introduction

A considerable amount of research is currently focused on developing nanoscale devices and alternative nanotechnologies to supersede conventional lithography-based CMOS technology. Current integrated circuits are designed using a *top-down* approach where lithography imposes a pattern. Unnecessary bulk material is then etched away to generate the desired structure. An alternative *bottom-up* approach, which avoids the sophisticated and expensive lithographic process, utilizes *self-assembly*, in which nanoscale devices can be self-assembled on a molecule by molecule basis. Researchers have shown successful realization of nano devices, such as carbon nanotubes (CNTs) and silicon nanowires (NWs), using self-assembly techniques [1, 2, 3]. Chemically self-assembled structures, as the building blocks for molecular-scale computing, are by their nature very regular and therefore well suited to the implementation of regular arrays similar to conventional *Field Programmable Gate Arrays* (FPGAs). Reprogrammable nano-architectures are currently being investigated [4, 5].

Self-assembly processes promise to considerably lower manufacturing costs, but at the expense of reduced control of the exact placement of these devices. Without fine-grained control, these devices will certainly exhibit higher defect rates [1, 2, 5]. Moreover, nanofabrication process yields nanowires which are a few atoms long in the diameter. For instance, the contact area between nanowires contains only a few tens of atoms. With such small cross-section and contact areas, fragility of these devices is orders of magnitude more than devices currently being fabricated using conventional lithography techniques.

The programmability of crossbar nano-architectures can be used to implement defect and fault tolerant schemes for the designs mapped into these architectures. After identifying defective resources using thorough test and precise fault localization [6, 7, 8, 9, 10], they can be bypassed by post-fabrication configuration. However, conventional application-dependent defect tolerant schemes impose complexity for the design and test flow as well as the post-fabrication configuration time. The test and diagnosis flow is required to specify the exact location of defective resources within the nano-architecture (application-independent test and diagnosis). This means that very high resolution and time consuming diagnosis is performed, and the information, in a very large *defect map*, is provided to the physical design flow. This information is used for post-fabrication configuration to bypass defects. The entire process has to be repeated in a per-chip basis which makes it impractical for high-volume production.

In this paper, we present an alternative approach for defect tolerant mapping in which bypassing defective resources for application mapping (post-fabrication configuration) is performed by the nano-architecture (*self-map*) using on-chip resources (*built-in*). This

built-in self map (BISM) strategy removes all complexities due to defect tolerance from design and test flow and makes post-fabrication configuration much faster. In BISM, only application-dependent test and diagnosis are used which are significantly less complex and time-consuming compared to application-independent test and diagnosis in the traditional flow. We present variations of BISM tailored for various defect density levels and configuration time. We also show how BISM can effectively be exploited in the implementation of fault tolerant schemes in order to tolerate defects during lifetime operation.

The rest of this paper is organized as follows. In Section 2, an overview of crossbar nano-architectures is presented. Section 3 describes the proposed BISM scheme. Finally, Section 4 concludes the paper.

2 Crossbar Array Nano-Architectures

Bottom-up approaches used in the fabrication of nano-scale devices rely on self-assembly for defining feature size and may offer opportunities to drastically reduce the number of steps required to produce a circuit. However, the biggest impact in going from top-down designs to bottom-up is the inability to arbitrarily determine placement of devices or wires. Without fine control of the design, devices made from self-assembly techniques tend to be restricted to simple structures, such as two-terminal devices. Since these devices are usually non-restoring, one design challenge would be providing signal restoration between nanoscale logic stages. Two dimensional (2D) crossbars are the building blocks of reconfigurable crossbar array architectures. Two layers of orthogonal nanowires or carbon nanotubes form the crossbars [11]. These nanowires or carbon nanotubes are aligned in parallel rows using bottom-up self assembly process. At each intersection, also called *crosspoint*, there is a programmable switch which is non-volatile. A one-time programmable switch consists of a monolayer of redox-active rotaxanes sandwiched between metal electrodes is demonstrated in [2]. In the *closed* state, the switch acts as a diode. It can also be irreversibly *opened* by applying a oxidizing voltage across the device. In these crossbars, configuration of crosspoint is performed by applying a higher voltage (programming voltage) to the intersecting nanowires. Unlike programmable crosspoints in conventional VLSI which are an order of magnitude larger than wire crossing area, crosspoint switches in this nanotechnology take the same area as of a wire crossing.

A chemically assembled electronic nanotechnology FPGA-like architecture called *NanoFabric* has been proposed in [5]. Nano logic arrays, also called *Nanoblocks*, implement a diode-resistor logic (DRL) since crosspoints act as programmable diodes. Another array based nano-architecture using *Programmable Logic Arrays* (PLAs) has been presented in [4]. The main building block, called the nano programmable logic array (nanoPLA), is built from a crossed set of N-type and P-type nanowires. The nanoPLA is programmed using lithographic-scale wires along with stochastically-coded nanowire addressing [12]. The *molecular CMOS* (CMOL) circuits proposed in [13] are designed using the same crossbar array structure as the nanoPLA design consisting of two levels of nanowires. The main difference with CMOL is how the CMOS/nanodevices are interfaced. Pins are distributed over the circuit in a square array, on top of the CMOS stack, to connect to either lower or upper nanowire levels. By angling the nanoarray, the nanowires do not need to be precisely aligned with each other and the underlying CMOS layer in order to be able to uniquely access a nanodevice.

3 Built-in Self-mapping (BISM)

Thorough testing and precise high-resolution location of failing resources in a defective part are keys to successful implementations of defect and fault tolerance. *Built-in-self-testing* (BIST) and *Built-in-self-diagnostics* (BISD) are key components for effective defect tolerance and self-repair with minimized dependence on the external test equipment. Built-in test techniques have been used for FPGAs [14, 15, 16, 17] and crossbar nano-architectures [7, 9, 10].

Since it is expected that all manufactured nanochips contain a considerable percentage of defects even in a mature fabrication process, defect tolerance is inevitable. The goal of defect tolerance is to bypass defective resources using test and diagnosis information. Since defects are device specific, this part of the design flow, mapping the application and bypassing defective resources, has to be device specific as well. However, the information required for such mapping, which is obtained only after test and diagnosis, is not available at the design time. Therefore, some parts of the application mapping phase have to be postponed from *design time* to the *test time*.

As parts of the design flow are shifted to the test time, we proposed a built-in self-mapping (BISM) approach to minimize per-chip customized mapping efforts. BISM allows the crossbar array to be configured by the on-chip interface circuitry and bypass defective resources. It also reduces physical design efforts in which detailed placement and routing will be per-

formed on-the-fly. In other words, only global placement and routing has to be completed at the design time and detailed configuration of individual crossbars (for logic mapping or signal routing) will be determined at the configuration time by BISM. The following BISM schemes are proposed depending on the defect density level.

3.1 Blind BISM

Since the two-dimensional crossbars used in crossbar array nano-architectures are complete structures, it is possible to find many different configurations (*legal configurations*) for a crossbar in order to implement a particular functionality (i.e. a specific logic mapping or signal routing, depending on the functionality of the crossbar). In a defect-free crossbar, any of these possible legal configurations can be used. However, in a defect-free crossbar some of these legal configurations use defective resources, and hence, are not valid. A *successful* configuration is a legal configuration which implements the desired functionality (logic mapping or signal routing) and does not use any defective resources. The goal of BISM is to automatically generate a successful configuration using on-chip resources and with the help of BIST and/or BISD schemes.

Blind BISM is the simplest form of BISM. In this scheme, a random configuration for the crossbar is generated on-the-fly and then application-dependent BIST is used to check whether this configuration is defect-free. The above process is repeated if BIST detects any fault in the generated configuration. The flowchart for Blind BISM is shown in Fig. 1. Note that application-dependent test is the simplest and fastest compared to diagnosis or application-independent test. Techniques for built-in application-dependent testing for reconfigurable systems have been presented in [14].

The reconfiguration and test method presented in [10] can be categorized as a Blind BISM scheme. Blind BISM is suitable for low defect densities in which it is expected that a randomly generated configuration is defect-free with a high probability such that few configuration retries are performed. Since no application-independent test is performed and no diagnosis is involved (neither application-independent nor application-dependent), blind BISM is very fast and effective for low defect-densities. The self-reconfiguration circuitry is also very simple and small.

3.2 Greedy BISM

When defect density is high, blind BISM approach becomes ineffective due to too many configuration re-

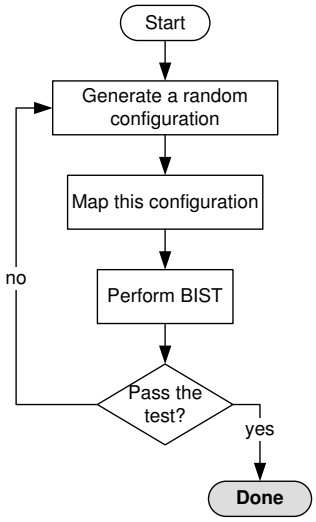


Figure 1. Blind BISM scheme

tries. In this case, *Greedy BISM* is performed as follows. It starts with a random configuration followed by BIST. If the configuration is failed, application-dependent BISD is performed to identify the defective resources utilized in the most recent configuration. The self-reconfiguration uses the diagnosis information to only bypass (reconfigure) the defective parts of that configuration. In other words, in each iteration, only a partial configuration, just for defective portion of the previous configuration, is generated. This process is repeated until the last configuration is defect-free. The flowchart for the Greedy BISM scheme is shown in Fig. 2. In this flowchart, the first generated configuration is also checked with BISD, as a variation of the approach described above.

The configuration generation algorithm (circuitry) for greedy BISM is a variation of that for blind BISM. In blind BISM, a complete configuration for the crossbar is generated at each retry phase. The input to the configuration generation process is the function to be implemented by the crossbar and the output is the configuration of the crossbar (on and off switches). However in greedy BISM, only a partial configuration corresponding to defective portion of the last configuration needs to be generated. The partial configuration algorithm for greedy BISM is similar to that for blind BISM except that it takes the partial function and the subset of the crossbar to be used for mapping the partial function as the inputs. In other words, the configuration generation circuitry for greedy BISM can also be used for blind BISM in which the partial function is the total function and the subset of crossbar for mapping is the entire crossbar.

Note that each retry phase in greedy BISM takes

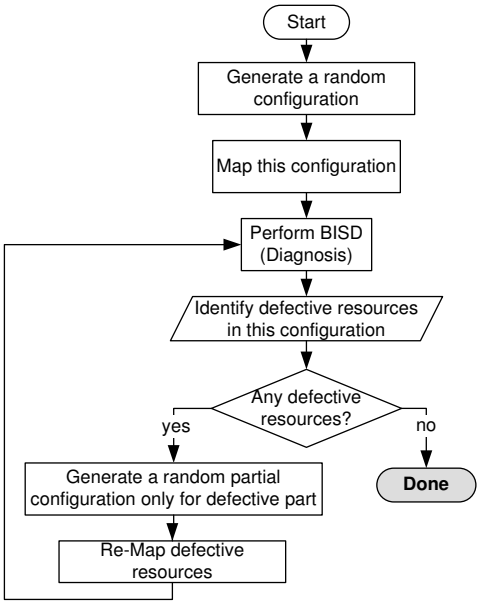


Figure 2. Greedy BISM scheme

longer than blind BISM since application-dependent BISD is used. The number of test configurations required for BISD is more than that for BIST. However, the number of retries is much smaller compared to blind BISM for higher defect densities, resulting in shorter mapping time in overall. Built-in application-dependent diagnosis techniques for reconfigurable systems have been presented in [15].

3.3 Hybrid BISM

As pointed out in the previous subsections, Blind BISM is suitable when defect density is low and it is expected that a successful configuration can be generated with few random retries. On the other hand, Greedy BISM uses diagnosis and partial configuration to perform a successive configuration generation. Therefore, when defect density is high and random retries (blind scheme) might not be successful, partial configuration and diagnosis will be effective.

The *Hybrid BISM* procedure is the combination of the Blind and Greedy BISM procedures. It is expected to work for all defect densities and also various defect density distributions across different crossbars in a nano-chip, i.e. ideal for both global and local defect density variations. In the hybrid BISM scheme, the BISM procedure initially starts with blind BISM. If it is not successful after a pre-defined number of retries, it automatically switches to greedy BISM. Therefore, it performs a thresholding on random reties and then switches to partial configuration and diagnosis. The hybrid approach can quickly configure the crossbars with smaller defect densities and also performs well on crossbars with higher defect densities. Also, when defect densities of various crossbars in the crossbar array nano-architecture are different, the hybrid BISM automatically performs blind BISM for crossbars with lower defect densities and greedy BISM for those with higher defect densities.

As described in Sec. 3.2, the configuration generation algorithms (circuitry) for blind and greedy BISMs can be the same in which blind configuration generation is a special case of greedy configuration generation. In this case, the same configuration generation scheme can be used for hybrid BISM as well.

3.4 Comparison of BISM Schemes

In order to evaluate the effectiveness of various BISM schemes described above, we consider the number of retries and reconfigurations for defect-free matching for different sizes of crossbars and defect density levels. *Defect density* is defined as the ratio (percentage) of defect-free resources (programmable switches) over the total number of resources in a crossbar. Defect-free matching is a typical configuration for an $n \times n$ crossbar which is used for both logic mapping [18] and signal routing [19]. In a defect-free perfect matching, all inputs of a crossbar are matched (one-to-one) to the outputs using defect-free switches.

Blind BISM. If defect density is d, then the probability that a given matching (containing n switches) in an $n \times n$ crossbar is defect-free can be calculated as $P_{matching} = (1-d)^n$. Here we assume that defects are randomly distributed in the crossbar. If the configurations for the matching are randomly generated, the probability that the first successful (defect-free) configuration is generated in the k^{th} retry is given by: $(1 - P_{matching})^{k-1} P_{matching}$. The expected number of retries is then calculated as follows:

$$\sum_{k=1}^{\infty} k(1 - P_{matching})^{k-1} P_{matching} = \frac{1}{P_{matching}} \quad (1)$$

Therefore, the expected number of retries for the Blind BISM scheme, $N_b(n)$, is calculated as follows.

$$N_b(n) = \lceil (1-d)^{-n} \rceil \quad (2)$$

Greedy BISM. In Greedy BISM approach, each retry reconfigures only the defective portion of the previous configuration. In order to obtain the expected number of retries for the Greedy BISM scheme,N_g, we use a recursive formulation. The goal is to calculate $N_g(n)$ which is the expected number of retries to match n

inputs to n outputs in an $n \times n$ crossbar with defect density d. Since precise diagnosis is used in each iteration of the BISM scheme, only defective resources of the partial configuration of iteration i are considered and configured in iteration $i+1$. Hence, if the size of matching for the partial configuration in the i^{th} iteration is m (m inputs need to be matches to m outputs, $m \leq n$), then the expected size of matching in $i+1^{th}$ iteration is $\lceil m \times d \rceil$. This is the number of defects in the previously generated matching which is diagnosed by BISD. Only this part of matching needs to be regenerated and remapped in the next iteration. When $m = 1$, the expected number of retries to map one input to one output is $\lceil 1/(1-d) \rceil$, since the probability of finding a defect-free switch is $1-d$. Therefore, we can use the following recursive formulation for $N_g(m)$:

$$N_g(m) = \begin{cases} 1 + N_g(md), & m > 1 \\ \left\lceil \frac{1}{1-d} \right\rceil, & m = 1 \end{cases} \quad (3)$$

Applying the recursive formulation k times, we have $N_g(m) = k + N_g(md^k)$. Obtaining k for $md^k = 1$,

$$N_g(m) = \log_d \frac{1}{m} + N_g(1) \quad (4)$$

Using this equation and the basis of the recursive formula, $N_g(N)$ can be found in non-recursive form:

$$N_g(n) = \left\lceil \frac{1}{1-d} \right\rceil + \left\lceil \frac{-\ln n}{\ln d} \right\rceil \quad (5)$$

Although the above equation describes the expected number of retries in the greedy BISM scheme, we also need to consider the number of test configurations required for diagnosis during each iteration. In blind BISM, only an application-dependent test is performed at each step. In greedy BISM, precise diagnosis is required which itself needs many configurations. The number of configurations for diagnosis is proportional to the diagnosis resolution, i.e. the number of faults that can precisely be located. There are n^2 switches in an nn crossbar. When defect density is d, the expected number of defective switches is n^2d. In an n by n matching, n switches are used. In the worst case scenario, all these n^2d defects in the crossbar can be in the matching resources, however, this number cannot exceed n. So, the maximum expected number of defects in a matching is $N_{defect} = min(\lceil n^2 d \rceil, n)$. The diagnosis flow should be able to precisely locate N_{defect} multiple defects. The number of test configurations to perform such diagnosis is $O(N_{defect})$. Therefore, the total number of steps in the greedy BISM scheme, considering reconfiguration retries and diagnosis configuration is:

$$N_{g_{total}}(n) = \left(\left\lceil \frac{1}{1-d} \right\rceil + \left\lceil \frac{-\ln n}{\ln d} \right\rceil \right) min(\lceil n^2 d \rceil, n) \quad (6)$$

Figure 3 shows the number reconfiguration retries for defect-free matching in 16×16, 32×32, and 64×64 crossbars for various defect densities using blind and greedy BISM. For Greedy BISM, the number of retry steps and total steps (retries and diagnosis) are shown. In blind BISM, the number of reconfiguration retries increases almost exponentially with the defect density (please note that the Y-axis is in logarithmic scale). The total number of steps in Greedy BISM has a negligible relation with defect density. As a result, for larger defect densities, Greedy BISM outperforms Blind BISM significantly.

Hybrid BISM, as a combination of Blind and Greedy BISMs, starts with Blind BISM and then switches to the Greedy scheme using a thresholding mechanism. By comparing Equations 2 and 6, or alternatively the graphs in Fig. 3, the optimum threshold for each size of crossbar can be obtained. The threshold must be set to the number of retries in which the curves for Blind BISM and Greedy BISM cross (Y-axis value). Below this threshold, the Blind BISM is more efficient and beyond it the Greedy BISM outperforms. Table 1 presents the optimum threshold for the number retries in the Hybrid BISM before switching to greedy approach. It also shows the corresponding defect density values for the crossing points. It can be seen that the switch from Blind to Greedy BISM for larger crossbars occurs at lower defect densities.

Size of crossbar	Retry threshold	Defect Density
12×12	48	27.5%
16×16	64	22.8%
32×32	96	13.3%
64×64	192	7.9%

Table 1. Retry threshold in Hybrid BISM

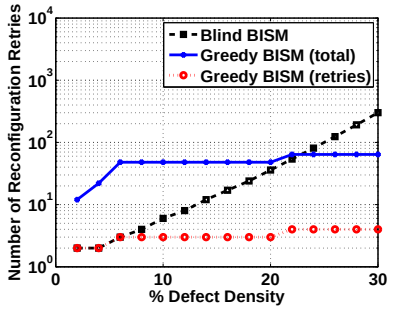
Finally, it needs to be mentioned that the BISM controller as well as the BIST/BISD controllers can be implemented in the reliable CMOS interface of hybrid nano-CMOS architectures. This provides better reliability and predictability for the proposed flow and the entire nano-architecture.

4 Conclusions

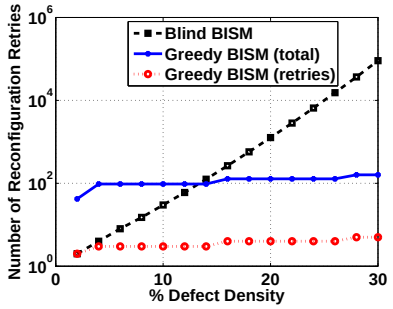
Defect and fault tolerance are inevitable for systems built using self-assembly processes. In this paper, we have presented a built-in self map (BISM) strategy in which the physical mapping of the designs into

programmable crossbar nano-architecture is performed on-the-fly using on-chip resources. This approach will reduce many complexities related to defect-aware design and test resulting in simpler and faster design and test flows, as well as reduced post-fabrication configuration time. BISM can effectively be exploited for the implementation of defect tolerance, for manufacturing defects, and fault tolerance, for lifetime operation defects. Various BISM schemes have been presented in the paper which can be tailored based on the defect density level and mapping (configuration) time.

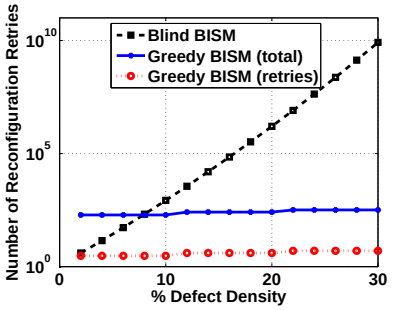
Future research directions include the implementation of BISM controll circuitry and simulation-based experimentations.



(a) 16×16



(b) 32×32



(c) 64×64

Figure 3. The number of configuration retries (a) 16×16 **crossbar (b)** 32×32 **crossbar (c)** 64×64

References

[1] M. Butts, A. DeHon, and S.C. Goldstein. Molecular Eletronics: Devices, Systems and Tools for Gigagate, Gigabit Chips. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 443–440, 2002.

[2] C.P. Collier, E.W. Wong, M. Belohradsky, F.M. Raymo, J.F. Stoddart, P.J. Kuekes, R.S. Williams, and J.R. Heath. Electronically Configurable Molecular-Based Logic Gates. In *Science*, volume 285, pages 391–394, 1999.

[3] Y. Cui and C.M. Lieber. Functional Nanoscale Electronics Devices Assembled Using Silicon Nanowire Building Blocks. In *Science*, volume 291, pages 851–853, 2001.

[4] A. DeHon. Array-Based Architecture for FET-Based, Nanoscale Electronics. In *IEEE Trans. on Nanotechnology*, volume 2, pages 23–32, 2003.

[5] S.C. Goldstein and M. Budiu. NanoFabrics: Spatial Computing Using Molecular Electronics. In *Proc. Int'l Symp. on Computer Architecture*, pages 178–189, 2001.

[6] M.B. Tahoori and S. Mitra. Defect and Fault Tolerance in Reconfigurable Molecular Computing. In *Proc. Field Custom Computing Machines*, pages 176–185, 2004.

[7] J. G. Brown and R. D. S. Blanton. CAEN-BIST: Testing the Nanofabrics. In *Proc. Int'l. Test Conf.*, pages 462–471, 2004.

[8] M.B. Tahoori. Defects, Yield, and Design in Sublithographic Nano-electronics. In *Proc. IEEE Int'l Symp. on Defect and Fault Tolerance of VLSI systems*, pages 3–11, 2005.

[9] Z. Wang and K. Chakrabarty. Using built-in self-test and adaptive recovery for defect tolerance in molecular electronics-based nanofabrics. In *Proc. Int. Test Conf.*, pages 477–486, 2005.

[10] R. Rad and M. Tehranipoor. SCT: an approach for testing and configuring nanoscale devices. In *Proc. VLSI Test Symp.*, pages 370–377, 2006.

[11] T. Rueckes, K. Kim, E. Joselevich, G.Y. Tseng, C.L. Cheung, and C.M. Lieber. Carbon Nanotube-based Nonvolatile Random Access Memory for Molecular Computing. In *Science*, volume 289, pages 94–97, 2000.

[12] A. DeHon, P. Lincoln, and J.E. Savage. Stochastic Assembly of Sublithographic Nanoscale Interfaces. In *IEEE Trans. on Nanotechnology*, volume 2, pages 165–174, 2003.

[13] X. Ma, D. Strukov, J. Lee, and K. Likharev. Afterlife for silicon: CMOL circuit architectures. In *Proc. IEEE Conf. on Nanotechnology*, pages 175–178, 2005.

[14] M. B. Tahoori. Application-dependent testing of fpgas. *IEEE Trans. on Very Large Integrated Systems*, 14(9):1024–1033, 2006.

[15] M.B. Tahoori. Application-dependent diagnosis of FPGAs. In *Proc. IEEE Int'l Test Conf.*, pages 645–654, 2004.

[16] M. Abramovici and C. Stroud. BIST-Based Detection and Diagnosis of Multiple Faults in FPGAs. In *Proc. IEEE Int'l Test Conf.*, pages 785–794, 2000.

[17] C. Stroud, S. Konala, C. Ping, and M. Abramovici. Built-in self-test of logic blocks in FPGAs (Finally, a free lunch: BIST without overhead!). In *Proc. VLSI Test Symp.*, pages 387 –392, 1996.

[18] H. Naeimi and A. DeHon. A Greedy Algorithm for Tolerating Defective Crosspoints in NanoPLA Design. In *Proc. Int'l Conf. on Field-Programmable Technology*, pages 49–56, 2004.

[19] J. Huang, M. B. Tahoori, and F. Lombardi. On the Defect Tolerance of Nano-Scale Two-Dimensional Crossbars. In *Proc. IEEE Int'l Symp. on Defect and Fault Tolerance of VLSI systems*, pages 96–104, 2004.