Application-Independent Defect-Tolerant Crossbar Nano-Architectures

Mehdi B. Tahoori
Electrical & Computer Engineering
Northeastern University
Boston, MA
mtahoori@ece.neu.edu

ABSTRACT

Defect tolerance is a major issue in nano computing. In this paper, an application-independent defect tolerant scheme for reconfigurable crossbar nano-architectures is presented. Architectural features are developed to reliably connect local defect-free subsets of crossbars in order to generate a defect-free architecture. It is also shown how to further reduce the area overhead associated with this flow by relaxing some constraints on the defect-free subsets. Experimental results show more than 9x reduction in the area overhead without any negative impact on the usability of modified defect-free subsets.

1. INTRODUCTION

A considerable amount of research is currently focused on developing nanoscale devices and alternative nanotechnologies to supersede conventional lithography-based CMOS technology. It has been shown that using bottom-up self-assembly techniques, it is possible to build nano devices, such as carbon nanotubes (CNTs) and silicon nanowires (NWs), without relying on lithography to define the smallest feature size [1, 2, 3, 4]. 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 Field Programmable Gate Arrays (FPGAs) [5, 6].

It is now very well known that ultra high density and ultra-small feature sizes at the nano-scale come with the expense of excessive defect densities, both at the manufacturing and during lifetime operation of nano devices [1, 2, 5, 6]. This increased defect density is mainly due to inherent non-determinism in chemical self-assembly processes used in nano fabrication.

The programmability of crossbar nano-architectures can be well exploited to implement defect and fault tolerant schemes for the design mapped into this architectures. After identifying defective resources using thorough test and precise fault localization, defects can be bypassed by post-fabrication configuration. However, conventional *application dependent* defect tolerant schemes are not very effective for nano-computing due to prohibitively large defect map and excessively increased post-fabrication customized design efforts. In our earlier publication, an alternative defect tolerant flow, the so-called *defect-unaware design flow*, is presented in which most design steps are unaware of the existence and the location of defects in the chip [7]. This is based on identifying universal defect-free subsets of crossbars within partially-defective crossbars that are used in the design flow.

This paper present an *application-independent* defect tolerant crossbar array in which defect-free subsets of the crossbars are reliably connected. The contributions of this paper are summarized as follows:

- A global application-independent defect tolerant architecture is presented. The proposed architecture is composed of *user crossbars* and *permutation crossbars*. User crossbars can be used as logic blocks, interconnect, or memory arrays, whereas the permutation crossbars provide connection among user crossbars.
- It is shown how to obtain an acceptable manufacturing yield for the implementation of the permutation crossbars.
- Finally, techniques to reduce the area overhead of the application-independent defect tolerance are provided.

The rest of the paper is organized as follows. In Sec. 2, some backgrounds on crossbar nano-architectures as well as the defect-tolerant design flow are provided. The proposed array-based defect-tolerant architecture is presented in Sec. 3. Techniques for area overhead reduction of the proposed defect tolerance are presented in Sec. 4. Finally, Sec. 5 concludes the paper.

2. PROGRAMMABLE CROSSBAR ARCHITECTURES

Two dimensional (2D) crossbars are the building blocks of reconfigurable molecular architectures. In these architectures, two layers of orthogonal nanowires or carbon nanotubes form the crossbars [8, 9]. At each intersection (*crosspoint*), there is a programmable non-volatile switch. Such $n \times n$ 2D crossbar can be represented by a *bipartite* graph [10]. In general, multiple pairs of orthogonal sets of nanowires can overlap with each other such that the crossbar get inputs to and produces outputs from each side (north, south, east, and west) [6].

Goldstein has proposed the chemically assembled electronic nanotechnology (CAEN) architecture called *NanoFabric* [6]. Nano logic arrays, also called *Nanoblocks*, implement a diode-resistor logic since crosspoints act as programmable diodes. In this non-inverting logic, inputs and their complements are given to nanoblocks and the output function and its complement are generated. DeHon has presented another array-based nano-architecture using *Programmable Logic Arrays* (PLAs) [5]. This architecture allows inversion by using nanowire FET devices as buffers. Logic functionality is achieved in the form of two-plane PLAs, implementing NOR-NOR logic.

In conventional application-dependent defect tolerance, defective resources are avoided in the physical design flow after identified using test and diagnosis. Although this approach is able to recover majority of defect-free resources, it has major shortcomings: the size of the defect map can be prohibitively large and most design steps have to be performed in a per-chip basis. This can also result in large and unacceptable performance variations for a same design mapped into different chips. All these make this approach unsuitable for high-volume production.

An *application-independent* defect tolerant design flow is presented in [7]. In this flow, defect tolerance is performed once and the same recovered set of resources are used for all applications. In the proposed flow, almost all design steps are unaware of the existence and the location of defects within the nano-chip. All design steps work with a universal defect-free subset of the chip called the *design view*. There is a *final mapping* phase at the end of physical design flow that makes the connection between the defect-free design view and the actual *physical view* of the nano-chip which contains actual defects. This is the only defect-aware step which has to be performed per chip.

For molecular crossbars, defect-free $k \times k$ crossbars within the partially-defective $n \times n$ crossbars ($k < n$) can be identified. The size of the maximum defect-free crossbar can be estimated using a sample of fabricated devices and this value (k) will be used for all chips manufactured in the same process environment. During the physical design, the original design will be mapped (placed and routed) into an array of $k \times k$ crossbars. Finding a defect-free $k \times k$ crossbar within a partially defective $n \times n$ crossbar can be modeled as finding the maximum biclique in the bipartite graph model of the defective $n \times n$ crossbar. As shown in [10, 7], nanowire and crosspoint defects can be modeled in the bipartite graph of the crossbar to obtain the graph model of the defective crossbar.

3. ARRAY-BASED DEFECT-FREE ARCHITECTURE

Although the defect-free subset of each crossbar extracted in the proposed design flow is universal (i.e. the size and structure of all subsets are identical for all crossbars), defect-free subsets of different crossbars cannot be easily connected in the crossbar array. Consider the example of two 6×6 crossbars shown in Fig. 1. crossbar A gets inputs from north and generates output in east direction. Crossbar B gets input from west and generates outputs in south direction. The input and output nanowires used in the 4×4 defect-free subsets, shown in bold, are not matched. This is because the information used for extracting the defect-free subset within each crossbar is *local*, which means that only the defects of the nanowires connected to that crossbar and the faults of the crosspoints inside that crossbar are used to identify the defect-free subset. As a result, the output nanowires of the first crossbar participating in its defect-free subset do not necessarily match with the input nanowires of the second crossbar participating in its defect-free subset, as illustrated in this example.

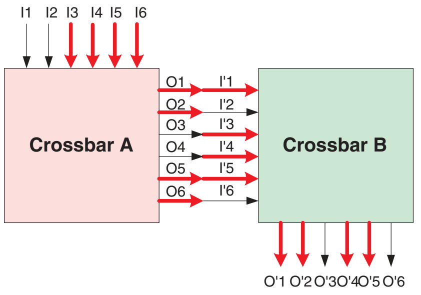

Figure 1: Non-matching defect-free subsets of two neighbor crossbars

In general, the defect-free subset locally obtained from each crossbar do not necessarily match with defect-free subsets of neighbor crossbars to form an array of defect-free crossbars. However, another crossbar can be used to make the matching between defect-free subsets of two crossbars. For the crossbars shown in Fig. 1, a crossbar is used to match the input and outputs nanowires of the defect-free subsets of those two crossbars, as shown in Fig. 2. This $n \times n$ crossbar, which makes the connection between k particular input nanowires to k particular output nanowires using a one to one matching, is called a *permutation crossbar*. In this figure, the permutation crossbar gets input from west and match them with east nanowires. The specific way that these k input nanowires are matched with these k output nanowires is not important. For example in Fig. 2, it is only important that the 4 inputs of the permutation crossbar, $\{O1, O2, O5, O6\}$, are matched in a one-to-one manner to its 4 output nanowires $\{I'1, I'3, I'4, I'5\}$. Any mapping between these particular inputs and outputs is acceptable, as long as it is defect-free. In Fig. 2, one possible one-to-one defect-free matching is shown.

In the application-independent defect-tolerant architecture, the crossbars are divided into two sets of *user crossbars* (UCB) and *permutation crossbars* (PCB). UCBs are used for the implementation of logic, programmable switch block, or non-volatile memory array. The design view of an $n \times n$ partially-defective crossbar used as a UCB is a $k \times k$ defect-free complete crossbar (biclique). The extraction of these defect-free bicliques and the manufacturing yield for a particular value of k are discussed in [7, 11]. UCBs are connected through PCBs. PCBs are transparent to the mapped design and the design flow. In other words, only UCBs exist in the design view and can be used for application mapping. PCBs provide defect-free matching between their input and output nanowires participating in the corresponding defect-free subsets of the adjacent UCBs to that PCB.

Note that the defect-free configuration of each PCB (the particular defect-free switches used for matching the required inputs and outputs of the PCB) is determined once the defect-free subsets of the UCBs are identified using fine-grained diagnosis information [12, 13]. Algorithms for finding a defect-free matching within a crossbar are presented in [14, 10]. Such algorithms can be used for determining the PCB configuration. This process, similar to finding defect-free $k \times k$ subsets of UCBs, is application-independent although has to be perform per crossbar.

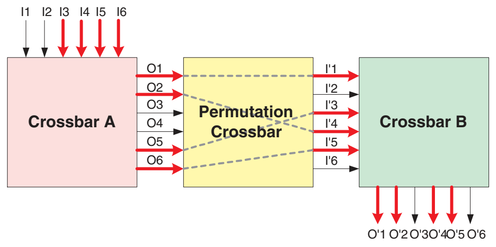

Figure 2: Using permutation crossbar in connecting two neighbor crossbars

Figure 3(a) shows this application-independent architecture made of arrays of $n \times n$ UCBs and PCBs. Note that from the manufacturing point of view, both UCBs and PCBs are the same; they are molecular crossbars. The corresponding design view, which is available for mapping applications to this platform, is shown in Fig. 3(b). The design view only consists of an array of $k \times k$ defect-free UCBs.

There are some implications of including PCBs in the architecture in terms of extra delay and power consumption that need to be considered in the design. However, PCB delay and power consumption parameters can be included as a part of UCB design parameters. Such UCB parameters adjustment makes the design flow totally transparent to existence of PCBs.

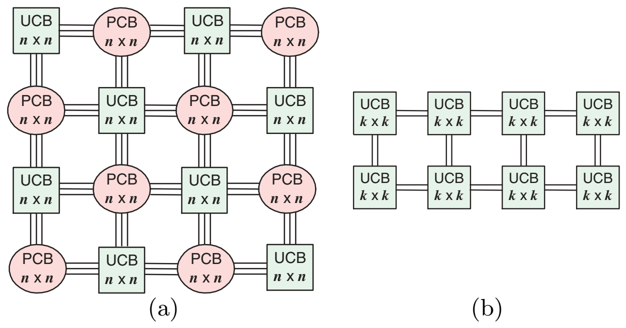

Figure 3: (a) Defect tolerant architecture using UCBs and PCBs (b) Design view

The size of defect-free subsets of UCBs (k) is chosen such that for the defect density d of the fabrication process, a required percentage (referred to as yield, y) of $n \times n$ fabricated devices have $k \times k$ defect-free subsets. Alternatively, if a particular size of the defect-free subset (to be used for application mapping) is desired, the minimum size of the fabricated crossbar (n) can be obtained based on d and y [11].

When a PCB is used for providing the one-to-one mapping between defect-free subsets of its neighboring UCBs, there should be at least one defect-free matching in the PCB to provide this connection. Intuitively, the probability of finding a defect-free matching (for PCBs) should be much higher than that for finding a defect-free biclique (for UCBs). This is because only k defect-free crosspoints are required for a defect-free matching whereas k^2 defect-free crosspoints are needed for a biclique.

Figure 4 compares the yields of finding a $k \times k$ biclique in an $n \times n$ UCB in the presence of switch open defects (up to 20% defect density) for various values of n and k. However, the probability of finding a defect-free matching in a PCB (for any arbitrary k inputs to any arbitrary k outputs in an $n \times n$ PCB) is always 100% when up to 30% of switches are defective (open defects). This confirms that the yield of PCBs is not a limiting factor for the overall yield of the crossbar array. In other words, the array yield is determined only by the yield of UCBs, the maximum defect-free subset of resources that can be used for application mapping.

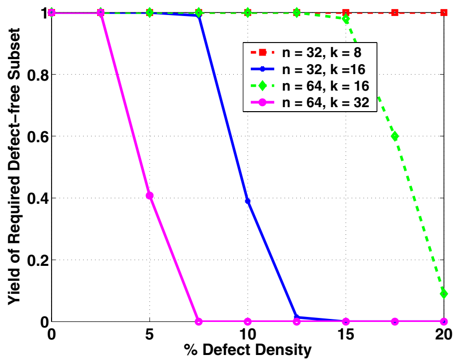

Figure 4: UCB yield for various values of n and k

4. REDUCING AREA OVERHEAD

Generally, application-dependent defect tolerance can recover and use more defect-free resources within the fabric compared to the application-independent approach. This is because the application-dependent flow is tuned for each particular application and defective chip. However, such better recovery (utilization) of defect-free resources comes at the expense of spending a considerable amount of design efforts per chip and application. Nevertheless, it is possible to hide some area overhead associated with the application-independent flow. By defining different forms of "$k \times k$ defect-free subsets" for logic and interconnect UCBs, it is possible to reduce the required size of the fabricated crossbars for these UCBs.

It needs to be mentioned that the following approaches are still application-independent, i.e. the size and the connection among defect-free subsets are identified before mapping any particular application to the crossbar array.

4.1 UCB Overhead Reduction

The structure of UCBs, as described in Sec. 3, is a complete yet smaller crossbar (complete $k \times k$ biclique). For many applications, a complete structure is excessive. Next it is described how to reduce the overhead of the application-independent flow by extracting different structures for defect-free subsets.

4.1.1 *Logic UCBs*

In logic mapping, only a random (arbitrary) defect-free matching is sufficient [14]. In this case, the structure (design view) of the logic UCBs can be similar to PCBs. This means that the size of the fabricated crossbar to be used as a logic UCB with a required size (k) can be greatly reduced compared to the general UCBs. Figure 5 compares the size of the fabricated crossbars to achieve defect-free 16×16 bicliques (general UCBs) and matchings (logic UCBs) for different open defect densities. Approaches to identify and extract defect-free matching from a defective crossbar are described in [14, 10]. With 20% open defect density, it is possible to extract a defect-free 16×16 matching from a 16×16 fabricated crossbar to implement logic whereas the size of the fabricated crossbar to achieve a 16×16 defect-free biclique is 126×126.

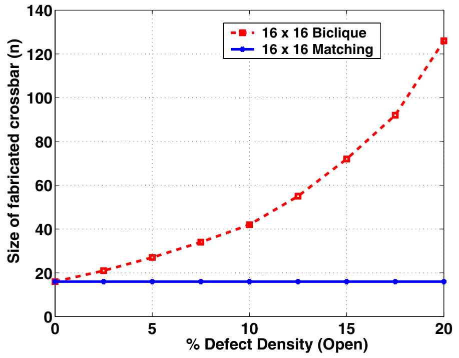

Figure 5: Size of the fabricated crossbar to achieve defect-free 16×16 biclique and matching

4.1.2 Interconnect UCBs

For the UCBs used as interconnect switch blocks, it might not be required to have a complete block in which each input is connectable (through programmable switches) to all outputs. In contemporary FPGAs, each input of the switch block is connectable to only few outputs. For example in Xilinx Virtex series, each single line is connectable to only 3 other single line in a 24×24 switch block [15]. These sparse switch block structures are able to provide the required routability for the logic block array with a reasonable routing delay and congestion. A $k \times k$ switch block structure in which each node is connected to exactly m other nodes through programmable switches is modeled by an *m-regular bi-graph*. In these structures, typically $m \ll k$.

Similar to an FPGA-like switch block structure, instead of extracting defect-free $k \times k$ bicliques for interconnect UCBs, it is possible to use defect-free m-regular bi-graphs where $m \ll n$. Since the number of edges (defect-free switches) in an m-regular bi-graph (mk) is much smaller than that in a complete bi-graph (k^2), the size of the fabricated crossbar (n) to yield an interconnect UCB of a required size (k) can be considerably reduced.

A heuristic greedy algorithm for finding the maximum m-regular subset of a bipartite graph is presented in Fig. 6. In the proposed algorithm, a variation of m-regular bi-graphs is extracted in which each node is connectable to at least (in contrast to "exactly") m nodes in the other partition.

The approach is based on converting this problem to the dual problem in the *complement* graph. Since the goal is to extract the nodes with degrees of at least m, nodes with degrees less than or equal to $n - m$ are selected in the complement graph. In the proposed heuristic, the nodes in the complement graph are sorted based on their degrees in the decreasing order and nodes with maximum degrees, along with their incident edges, are removed from the graph. If r nodes are removed from one partition (U), then the degree of nodes in the other partition (V) needs to be at most $n - (m + r)$ to be selected in the m-regular subset. Since

```
1 **Function** HasRegular($G(U, V, E), m$)
2   Obtain $\overline{G}(U, V, \overline{E}), \overline{E} = K_{|U|,|V|} - E$
3     Sort $U$ based on $d(u)$ in $\overline{G}$ (decreasing order)
4     Sort $V$ based on $d(v)$ in $\overline{G}$ (decreasing order)
5     $toggle \leftarrow true$
6     $reg_U \leftarrow false, reg_V \leftarrow false$
7     **Repeat**
8       **if** $toggle$ **then**
9         $u \leftarrow$ node in $U$ with maximum degree
10        **if** $d(u) > |V| - m$ **then**
11          $U \leftarrow U - \{u\}$
12          **for** each $v' \in V$ such that $(u, v') \in \overline{E}$ **do**
13            $d(v') \leftarrow d(v') - 1$
14          Re-sort $V$ accordingly
15        **else**
16          $reg_U \leftarrow true$
17      **else**
18        $v \leftarrow$ node in $V$ with maximum degree
19        **if** $d(v) > |U| - m$ **then**
20          $V \leftarrow V - \{v\}$
21          **for** each $u' \in U$ such that $(u', v) \in \overline{E}$ **do**
22            $d(u') \leftarrow d(u') - 1$
23          Re-sort $U$ accordingly
24        **else**
25          $reg_V \leftarrow true$
26      $toggle \leftarrow \neg toggle$
27    **Until** $U = \phi \vee V = \phi \vee (reg_U \wedge reg_V)$
28  **return** $|U| \times |V|$ as the largest $m$-regular subgraph
```

Figure 6: Algorithm for extracting m-regular subset

nodes with highest degree are directly removed from U (and alternatively V), $|U| - m = (n - r) - m = n - (m + r)$. Deleting the nodes with the maximum degree allows us to remove a maximum number of edges with a minimum node removal. This increases the chance of finding a large set of nodes with the required degree, $n - (m + r)$, in the subset of remaining nodes. In the node removal process, the algorithm alternates between two partitions such that the difference in the number of removed nodes from two partitions is at most one. This process terminates when the degrees of all nodes in the reduced complement graph become at most $n - (m + r)$. This means that each node is connected to at least m nodes in the other partition and hence, the algorithm works correctly.

Removing nodes with maximum degrees modifies the degrees of remaining nodes in the graph. By implementing U and V sets as *binary heaps*, deletion and reordering can be performed in a logarithmic time. If $|U| + |V| = O(n)$, the initial sorting takes $O(n \log n)$ (lines 3-4). The repeat-until loop (lines 7-27) will be executed n times in the worst case, since at each step at least one node is removed from U or V. The execution of the body of the loop takes $O(\log n)$ assuming that the binary heap is used for the implementation of U and V sets (deletion and reordering). Therefore, the worst case time complexity of this algorithm is $O(n \log n)$.

In the presence of switch short defects, the area overhead is noticeably higher since there should not be any defective short switches between the k input nanowires and k output nanowires participating in the defect-free m-regular subset. Figure 7 compares the area overhead of 16×16 biclique and 8-regular structure for both open and short defects. For 20% defect denisty, only a 42×42 crossbar is required to be fabricated to yield a defect-free 16×16 8-regular structure. This is 9x smaller than the size of the crossbar to achieve a defect-free 16×16 biclique. Therefore, using m-regular defect-free subsets for interconnect UCBs provides a sufficient degree of routability with much reduced area overhead.

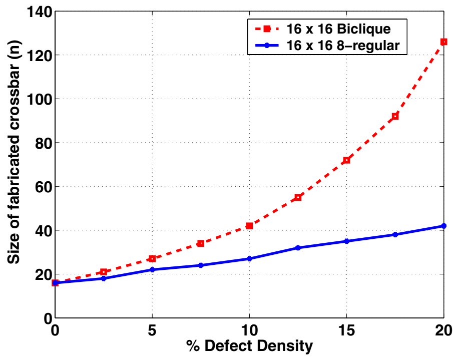

Figure 7: Area overhead of defect-free 16×16 biclique and 16×16 8-regular structures

4.2 PCB Overhead Reduction

In the architecture presented in Fig. 3 (a), there is a PCB between each two UCBs. However, it might be possible to directly match the outputs of a UCB to the inputs of its neighboring UCB without using a PCB if the k output nanowires of a crossbar participating in its defect-free subset are exactly the same as the k defect-free input nanowires of the neighbor crossbar.

Since the value of k is fixed for all fabricated crossbars within the fabric, the actual maximum defect-free subsets of all crossbars in the fabric can be extracted and compared for a possible match of k signals between neighboring crossbars. Consider that the maximum defect-free subset (biclique, m-regular, or matching, depending on the particular use) of a crossbar A is $k_1 \times k_2$ where $k_1, k_2 \geq k$. The maximum defect-free subset of the neighboring crossbar B is $k_1' \times k_2'$ ($k_1', k_2' \geq k$). If the intersection between k_2 defect-free outputs of crossbar A with k_1' defect-free inputs of crossbar B is at least k, these two (UCB) crossbars are adequately matched and can be directly connected without a PCB. Figure 8 shows an example in which two UCBs are required to have 4×4 defect-free subsets. However in UCB1, $k_2 = 6$ and in UCB2 $k_1' = 5$, and the overlap between defect-free inputs and outputs (shown in bold) is 4. Therefore, no PCB is required to connect these two UCBs. As the size of the actual defect-free subset of the crossbar becomes larger than the required defect-free size, the possibility of such direct match increases. This process, although performed in a per-chip basis, is still application-independent, i.e. before mapping applications to the chip.

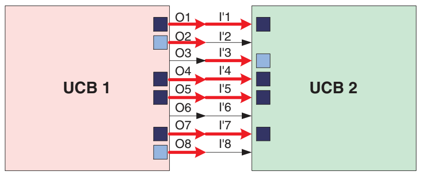

Figure 8: Connecting UCBs without a PCB

5. SUMMARY AND CONCLUSIONS

Defect tolerance is an integral part of nano-computing to control the excessive number of defects introduced by self-assembly fabrication processes used in nanotechnologies. The fine-grained reprogrammability of molecular crossbar architectures can be exploited to implement defect tolerant schemes.

In this paper, an application-independent defect tolerant architecture is presented in which a universal (application-independent) defect-free subset of fabricated resources are extracted and used in the design flow. It is shown how to connect locally-extracted defect-free subsets of the crossbars to achieve an array-based defect-free architecture. Finally, techniques to reduce the area overhead of the proposed defect-tolerant flow have been presented. These techniques result in more than 9x area overhead reduction by using different defect-free subsets for logic and interconnect crossbars.

6. 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] A. Bachtold, P. Harley, T. Nakanishi, and C. Dekker. Logic Circuits with Carbon Nanotube Transistors. In *Science*, volume 294, pages 1317–1320, 2001.

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

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

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

[7] M.B. Tahoori. A Mapping Algorithm for Defect Tolerance of Reconfigurable Nano Architectures. In *Proc. IEEE Int'l Conf. on Computer Aided Design*, pages 668–672, 2005.

[8] 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.

[9] 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. In *Nanotechnology*, volume 14, pages 462–468, 2003.

[10] 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.

[11] 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.

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

[13] Z. Wang and K. Chakrabarty. Using Built-In Self-Test and Adaptive Recovery for Defect Tolerance in Molecular Electronics-Based Nanofabrics. In *Proc. Int'l. Test Conf.*, pages 477–486.

[14] 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.

[15] Xilinx. In *Xilinx Datasheets*, www.xilinx.com, 2004.