Application-Independent Defect Tolerance of Reconfigurable Nanoarchitectures

MEHDI B. TAHOORI
Northeastern University

Self-assembled nanofabrication processes yield regular and reconfigurable devices. However, defect densities in this emerging nanotechnology are higher than those in conventional lithography-based VLSI. In this article, we present an application-independent defect tolerant design flow to minimize customized postfabrication design efforts to be performed per chip. In this flow, higher level design steps are not needed to be aware of the existence and the location of defects in the chip. Only a final mapping step is required to be defect aware. Application independence of this flow minimizes the number of per-chip design steps, making it appropriate for high volume production. We also present two mapping algorithms, recursive and greedy, which make the connection between defect-unaware design steps and the final defect-aware mapping step. Experiments show that the results obtained by the greedy algorithm are very close to the exact solutions. Using these algorithms, we analyze the manufacturing yield of molecular crossbars under different defect distribution models. We report on the size of the minimum crossbar to be fabricated such that a defect-free crossbar of the desirable size can be found with a guaranteed manufacturing yield.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms: Design, Reliability

Additional Key Words and Phrases: Defect tolerance, nanotechnology, reconfigurable architectures

1. INTRODUCTION

Conventional lithography-based CMOS technology faces serious challenges due to its fundamental physical limits such as ultra-thin gate oxides, short channel effects, doping fluctuations across the chip, and increasingly difficult and expensive lithography. 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 [Butts et al. 2002; Bachtold et al. 2001; Cui and Lieber 2001; Huang et al. 2001]. Chemically self-assembled structures, as the building blocks for molecular-scale computing, are by their nature very regular and therefore well

Author's address: Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115.

suited to the implementation of regular arrays similar to Field Programmable Gate Arrays (FPGAs). Reprogrammable nanoarchitectures are currently being investigated [DeHon 2003; DeHon and Wilson 2004; Goldstein and Budiu 2001; Ziegler and Stan 2002].

Nanofabrication process yields nanowires which are a few atoms long in diameter. 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, resulting in more defects. The need for defect and fault tolerance for this technology has already been stressed [Butts et al. 2002; Collier et al. 1999; DeHon 2003; Goldstein and Budiu 2001].

The reconfigurability of this nanotechnology is well suited for the implementation of defect and fault tolerant mechanisms. After identifying defective resources in the chip using test and diagnosis, they can be bypassed by postfabrication configuration. This approach is similar to a conventional memory defect tolerant scheme which utilizes spare rows and columns. Nevertheless, the efficient implementation of this flow is not straightforward. As will be explained later in Section 3, applying a per-chip customized configuration in the entire physical design flow causes serious problems for high volume production. These include prohibitively large amounts of information that must be stored in order to locate defect-free resources within the chip (defect map) and excessively increased postfabrication efforts per chip (including test, diagnosis, and design).

In our previous work [Tahoori 2005b], we presented a methodology to analyze the manufacturing yields of molecular crossbars in the presence of defects. In another publication [Tahoori 2005a], we proposed a greedy mapping algorithm to generate the defect map. In this article, we extend our previous work by presenting an alternative defect tolerant flow, the *defect-unaware design flow*, in which most physical design steps (and also all higher-level design steps) are unaware of the existence and the location of defects in the chip. The key idea is to identify universal defect-free subsets within the partially defective chip to be used in the design flow. These subsets are called *universal* because these defect-free subsets are supposed to be identical for all manufactured chips with the same level of defect density. As a result, the size of the defect map as well as the amount of per-chip customized design time can be drastically reduced. Since this defect tolerant flow is *application-independent*, that is it is not tailored for each particular mapped design, the amount of per-chip design effort is minimized. Therefore, it is applicable for high volume production. The only part of the physical design that needs to be aware of the location of the defects within the fabric is the *final mapping* in which the used resources in the universal subsets are mapped to the actual defect-free resources in the partially defective fabric using defect map information.

The amount of excessive (spare) resources, also called *redundancy*, is a key factor that affects both the manufacturing yield and the cost of nanoscale systems. The amount of required redundancy is a function of manufacturing defect density, the required manufacturing yield, the size of the device, the architecture, and how the device will be used at the design steps. We try to analyze

the amount of required redundancy in nanoarchitectures by examining two-dimensional (2D) programmable crossbar structures. In the presence of some defects, an $n \times n$ fully populated crossbar can still be used as a $k \times k$ ($k < n$) crossbar at the architectural level. We analyze the manufacturing yield of molecular crossbars under this scenario which is compatible with the proposed design flow. Manufacturing yield is redefined as the probability of finding a defect-free crossbar within the original fabricated crossbar. Based on this analysis, the minimum required redundancy to achieve a defect-free crossbar with a desirable size at the specified yield is estimated. Results for crossbars of various sizes are reported under different types of faults and defect distribution models.

We also present and compare two different approaches to identify the maximum defect-free subsets and construct a compact defect map. One of the algorithms is an exact method, whereas the other one is a heuristic approach for the same problem with faster runtime. These algorithms are also utilized in the yield analysis method. One of them is a recursive algorithm to solve the decision version of the problem. The second one is a very fast greedy mapping algorithm with $O(n \log n)$ time complexity.

Since the molecular crossbar is the main building block in CNT-based nanotechnology to implement logic, programmable interconnects, and memory arrays, the main focus of the article is on the analysis of these crossbars for defect and fault tolerance. Nevertheless, we model and analyze the interconnect (nanowire) defects in our analysis by considering the crossbar defects and its surrounding nanowires.

The rest of the article is organized as follows. In Section 2, some background on CNT-based crossbar architectures is presented. The proposed design flow is described in Section 3. The yield metric and estimation method are presented in Section 4. The algorithms for yield estimation and mapping are explained in Section 5. The yield results as well as a comparison of the proposed heuristic algorithm versus the exact method are discussed in Section 6. Finally, Section 7 concludes the article.

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 [Rueckes et al. 2000; Chen et al. 2003]. These nanowires or carbon nanotubes are aligned in parallel rows using a bottom-up self-assembly process. This fabrication scheme is totally different from top-down lithography-based patterning processes which are used in CMOS manufacturing. At each intersection, also called *crosspoint*, there is a programmable switch which is nonvolatile. A one-time programmable switch consisting of a monolayer of redox-active rotaxanes sandwiched between metal electrodes is demonstrated in [Collier et al. 1999]. In the *closed* state, the switch acts as a diode. It can also be irreversibly *opened* by applying an oxidizing voltage across the device. In this architecture, the configuration of a crosspoint is performed by applying a higher voltage (programming voltage) to the intersecting nanowires. In other words, the same signal lines can be used as

configuration circuitry. Unlike programmable crosspoints in conventional VLSI, which are an order of magnitude larger than a wire crossing area, crosspoint switches in this nanotechnology take the same area as that of a wire crossing.

There are three crossbar-based architectures in the literature: NanoFabric, PLA-based, and CMOS-compatible crossbars. These architectures are described next.

In Goldstein and Budiu [2001], a chemically-assembled electronic nanotechnology FPGA-like architecture called *NanoFabric* has been proposed. Nano logic arrays, also called *Nanoblocks*, implement a diode resistor logic (DRL) since crosspoints act as programmable, diodes. Since only AND and OR logic can be implemented by DRL, that is, no inversion, inputs, and their complements are given to nanoblocks, and the output function and its complement are generated. Signal restoration is performed by using a molecular latch at the output of crossbars.

DeHon has presented another array-based nanoarchitecture using *Programmable Logic Arrays* (PLAs) [DeHon and Wilson 2004; DeHon 2003; DeHon et al. 2003]. This architecture allows inversion by using nanowire Field Effect Transistor (FET) devices as buffers. Logic functionality is achieved in the form of two-plane PLAs. Each plane consists of a 2D crossbar, implementing programmable OR array, followed by a restoration and selective inversion array. Therefore, NOR-NOR logic is used in this architecture.

The third architecture is a CMOS-compatible crossbar memory array proposed by Nantero Inc. called *NRAM* [Nantero 2005]. In this architecture, everything but nanoelectromechanical switches are implemented in CMOS using conventional lithography processes (CMOS-compatible fabrication). The programmable switches are realized by a belt of carbon nanotubes (monolayer fabric of nanotubes). The same technology can also be used to implement programmable logic and interconnection network.

Defect tolerance of nanoarchitectures has been discussed in the literature. Hierarchical sparing is used in [DeHon 2003] to achieve defect tolerance. In DeHon and Wilson [2004], manufacturing yield of the proposed array-based architecture is analyzed based on the nanowire defect density, the yield of the stochastic decoder, and the stochastic buffering. In Naeimi and DeHon [2004], the problem of logic mapping in a defective crossbar is modeled by matching a bipartite graph and a greedy matching algorithm with linear time complexity for logic mapping. While defect-free matching is sufficient for logic mapping (i.e., when the crossbar is used as the programmable logic device), it cannot be used for signal routing and defect-free memory arrays. The goal of this article is to identify defect-free structures that are amenable for signal routing and memory arrays for as well as for logic mapping.

In CNT nanotechnology, the molecular crossbar is the main building block. An example of a 4×4 crossbar is shown in Figure 1(a). The crossbar consists of two sets of orthogonal nanowires. The horizontal nanowires are the inputs, whereas the vertical nanowire are the outputs. There is a programmable switch at each intersection (crosspoint).

An $n \times n$ 2D crossbar can be represented by a *bipartite* graph $B = (U, V, E)$ [Huang et al. 2004], as illustrated in Figure 1(b). U represents the set

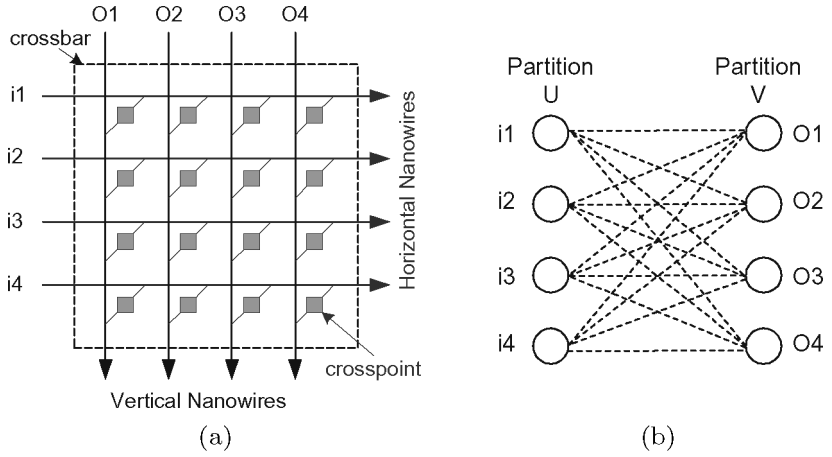


Fig. 1. (a) 4×4 2D nanoscale crossbar (b) bipartite graph representation.

of input nanowires, and V represents the output nanowires; E models the programmable switches in the crossbar. In a defect-free crossbar, there is a switch at the intersection of each input and output nanowire, that is, $|E| = |U| \times |V|$. This graph is a *complete* bipartite graph. In the presence of defects, some nanowires or switches become unusable, and the corresponding bipartite graph is no longer complete. However, it might be possible to find a maximum defect-free subset of the crossbar which is complete. The maximum complete subset of a bipartite graph is called the maximum *biclique*. A *matching* T is a set of edges such that no two edges share the same vertex. If an edge (v_i, v_j) is in the matching, then vertices v_i and v_j are said to be matched. A *perfect* matching of a graph is a matching such that all vertices are matched.

3. DEFECT-UNAWARE DESIGN FLOW

In the conventional adaptive defect-tolerant flow, the existence and the location of defective elements are identified using test and diagnostic steps. This information is stored in the *defect map* which identifies the usability of the (programmable) elements of each manufactured chip. Defect tolerance is achieved by avoiding defective resources in the physical design flow using the defect map. Particularly, placement and routing are performed based on this defect map. This flow is shown in Figure 2(a). In this *defect-aware* (application-dependent) design flow, higher-level design steps, such as logic and architecture design, might need to be modified to incorporate defect information. An example of an application-dependent defect tolerance is the work presented in Naeimi and DeHon [2004].

However, there are several problems associated with this approach.

—First, the size of the defect map for the entire fabric can be prohibitively large (10^{12} to 10^{14} devices per chip). Moreover, it would be impossible to store and use this defect map for online testing and fault tolerance purposes.

—Second, the entire design flow has to be customized per chip. Most steps have to be repeated for each manufactured chip, adding the design time

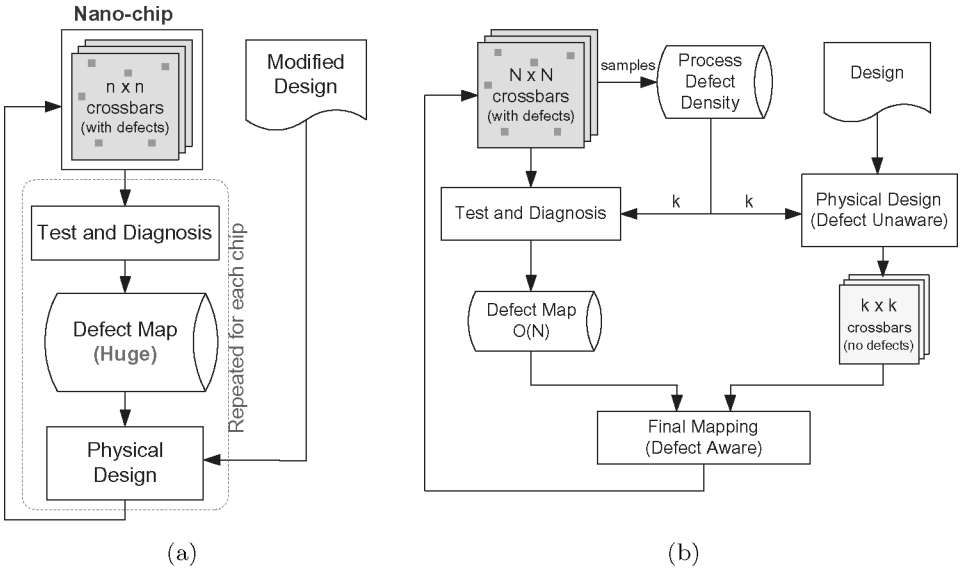


Fig. 2. (a) Defect-aware design flow (b) Defect-unaware design flow.

(∼ hours) to the postfabrication test time (∼ seconds to minutes). Note that in the current design flow, the design effort is independent of the number of manufactured chips. However in this defect-aware flow, most design steps have to be performed on a per-chip basis.

—Third, per-chip design customization results in large and unacceptable performance variations for the same design mapped onto different chips. This is because the defect maps of different chips can be considerably dissimilar. Since the outcome of physical design is highly dependent on the defect map, the performance characteristics of the same design mapped onto various chips might be completely different.

—Fourth, this scheme results in a radical shift from conventional design methods in which all logic and physical design tools have to be redeveloped.

Due to these fundamental problems, this traditional approach cannot be used for high-volume production of nanochips.

In contrast to the defect-aware design flow, we propose a *defect-unaware* design flow to tolerate defects. Some of the drawbacks of the defect-aware flow are due to the fact that this method is *application-dependent*, that is, defects are handled or a per-application basis. However, our proposed approach is an *application-independent* flow, which means that the defects are bypassed before mapping any particular application to the platform. In other words, defect tolerance is performed once, and the same recovered platform can be used for all applications. In our proposed flow, almost all design steps, from architecture design to the last step of the physical design, are unaware of the existence and the location of defects within the nanochip. In other words, all design steps work on a *design view* of the chip which is defect-free. 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 the actual defects.

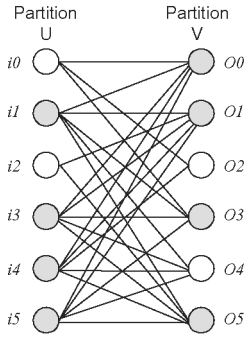
The key idea in this defect-unaware flow is to identify *universal* defect-free subsets of resources within the original partially defective fabric. All design steps use these universal defect-free subsets as the physical implementation devices. Hence, conventional architectural, logic, and physical design tools can be reused. These defect-free subsets are called universal because the size of these subsets is identical for all chips fabricated in the same process environment (similar defect densities). An additional step, after placement and routing, is required to map the used resources within the universal subset (design view) onto the actual resources within the original device (physical view). This is the only defect-aware step in the entire design flow which has to be performed for each manufactured chip. For molecular crossbars, our goal is to identify defect-free $k \times k$ crossbars within the original partially defective $n \times n$ crossbars which work as smaller devices ($k < n$). The final mapping step maps the used resources within $k \times k$ crossbars onto the actual defect-free resources within the original $n \times n$ crossbars.

Figure 2(b) shows the defect-unaware physical design flow. Test and diagnosis procedures identify the size and the location of maximum defect-free $k \times k$ crossbars within partially defective fabricated $n \times n$ crossbars. 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. All crossbars with a defect-free subset not smaller than $k \times k$ are considered as good (pass) devices, otherwise they are marked as failing devices. The information regarding the location of defect-free subsets is stored in a compact defect map. Compared to the defect map in the original flow which stores the location of each defective switch within each crossbar, this one stores only the position of the defect-free $k \times k$ crossbars within the original crossbar.

During the physical design, the original design will be mapped (placed and routed) into a network of $k \times k$ crossbars. A final defect-aware mapping step is required to remap the used resources within $k \times k$ crossbars into the actual defect-free resources within partially defective $n \times n$ crossbars using the defect map information.

Note that the value of k cannot be determined in a per-chip based, otherwise some design steps will have to be customized for each chip. Therefore, the value of k is estimated based on the defect density level of the fabrication process. This is the critical point in making the design efforts independent of the number of manufactured chips. Moreover, the value of k is fixed for all manufactured parts in the same fabrication environment. For example, if the value of k is set to 16, then, all crossbars for which $k \geq 16$ are considered as good (passing) devices, and those with $k < 16$ are considered as failing devices. It needs to be mentioned that the desirable value of k is set such that a large (and acceptable) fraction of manufactured devices passes the test.

Another advantage of this approach is the reduction in the size of the defect map. In the conventional defect-aware flow, the size of the defect map is $O(n^2)$ for an $n \times n$ crossbar. This is because one bit information is required for each switch in order to specify whether it is defect-free or not. In the proposed



(a)

	i_0	i_1	i_2	i_3	i_4	i_5
O_0	1	1	0	1	1	1
O_1	0	1	1	1	1	1
O_2	1	0	0	0	1	0
O_3	1	1	0	1	1	1
O_4	0	1	0	1	0	1
O_5	0	1	1	1	1	1

(b)

V = [1 1 0 1 0 1]
H = [0 1 0 1 1 1]

(c)

Fig. 3. (a) 6 × 6 Crossbar (b) complete $O(n^2)$ defect map (c) reduced defect map: two vectors of $O(n)$.

defect-unaware flow, we only need to identify the location of the $k \times k$ defect-free crossbar within the partially-defective $n \times n$ crossbar. Therefore, only two binary vectors of size n, the vertical and horizontal vectors, need to be stored. Each bit position in the vertical (horizontal) vector corresponds to a row (column) in the original crossbar and specifies whether this row (column) participates in the defect-free biclique ($k \times k$ crossbar). These two $O(n)$ vectors are sufficient to precisely locate the defect-free subset within the original crossbar. Consequently, the size of the defect map is reduced from $O(n^2)$ to $O(n)$.

Figure 3 shows how the reduced defect map can be interpreted. In Figure 3(a), the bipartite graph model of a defective crossbar is shown. The missing edge corresponds to a defective switch. The complete defect map, as used in the original defect-aware flow, is shown in Figure 3(b) in which a "1"("0") entry corresponds to a usable (unusable) switch. The reduced defect map containing only two vectors of size n is shown in Figure 3(c). Using these two vectors, the defect-free 4×4 subset, corresponding to the complete 4×4 subgraph can be uniquely identified. The horizontal vector identifies the nodes in the input partition (U) participating in the defect-free subset, whereas the vertical vector corresponds to the output partition (V). This subset is highlighted in the graph model, as well.

The nanoarchitecture can be viewed as an array of crossbars. The *global view* of this architecture considers only the interconnection of crossbars, whereas the *local view* deals with the individual resources within each crossbar. The main focus of this article is the local view of the nanoarchitecture in which the structure of a crossbar is fully analyzed. Considering the local view (used in detailed placement and routing), the problem of performance variation due to detailed mapping, placement, and routing can be solved with the proposed design flow. Nevertheless, we discuss some issues related to defect and fault tolerance in the global view of this nanoarchitecture in Section 6.3.

In the next section, we present a yield metric based on this design flow in which we show how we can set the value of k such that a large fraction (high yield) of manufactured crossbars contain defect-free subsets of minimum size k. In Section 5, two algorithms for identifying the defect-free subset within the

fabricated crossbar, based on the information obtained from test and diagnostic steps, are presented. These algorithms can also be used to estimate the value of k from process defect density information using a sample of manufactured chips. The reduced $O(n)$ defect map will be generated as one of the outputs of this algorithm. The generated defect map is also used in the final mapping phase. Since this step has to be performed for each manufactured chip, its runtime has to be minimized. Therefore, we propose a fast greedy algorithm for this purpose.

4. YIELD METRIC

In this section we present a yield analysis framework which is compatible with the design flow presented in the previous section. Specifically, this framework is used to analyze the yield and the expected size of the defect-free subsets within the partially defective crossbars.

We have addressed the matching problem in defective crossbars in an earlier publication [Huang et al. 2004]. In the matching problem, the goal is to find a routing between m distinct input signals to m output signals through an $n \times n$ crossbar using m nonincident defect-free switches ($m \leq n$). However, defect-free matching, which is sufficient for logic mapping, is not adequate for signal routing in the general form since the selective connectivity is lost. From an algorithmic point of view, there is an exact solution for a defect-free matching problem-based network flow algorithm with polynomial time complexity $O(n^{\frac{5}{2}})$ [Huang et al. 2004]. The goal here is to find a defect-free $k \times k$ crossbar in a partially-defective $n \times n$ ($k < n$) crossbar at a specified defect density level. Based on the design flow presented in Section 3, the following yield metric is defined:

Yield Metric $Y^d_{n,k}$: The probability of finding a biclique (defect-free crossbar) of size $k \times k$ in an $n \times n$ crossbar when the defect density is d.

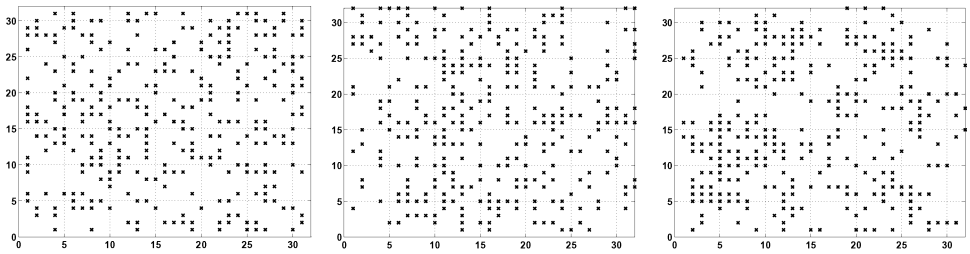
Based on this yield metric, we can identify the minimum size of a fabricated (partially defective) crossbar, $n \times n$, such that a defect-free crossbar of size $k \times k$ can be found with the yield of y when the defect density is d. In other words, $Y^d_{n-1,k} < y$ and $Y^d_{n,k} \geq y$.

We consider both *clustered* and *unclustered* defect distribution models. In the unclustered model, the defect probability distribution over the crossbar is uniform. In other words, the probability of defect at each location (e.g., crosspoint or nanowire) is p, the defect density. However, in reality, defects tend to cluster together and the distribution in not uniform [Meyer and Pradhan 1989]. We have considered this clustering effect in a model in which the defect probabilities in the regions near defect clusters are higher than other regions. This probability is also inversely proportional to the distance from the existing defect clusters. Formally, if N_d defects already exist (are injected) in the crossbar, the defect probability at location i, P_i, is calculated as

$$P_i \propto \sum_{j=1}^{N_d} \left(\frac{1}{d_{ij}}\right)^{\alpha}, \tag{1}$$

Clustering Defect Injection(N, d, α)
$N_d \leftarrow 1$
Inject the first defect randomly
while $N_d < N.d$ **do**
for each location i **do** (defects already injected in locations j)
$P_i \leftarrow \sum_{j=1}^{N_d} \left(\frac{1}{d_{ij}}\right)^{\alpha}$
Normalize P_i s.t. $\sum_{i=1}^{N} P_i = 1$
$r \leftarrow$ random(0,1)
choose k s.t. $\sum_{i=1}^{k} P_i < r \leq \sum_{i=1}^{k+1} P_i$
Inject defect at location k
$N_d \leftarrow N_d + 1$

Fig. 4. Clustering defect injection algorithm.



(a) Unclustered (b) Clustered $\alpha = 1$ (c) Clustered $\alpha = 2$

Fig. 5. Defect injection in a 32 × 32 crossbar using different defect distribution models.

where d_{ij} is the distance between switch i and j, and α is the clustering parameter. Note that a larger α corresponds to more clustering.

In defect injection based on this clustering model, if the defect density is d and there are N resources (crosspoints and nanowires) as candidates for defect injection, a total of $N.d$ defects will be injected. The algorithm for defect injection based on the clustering defect distribution model is given in Figure 4. Figure 5 shows an example of defect injection patterns in a 32 × 32 crossbar with 30% defect density using unclustered and clustered models.

4.1 Fault Injection Method

Faults are introduced in the complete bipartite graph $B = (U, V, E)$, representing the defect-free crossbar, resulting in a new bipartite graph $B' = (U', V', E')$, corresponding to the defective crossbar. The problem of finding a defect-free crossbar (of size $k \times k$) within a partially defective one can be expressed as finding a biclique of the desirable size (k) in bipartite graph B'.

In this work, we consider the faults within the crossbar as well as the interconnect faults surrounding the crossbar (outside the crossbar). By considering the interconnect (nanowire) faults surrounding each crossbars, faults in the entire interconnect structure can be modeled.

Switch stuck-open faults (*within crossbar*). A switch stuck-open fault corresponds to a missing switch at the crosspoint. Hence, the faulty bipartite graph

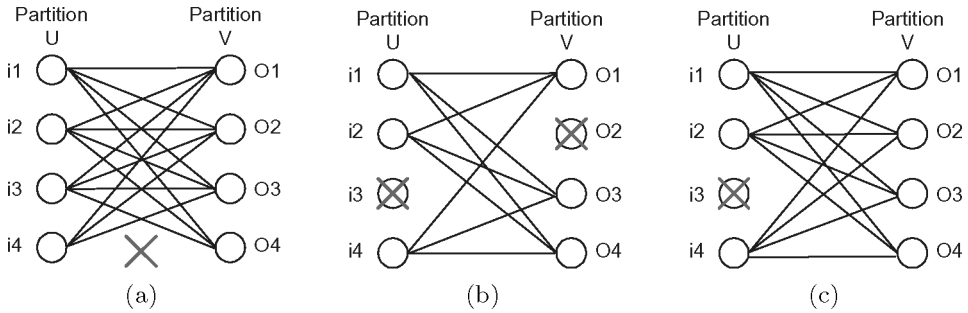


Fig. 6. (a) Switch Open ($i4$, $O4$), (b) Switch Short ($i3$, $O2$), (c) Nanowire $i3$ Open.

B' can be obtained by simply deleting from B the edges corresponding to the faulty switches.

Switch stuck-closed faults (*within crossbar*). If there is a stuck-closed fault on a switch corresponding to edge (u, v), the horizontal nanowire $u \in U$ and the vertical nanowire $v \in V$ are shorted together and both become unusable. Therefore, B' can be obtained by removing the nodes u and v from B along with all edges incident to these nodes ($U' = U - \{u\}, V' = V - \{v\}$). In this case, some fault-free switchings connected to u or v become unusable as well. Note that this situation is different from the effect of switch stuck-closed faults when defect-free matching is required [Huang et al. 2004].

Nanowire open fault (*within and outside crossbar*). In some architectures, the precharge/evaluation logic is used at the output stage of molecular crossbars [DeHon and Wilson 2004]. As a result, broken nanowires are of no use at all and can not be used to carry signals. In this case, all switches connected to a nanowire with an open fault become unusable. So B' is obtained by deleting the faulty node (broken nanowire) from B.

Nanowire bridging fault (*within and outside crossbar*). In case of a nanowire bridging fault, two (or more) nanowires are shorted together and both (all) of them become unusable. When constructing the faulty bipartite graph B', all nodes corresponding to the shorted nanowires and all incident edges to these nodes are removed from the original graph B.

Figure 6 shows the graph model of a 4×4 crossbar in the presence of some faults. As can be seen from this example, short faults (nanowires and switches) clearly have a more severe effect on the usability of crossbars than switch open faults.

The fault injection method used in this article is based on the approach presented in Huang et al. [2004]. Depending on the defect density d and the particular fault model, there are FP fault patterns and each corresponds to a different B'. For example, in the case of switch stuck-open faults, there are $\binom{n^2}{\lceil n^2 d \rceil}$ possible fault patterns. Since the number of possible fault patterns can be quite large, we only consider a statistical sample of them. The yield metric $Y^d_{n,k}$ is estimated as the ratio of the number of simulated fault patterns with a minimum biclique of size $k \times k$ over the total number of simulated fault patterns.

```
1 Function HasBiclique(G(U, V, E), k1, k2)
2     if k1 ≤ 0 ∨ k2 ≤ 0 then
3       return TRUE
4     if |U| ≤ k1 ∨ |V| ≤ k2 then
5       return FALSE
6     U1 ← {u|u ∈ U, d(u) < k2}
7     V1 ← {v|v ∈ V, d(v) < k1}
8     U2 ← {u'|u' ∈ U, d(u') = |V|}
9     V2 ← {v'|v' ∈ V, d(v') = |U|}
10    U ← U − (U1 ∪ U2); V ← V − (V1 ∪ V2)
11    k1 ← k1 − |U2|; k2 ← k2 − |V2|
12    u ←node with minimum degree in U
13    V' ←{nodes connected to u in V}
14    return HasBiclique(G(U − {u}, V, E),k1,k2) ∨ HasBiclique(G(U − {u}, V', E),k1 − 1,k2)
```

Fig. 7. Recursive biclique algorithm.

5. ALGORITHMS

In this section, we present two different approaches to find the maximum defect-free crossbar within a partially defective crossbar. The first approach is a variation of the exact method, while the second one is a greedy heuristic method. These algorithms can be used for the yield analysis step as explained in Section 4. Moreover, they are the basis for the final mapping phase and the generation of the reduced defect map (Section 3).

5.1 Recursive Biclique Algorithm

Given the location of defect-free switches and nanowires in the crossbar obtained from test and diagnostic procedures, the graph model of the defective crossbar can be formed. The goal is to find (and locate) the maximum defect-free $k \times k$ crossbar within the original crossbar. This problem corresponds to finding the maximum biclique in a bipartite graph.

Finding the maximum biclique in a bipartite graph is an NP-complete problem [Doherty et al. 1999]. However, we solve a *decision* version of this problem which is less complex compared to the original *optimization* version. Instead of finding the maximum biclique in $G(U, V, E)$, we solve the following decision (Yes/No) problem: "Does $G(U, V, E)$ have a biclique of size $k_1 \times k_2$?"

Note that in this problem $k_1 \times k_2$ is not necessarily the size of the maximum biclique of G. Using the fault injection approach presented in Section 4.1, by setting $k_1 = k_2 = k$, the desirable probability of $Y^d_{n,k}$ can be obtained. The recursive algorithm shown in Figure 7 is developed for solving the decision problem.

The induction basis is checked in lines 2–5. Bicliques of sizes $0 \times k_2$ and $k_1 \times 0$ can be always found (lines 2-3). Similarly, the size of any biclique cannot be larger than the size of the original graph (lines 4–5).

If there is a node in U whose degree in less than k_2, it cannot participate in the biclique and hence, can be excluded from the graph during the search. The set of these nodes are represented by $U1$. Moreover, if there is a complete node (i.e., connected to all nodes in V), it can always be included in the biclique. Hence, we can remove this node from U and decrease k_1 by one. The set of

```
1 Function Biclique(G(U, V, E))
2   Obtain Ḡ(U, V, Ē), Ē = K_{|U|,|V|} − E
3   Find maximum independent set in Ḡ
4     Sort U based on d(u) in Ḡ (decreasing order)
5     Sort V based on d(v) in Ḡ (decreasing order)
6     U^b ← φ, V^b ← φ, flag ← TRUE
7     Repeat
8       U^b ← U^b ∪ {u|u ∈ U, d(u) = 0}, U ← U − U^b
9       V^b ← V^b ∪ {v|v ∈ V, d(v) = 0}, V ← V − V^b
10      if flag then
11        u ← node in U with maximum degree
12        U ← U − {u}
13        for each v' ∈ V such that (u, v') ∈ Ē do
14            d(v') ← d(v') − 1
15        Re-sort V accordingly
16      else
17        v ← node in V with maximum degree
18        V ← V − {v}
19        for each u' ∈ U such that (u', v) ∈ Ē do
20            d(u') ← d(u') − 1
21        Re-sort U accordingly
22      flag ← ¬flag
23    Until U = φ and V = φ
24   return U^b × V^b as the maximum biclique
```

Fig. 8. Greedy biclique algorithm.

such nodes is represented by $U2$. The similar cases for the nodes in the second partition, V are also considered (lines 6–11).

The induction step is performed as follows. Consider an arbitrary node $u \in U$. This node is either included in the biclique or not. If it is not included, we need to find a biclique of size $k_1 \times k_2$ in G$(U - \{u\}, V, E)$. If this node is included in the biclique, the nodes of V that can participate in the biclique are those connected to u. So, if we call this set V' we should be able to find a biclique of size $k_1 - 1 \times k_2$ in G$(U - \{u\}, V', E)$. In order to reduce the number of recursions, u is chosen as the node with the minimum degree. This way, the size of V', which directly affects the search space, is minimized. In other words, if the answer is supposed to be FALSE, we will be able to find it out with the minimum number of iterations.

5.2 Greedy Mapping Algorithm

Since finding the maximum biclique in a bipartite graph is known to be an NP-complete problem, here we present a heuristic greedy algorithm for finding the maximum biclique. Our approach is to convert this problem to the problem of finding the maximum independent set in the *complement* graph. The outline of the proposed greedy algorithm is shown in Figure 8. The complement of a graph G is a graph $\overline{G}$ with the same set of vertices such that two vertices of $\overline{G}$ are adjacent if and only if they are not adjacent in G. An *independent set* S in a graph G is a subset of nodes that are disconnected, that is, there are no edges between any two nodes in an independent set: $\forall u, v \in S,\ (u, v) \notin E(G)$. The

maximum independent set is an independent set with the maximum number of nodes.

Even in the presence of defects (defect density $< 30\%$), the corresponding bipartite graph model of the crossbar is still dense, that is, $|E| = O(n^2)$. Consequently, the complement graph would be sparse and, therefore we can efficiently use a heuristic approach for finding the maximum independent set in the complement graph. This is the main motivation behind converting the maximum biclique problem into the maximum independent set problem in the complement graph.

The proposed heuristic works as follows. Nodes with zero degree (i.e., the number of edges connected to that node) can always participate in the independent set. Our objective is to remove some nodes along with their incident edges from the complement graph such that more nodes with zero degree can be found in the reduced graph. Our heuristic is based on removing the nodes with the maximum degree since it is assumed that they will not participate in the maximum independent set. 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 independent set in the remaining nodes. Since the complement graph is sparse and nodes with the maximum degree in the reduced graph are removed, we hope that the set of independent nodes with zero degree in the reduced graph is very close to the maximum independent set.

First, the complement graph is formed (line 2). The nodes in each partition are sorted based on their degrees in the complement graph in decreasing order (lines 4-5). At each iteration, nodes with zero degree are added to the solution list (lines 8-9). In this pseudocode, the set of nodes from U and V participating in the maximum independent set are denoted as U^b and V^b, respectively.

Bicliques with square shapes are preferred, that is, if the biclique obtained by this algorithm is $k_1 \times k_2$, it is preferred that k_1 is very close to k_2. The presented heuristic approach (removing nodes with the highest degrees) finds an independent set (biclique) with minimum node removal. In other words, this algorithm tries to maximize $k_1 + k_2$, the number of nodes in the biclique (independent set). Since the number of edges (switches) in the biclique is $k_1.k_2$, the number of edges in the biclique is maximized when $k_1 \approx k_2$ for a fixed $k_1 + k_2$.

To obtain a maximum-edge biclique, our heuristic alternates between U and V when removing the nodes with the highest degrees. Hence, at every other iteration, nodes with the highest degree are removed from U (lines 10–15), whereas in the alternating iteration, nodes with the maximum degree are removed from V (lines 16–21). Since $|U| = |V|$, at the end of the execution of the algorithm, $|U^b| = |V^b| \pm 1$. Consequently, the resulting biclique has a square shape and hence contains a maximum number of edges.

5.2.1 *Proof of Correctness.* At each iteration of the repeat-until loop (lines 7–23), one node with the maximum degree will be removed either from U or V. Let's denote the set of nodes removed from U and V as U^r and V^r, respectively. The invariant of the algorithm can be defined as follows.

Invariant. *During the execution of the algorithm, $U^b \cup V^b$ forms an independent set of $\overline{G}$.*

PROOF. At the beginning of the algorithm (line 6), U^b and V^b are initialized to the empty sets, and hence, the invariant holds. At each iteration of the repeat-until loop, nodes with zero degree from $U - U^r$ and $V - V^r$ are added to U^b and V^b, respectively. Let's denote the set of nodes removed from U by the end of the ith iteration as U_i^r. We define V_i^r similarly. Since $U_i^r \subseteq U_{i+1}^r$ and $V_i^r \subseteq V_{i+1}^r$, $d(u)\ in\ \overline{G}(U - U_i^r, V - V_i^r, \overline{E}) \geq d(u)\ in\ \overline{G}(U - U_{i+1}^r, V - V_{i+1}^r, \overline{E}), u \in (U - U_{i+1}^r) \cup (V - V_{i+1}^r)$. Therefore, if $d(u) = 0$ in the ith iteration, it is independent from all nodes in $(U - U_i^r) \cup (V - V_i^r)$. Consequently, any node u' to be added to U^b or V^b in the $i + 1$th iteration is also independent from u. This proves the invariant. □

Based on this invariant, at the end of the execution of the algorithm, $U^b \cup V^b$ forms an independent set in the complement graph, $\overline{G}$, and hence a biclique in the original graph, G.

5.2.2 *Runtime Analysis.* Removing nodes with maximum degrees modifies the degrees of remaining nodes in the graph. Therefore, the ordering of nodes based on their degrees has to be refreshed at each step. 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 4-5). The repeat-until loop (lines 7–23) 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 $O(n \log n)$.

6. EXPERIMENTAL RESULTS

6.1 Yield Analysis Results

The objective in defect tolerance is to extract the maximum defect-free block from a partially defective manufactured device. Alternatively, if a $k \times k$ defect-free crossbar is required, we find the smallest $n \times n$ crossbar ($n > k$) to be fabricated in a manufacturing environment with defect density d. For the fixed values of k and d and desirable yield of y, we find an n such that $Y_{n,k}^d = y$ but $Y_{n-1,k}^d < y$. Typically, we set $y = 100\%$. Such evaluations can be performed based on the yield analysis framework as discussed in Section 4 and are utilized in the proposed defect tolerance flow to make the connection between the design view and the physical view (Section 3).

Experimental results for various sized defect-free crossbars in the presence of stuck-open faults are presented in Figure 9. In these experiments, the number of simulated fault patterns, $FP_n^{d,total}$, ranges between 200 and, 5,000 depending on the size of the crossbar (n). The results are presented for three different defect distribution models, unclustered and clustered (with $\alpha = 1, 2$). For each

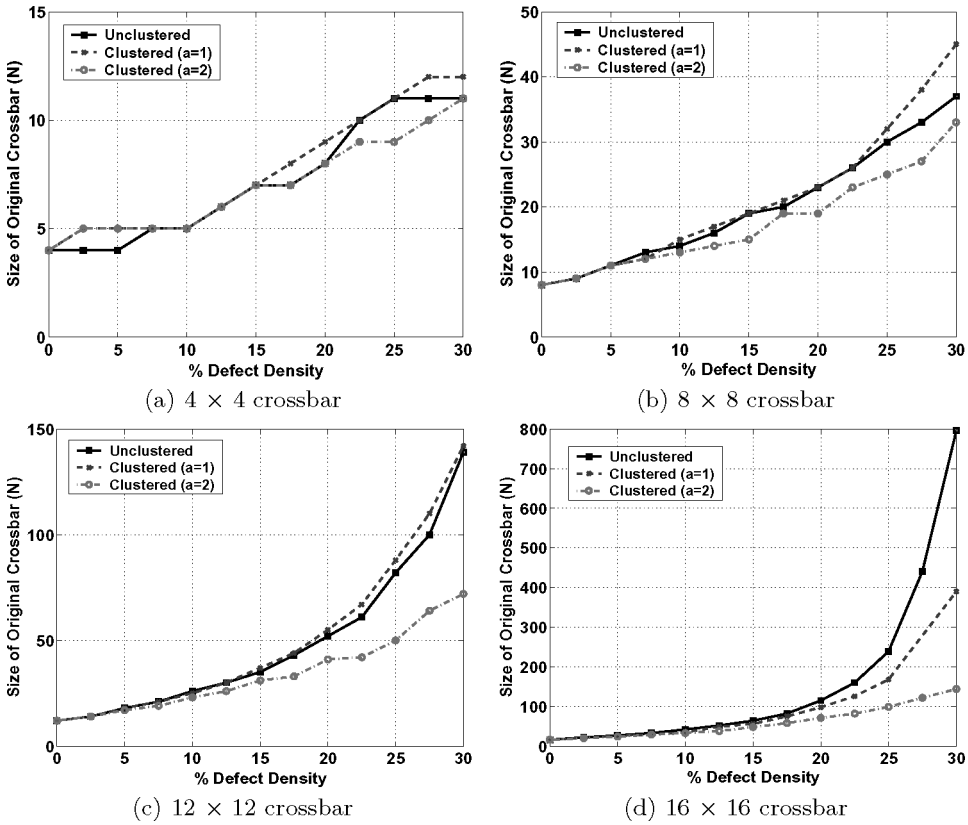


Fig. 9. The size of the original crossbar (n) required for obtaining the desirable defect-free crossbar (k) in the presence of stuck-open faults (yield = 100%).

defect density, the minimum size of the original crossbar (n) is given such that the desirable defect-free crossbar can always be found (yield = 100%).

Figure 10 shows the same data in a different format. In this figure, the amount of area overhead required to achieve the desirable defect-free subset is reported. The overhead is normalized to the area of the defect-free subset, that is, n^2/k^2. Note that the y-axis is in the logarithmic scale. As can be clearly seen in this figure, the amount of redundancy required to obtain smaller defect-free subsets is much smaller. Also, the overhead associated with the clustered defect distribution model is much lower than with the unclustered one. This figure shows that, for any fixed defect density, the area overhead increases exponentially with the size of the required defect-free crossbar. Therefore, it might be more economical to tailor the defect-unaware steps of the design flow to map the design into a set of smaller crossbars rather than larger crossbars. In other words, it costs less to use (map the design into) a large number of small crossbars rather than a small number of large crossbars. However, the impact on the routability of the crossbar array needs to be carefully considered, as well. Hence, there is a minimum (threshold) on the size of the crossbar such that,

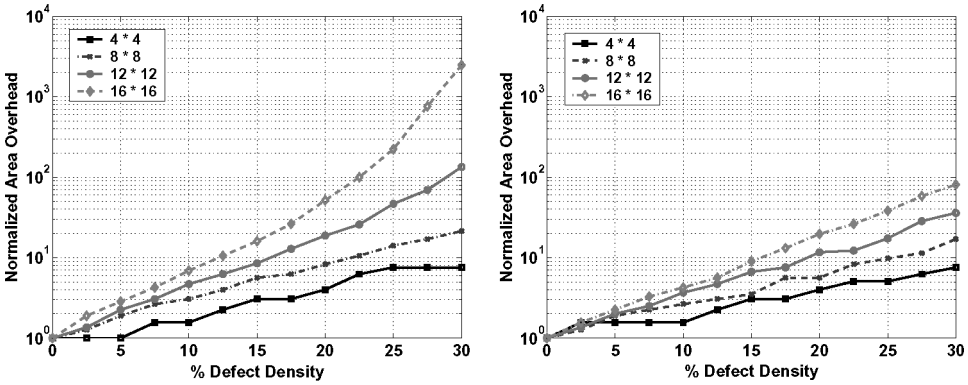


Fig. 10. Area overhead (redundancy) required to achieve defect-free crossbars with different sizes (yield = 100%).

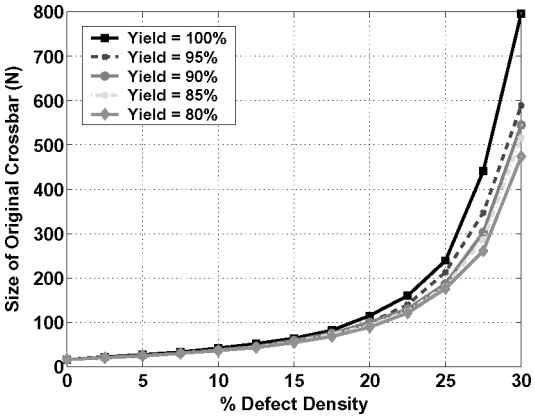


Fig. 11. Size of the minimum fabricated crossbar required for findings 16 × 16 crossbar with various yields (unclustered defect distribution).

below that threshold, the routability could be negatively affected. Figure 11 shows the size of the minimum fabricated crossbar required to achieve a defect-free 16 × 16 crossbar for various yields. The defect distribution is unclustered.

The situation for short defects is different from open defects. If there is a stuck-closed fault, both nanowires intersecting at the faulty switch position become unusable. This corresponds to missing vertices in the graph model instead of missing edge as in the case of open defects. As a result, the probability of finding a defect-free crossbar in the presence of stuck-closed faults is a function of the number of faulty switches rather than the defect density, which is the case for switch stuck-open faults.

The difference between stuck-open and stuck-closed faults is well depicted in Figure 12(a). In this figure, the probabilities of finding a defect-free 16 × 16 crossbar within various sized defective crossbars with defect density of 2.5% for both stuck-open and stuck-closed faults are plotted. The defect distribution

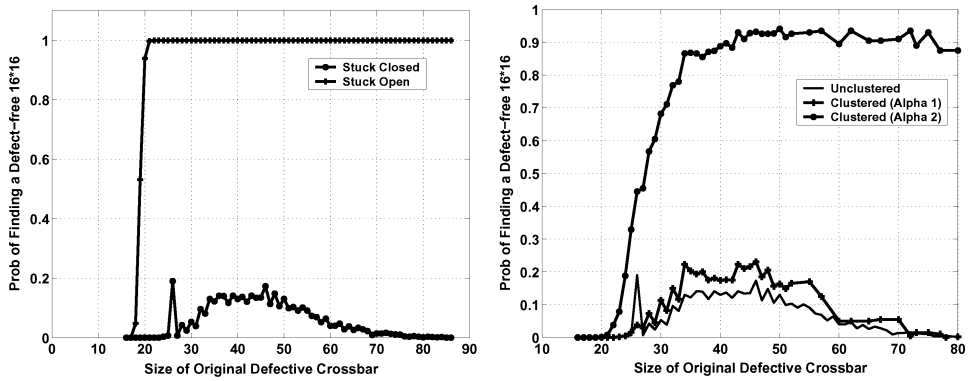


Fig. 12. Finding a defect-free 16 × 16 crossbar within defective crossbars (d = 2.5%): (a) open and short faults (b) short faults.

model is unclustered. The probability of finding a defect-free crossbar with a fixed size in the presence of stuck-open faults is monotically increasing if the size of the original defective crossbar is increased (assuming that the defect density is also fixed). However, for stuck-closed faults, this probability decreases at some point. This is due to the fact that as the size of the original defective crossbar increases (with a fixed defect density), the number of faulty switches will also increase.

Unlike stuck-open faults, the effect of different defect distribution models on the probability of finding a defect-free crossbar in the presence of stuck-closed faults is quite significant. Figure 12(b) shows the probability of finding a defect-free 16 × 16 crossbar within defective crossbars with defect density of 2.5% using different defect distribution models. As can be seen in this figure, the probability for the clustered distribution model with $\alpha = 2$ is much higher than other models and converges to more than 90% (while the other two converge to zero).

Fortunately, the stuck-closed faults are much less likely than stuck-open faults as shown in DeHon [2003]. Moreover, in the fabrication process, it is possible to tailor the chemical synthesis technique to decrease the probability of having switch stuck-closed faults at the expense of increasing the total number of switch faults [Goldstein and Budiu 2001]. In this case, we expect to have a much higher defect density of stuck-open faults than stuck-closed faults, thus resulting in a better defect tolerance overall.

6.2 Comparison of Recursive and Greedy Algorithms

Here we compare the accuracy and runtime of the two algorithms presented in Section 5. As explained in Section 5.1, the recursive algorithm is an exact approach, which gives the actual maximum biclique. However, the greedy algorithm is a heuristic approach with much reduced runtime in which the maximum biclique is not guaranteed. Figure 13 shows the results obtained by

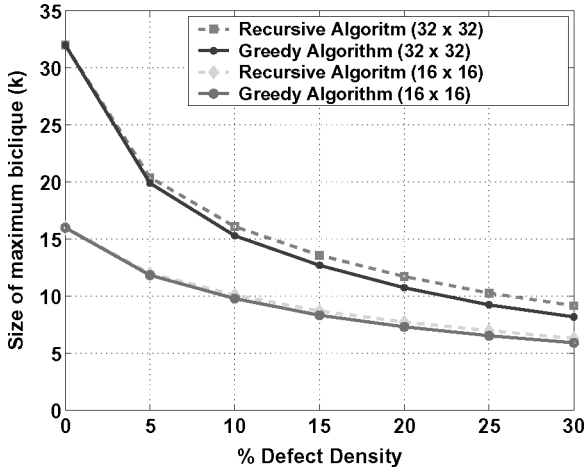


Fig. 13. Greedy algorithm vs. recursive algorithm.

Table I. Normalized Execution Time for Greedy and Recursive Algorithms

Defect Density	16 × 16		24 × 24		32 × 32	
	Greedy	Recursive	Greedy	Recursive	Greedy	Recursive
5%	1.0	3.7	3.8	111.4	9.9	3258.4
10%	1.6	9.1	5.8	260.6	12.6	5127.4
15%	1.9	12.3	6.9	205.7	14.8	3617.4
20%	2.1	11.3	6.5	208.6	16.5	2337.9
25%	2.4	9.9	7.2	133.6	16.5	1520.6
30%	2.6	9.5	7.8	103.7	17.7	866.3

these two methods for 16 × 16 and 32 × 32 crossbars, using defect densities (stuck-open defects) up to 30%. For each data point, 1,000 crossbars with the given defect density are randomly generated, and the average sizes of the maximal bicliques obtained by these two methods are reported. As can be seen in this figure, the difference between the recursive (exact) method and the greedy algorithm is almost negligible.

Table I shows the runtime comparison of these two approaches for three different sizes of crossbars, 16 × 16, 24 × 24, and 32 × 32. The runtime numbers are normalized to the run of the greedy algorithm for a 16 × 16 crossbar with 5% defect density (the smallest runtime number). With these crossbars, the greedy algorithm is up to 406 times faster than the recursive (exact) algorithm, and this speedup exponentially increases for larger crossbars. As shown in Table I, the runtime of the recursive (exact) approach grows exponentially with the size of the crossbar. However, the runtime of the greedy approach grows almost linearly with crossbar size. Another observation is that the runtime of the greedy algorithms grows monotically with the defect density. This is because the efficiency of the heuristic has a direct relation with the sparseness of the complement graph, that is, as the defect density increases, the algorithm requires that more nodes be removed before termination. However, the runtime of the recursive approach does not necessarily increase with the defect density.

When the defect density is very low, there are many nodes that are connected to all nodes in the other partition (i.e., satisfying the conditions of lines 8–9 in the pseudocode of Figure 7). However, when the defect density is high enough, there are many nodes connected to less than k nodes (i.e., satisfying the conditions of lines 6–7 in the pseudocode of Figure 7). This is why the runtime for very low and very high defect densities is smaller.

A defect map is generated as a byproduct of these algorithms. As a result, no extra time is required to obtain the defect map (i.e., the execution times in Table I include defect map generation). As stated in Section 3, the size of the reduced defect map is only $2n$ (the vertical and horizontal vectors).

6.3 Defect Tolerant Crossbar Array

The nanoarchitecture is based on an array of molecular crossbars. The *global view* of this architecture considers the connections among crossbars and how the design is partitioned and mapped into crossbars. The *local view* focuses on individual crossbars and the resources within each crossbar and how the crossbar resources are utilized for a particular mapped design. The floorplanning, global placement, and global routing phases of the physical design use the global view, whereas the detailed placement and routing are based on the local view. As stated in the introduction, the main focus of this article is defect tolerance in the local view of the nanoarchitecture. Nevertheless, here we discuss some defect tolerance issues related to the global view of the nanoarchitecture.

In order to achieve a global defect tolerant architecture, one option is to generate a defect-unaware (application-independent) global architecture which consists of an array of $k \times k$ defect-free crossbars. The size of these crossbar arrays needs to be the same for all the chips manufactured in the same fabrication environment. In other words, the value of k has to be constant for all crossbars in the chip. Therefore, k has to be chosen such that that all crossbars in the chip have defect-free subsets of minimum size k. This might result in excessive area overhead and smaller values for k. Moreover, in this scheme, the chips containing crossbars with very small defect-free subset (smaller than expected k) will be thrown away, and this affects the overall yield of manufactured chips.

The other option is to use an application-dependent flow for the global view and use the proposed defect-unaware flow only for the local view. In this scheme, the crossbars on the chip with defect-free subset smaller than k are marked as unusable and will not be used for application mapping. As a result, the size a usable crossbar array might vary from chip to chip. The advantage of this approach is its reduced area overhead (i.e., larger values of k) compared to the application-independent global defect tolerance, as explained previrvsly. This comes with the cost of using application-dependent global mapping per chip. Since the complexity of global mapping (placement and routing) is less than local mapping (detailed placement and routing), the amount of per-chip customization of this mixed globally-defect-aware locally-defect-unaware defect tolerance flow is not high.

7. SUMMARY AND CONCLUSIONS

Defect densities in self-assembly enabled nanotechnology are significantly higher than those in the lithography-based CMOS technology. Defect tolerant techniques are therefore essential for the designs realized using this nanotechnology in order to achieve an acceptable level of manufacturing yield.

In this article, we have proposed a defect-unaware design flow for defect tolerance of reconfigurable molecular devices. The presented application-independent flow has many advantages over the defect-aware adaptive defect tolerance (application-dependent), including a considerable reduction in the defect map size, negligible per-chip customized design effort, and predictability in the performance of mapped designs. We have also investigated defect tolerance of the two-dimensional crossbar, which is used as the basic building block for logic, interconnect, and memory in various nanoarchitectures. We have presented a yield analysis method based on the proposed defect-unaware flow in which the probability of finding a defect-free subset within a partially defective fabricated crossbar is identified. We have presented algorithms to identify the maximum defect-free crossbar with the partially defective fabricated crossbar. These algorithms can be used in yield analysis as well as in the final mapping and defect map generation steps.

REFERENCES

BACHTOLD, A., HARLEY, P., NAKANISHI, T., AND DEKKER, C. 2001. Logic circuits with carbon nanotube transistors. *Science*. *294*, 1317–1320.

BUTTS, M., DEHON, A., AND GOLDSTEIN, S. 2002. Molecular Eletronics: Devices, Systems and Tools for Gigagate, Gigabit Chips. In *Proc. Int'l Conf. on Computer-Aided Design*. 443–440.

CHEN, Y., JUNG, G., OHLBERG, D., LI, X., STEWART, D., JEPPESEN, J., NIELSEN, K., STODDART, J., AND WILLIAMS, R. 2003. Nanoscale molecular-switch crossbar circuits. *Nanotechn*. *14*, 462–468.

COLLIER, C., WONG, E., BELOHRADSKY, M., RAYMO, F., STODDART, J., KUEKES, P., WILLIAMS, R., AND HEATH, J. 1999. Electronically configurable molecular-based logic gates. *Science 285*, 391–394.

CUI, Y. AND LIEBER, C. 2001. Functional nanoscale electronics devices assembled using silicon nanowire building blocks. *Science 291*, 851–853.

DEHON, A. 2003. Array-based architecture for FET-based, nanoscale electronics. *IEEE Trans. Nanotechn*. *2*, 23–32.

DEHON, A., LINCOLN, P., AND SAVAGE, J. 2003. Stochastic assembly of sublithographic nanoscale interfaces. *IEEE Trans. Nanotechn*. *2*, 165–174.

DEHON, A. AND WILSON, M. 2004. Nanowire-based sublithographic programmable logic arrays. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*. 123–132.

DOHERTY, F. C., LUNDGREN, J. R., AND SIEWERT, D. J. 1999. Bicliquecovers and partitions of bipartite graphs and digraphs and related matrix ranks of 0,1-matrices. *Congressus Numerantium 136*, 73–96.

GOLDSTEIN, S. AND BUDIU, M. 2001. NanoFabrics: Spatial computing using molecular electronics. In *Proceedings of the International Symposium on Computer Architecture*. 178–189.

HUANG, J., TAHOORI, M. B., AND LOMBARDI, F. 2004. On the defect tolerance of nano-scale two-dimensional crossbars. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*. 96–104.

HUANG, Y., DUAN, X., CUI, Y., LAUHON, L., KIM, K., AND LIEBER, C. 2001. Logic gates and computation from assembled nanowire building blocks. *Science 294*, 1313–1317.

MEYER, F. J. AND PRADHAN, D. K. 1989. Modeling defect spatial distribution. In *IEEE Trans. Comput*. *38*, 538–546.

NAEIMI, H. AND DEHON, A. 2004. A greedy algorithm for tolerating defective crosspoints in NanoPLA design. In *Proceedings of the International Conference on Field-Programmable Technology*. 49–56.

NANTERO. 2005. www.nantero.com.

RUECKES, T., KIM, K., JOSELEVICH, E., TSENG, G., CHEUNG, C., AND LIEBER, C. 2000. Carbon nanotube-based nonvolatile random access memory for molecular computing. *Science 289*, 94–97.

TAHOORI, M. 2005a. A mapping algorithm for defect tolerance of reconfigurable nano architectures. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 668–672.

TAHOORI, M. 2005b. Defects, yield, and design in sublithographic nano-electronics. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*. 3–11.

ZIEGLER, M. AND STAN, M. 2002. Design and analysis of crossbar circuits for molecular nanoelectronics. In *Proceedings of the IEEE International Conference on Nanotechnology*. 323–327.

Received June 2006; accepted June 2006 by Vijay Krishnan Narayanan