Low-Overhead Defect Tolerance in Crossbar Nanoarchitectures

MEHDI B. TAHOORI
Northeastern University

It is anticipated that the number of defects in nanoscale devices fabricated using bottom-up self-assembly process is significantly higher than that for CMOS devices fabricated by conventional top-down lithography patterning. This is mainly because of inherent lack of control in self-assembly fabrication as well as atomic scale of devices. The goal of defect tolerance, as an integral part of nano computing, is to obtain error-free computation from such fabrics containing defective elements.

In this article, an application-independent defect tolerant scheme for reconfigurable crossbar array nanoarchitectures is presented. The main feature of this approach is that the existence and location of defective resources within the nano-fabric are hidden from the entire design flow, resulting in minimum post-fabrication customization per chip and minimum changes to the entire design and synthesis flow. It is also shown how to drastically minimize the area overhead associated with this flow. The proposed technique requires extraction of regular yet incomplete defect-free subsets, in contrast to previously proposed complete defect-free subsets. This can greatly reduce the area overhead required for defect tolerance while not sacrificing logic mapping or signal routing capabilities. Extensive simulation results confirm considerable reduction in the area overhead without any negative impact on the usability of modified defect-free subsets.

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

ACM Reference Format:

Tahoori, M. B. 2009. Low-overhead defect tolerance in crossbar nanoarchitectures. ACM J. Emerg. Technol. Comput. Syst. 5, 2, Article 11 (July 2009), 24 pages. DOI = 10.1145/1543438.1543444 http://doi.acm.org/10.1145/1543438.1543444

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

Author's address: M. B. Tahoori, Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115; email: mtahoori@ece.meu.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2009 ACM 1550-4832/2009/07-ART11 $10.00

DOI 10.1145/1543438.1543444 http://doi.acm.org/10.1145/1543438.1543444

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 [Butts et al. 2002; Collier et al. 1999; Bachtold et al. 2001; Cui and Lieber 2001]. 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) [DeHon 2003; Goldstein and Budiu 2001].

Self-assembly processes promise to considerably lower manufacturing costs, at the expense of reduced control of the exact placement of these devices. Without fine-grained control, these devices will exhibit higher defect rates, both at the manufacturing and during lifetime operation of nano devices [Butts et al. 2002; Collier et al. 1999; DeHon 2003; Goldstein and Budiu 2001]. In addition to lacking control of precise placement of nanoscale devices, designers will also need to consider the effect of using a fabrication process that yields devices that are only a few atoms in diameter. For instance, the contact area between silicon nanowires is a few tens of atoms. With such small cross-sectional and contact areas, fragility of these devices will be orders of magnitude higher than devices currently being fabricated using conventional lithographic techniques. This will result in higher susceptibility to static and transient faults.

The programmability of crossbar nanoarchitectures can be well exploited 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 [Tahoori and Mitra 2004; Brown and Blanton 2004; Wang and Chakrabarty 2005; Tehranipoor 2005], they can be bypassed by post-fabrication configuration. However, conventional application-dependent defect tolerant schemes are not very effective for nano-computing since per-chip-customized configuration in the entire physical design flow causes serious problems for high-volume production. These include prohibitively large amount of information to be stored for the location of defect-free resources within the chip (defect map) and increased post-fabrication customized design efforts per chip.

In earlier publications, an alternative defect tolerant flow, the so-called *defect-unaware design flow*, is presented in which most physical design steps (as well as higher level design steps) are unaware of the existence and the location of defects in the chip [Tahoori 2006b, 2005a]. The main idea is to identify universal defect-free subsets of crossbars within the fabricated (partially defective) crossbars to be used in the design flow. The same structure for defect-free subsets is used for all crossbars in the chip and all chips fabricated in a similar fabrication process environment. This approach is able to considerably reduce the size of the defect map as well as the amount of per-chip customized design effort due to universality of defect-free subsets used in the design flow.

This article extends the application-independent defect tolerance at the crossbar level to crossbar array (i.e., architecture) level by providing a technique for reliably connecting defect-free subsets of individual crossbars. The technique presented in this article is essential to extend defect tolerance from the crossbar-level to the architecture-level. We also present techniques to hide a considerable amount of area overhead associated with the application-independent approach. This work complements our previous work [Tahoori 2006a] by presenting a thorough simulation-based experimental analysis of various area overhead reduction techniques.

The main focus of this work is on manufacturing defects that affect the manufacturing yield. Hence, run-time defects reducing the lifetime reliability of these devices, which are handled by fault tolerant techniques, are outside of the scope of this work.

The contributions of this article are summarized as follows:

—An application-independent defect tolerant crossbar array 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. It is experimentally shown that the probability of finding defect-free permutation crossbars with required connectability is always more than the probability of finding defect-free user crossbars given the same defect density.
—Techniques to reduce the area overhead of the application-independent defect tolerance are provided. These approaches are based on extracting regular yet incomplete "defect-free subset" of a partially defective crossbar in contrast to complete defect-free subset. The results show that this can greatly reduce the area overhead without affecting the usability of the crossbar (routability or the ability to map logic functions depending on the role of the crossbar). An efficient algorithm to extract these modified defect-free subsets is provided. Also, area overhead reduction versus routability of various defect-free subsets are experimentally analyzed.
—Finally, the situations in which permutation crossbars can be avoided to reduce the associated area and performance overhead are investigated.

The rest of the article is organized as follows. In Section 2, some backgrounds on crossbar nanoarchitectures, definitions, and defect-tolerant design flows are provided. The proposed array-based defect-tolerant architecture is presented in Section 3. Techniques for area overhead reduction of the user crossbars presented in Section 4. Hiding the area overhead of permutation crossbars is discussed in Section 5. Finally, Section 6 concludes the article.

2. BACKGROUND

2.1 Programmable Crossbar Array Architectures

Of the molecular-scale devices being developed using self-assembly techniques, the nonvolatile programmable switch has gained much attention. With these bottom-up techniques, it is possible to build features (e.g., wires and programmable switches) without relying on lithography. The bottom-up approaches used in the fabrication of nanoscale 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.

Recent work shows how to build nanoscale *programmable logic arrays* (PLAs) using the bottom-up synthesis techniques being developed by physical chemists [Goldstein and Budiu 2001; DeHon and Wilson 2004]. The molecular switches can provide connection and disconnection states at the crosspoints of vertical and horizontal wires in a crossbar, thereby providing a path to continue the advance of field-programmable technology beyond the end of the traditional, lithographic roadmap. 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]. At each intersection (*crosspoint*), there is a programmable nonvolatile switch, a two-terminal nanodevice formed by the layering [Huang et al. 2001]. In these crossbars, configuration of 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 wire crossing area, crosspoint switches in this nanotechnology take the same area as of a wire crossing. These crossbar arrays are similar to PLAs and can be used as building blocks for implementing logic. The array can then be interconnected, using CMOS circuitry, as part of a hybrid nano-scale/CMOS design architecture.

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 an array based nanoarchitecture using PLAs [DeHon and Wilson 2004; DeHon 2003; DeHon et al. 2003]. The main building block, called the nano programmable logic array (nanoPLA), is built from a crossed set of N-type and P-type nanowires. This architecture allows inversion by using nanowire Field Effect Transistor (FET) devices as buffers. Logic functionality is achieved in the form of two stages of programmable crossbars. The first stage defines the logical product terms (pterms) by creating a wired-OR of appropriate inputs. The outputs of this wire-OR plane are restored through field-effect controlled nanowires that invert the outputs (thus creating the logical NOR of the selected input signals). These restored signals are then sent to the inputs of the next stage of programmable crosspoints. Each nanowire in this plane computes the wired-OR of one or more restored pterms. The outputs of the stage are then restored in the same manner as the first stage. The two stages together provide NOR-NOR logic (equivalent to a conventional PLA) The nanoPLA is programmed using lithographic-scale wires along with stochastically-coded nanowire addressing [DeHon et al. 2003].

The *molecular CMOS* (CMOL) circuits proposed in Ma et al. [2005] 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. The nano crossbar is turned by some angle less than 90° relative to the CMOS pin array. By activating two pairs of perpendicular CMOS lines, two pins together with the two nanowires they contact are connected to the CMOS lines. Each nanodevice may be uniquely accessed using this approach. 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. The most straightforward application of CMOL would be for memories (embedded or standalone). The CMOL circuits have also been proposed for building FPGA-like architectures for implementing random logic [Strukov and Likharev 2005].

A CMOS-compatible crossbar memory array, called *NRAM*, has been proposed by Nantero Inc.[1] 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 be also used to implement programmable logic and interconnection network.

2.2 Definitions

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 $G = (U, V, E)$, as illustrated in Figure 1(b). A bipartite graph is a special graph where the set of vertices can be divided into two disjoint sets U and V such that no edge has both end-points in the same set. U represents the set of input nanowires and V represents the output nanowires; E, the set of edges, 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, $|E| = |U| \times |V|$. Therefore, a defect-free crossbar can be modeled by 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 that is complete. A subgraph of a bipartite graph is a *biclique* if this subgraph is a complete

[1]www.nantero.com.

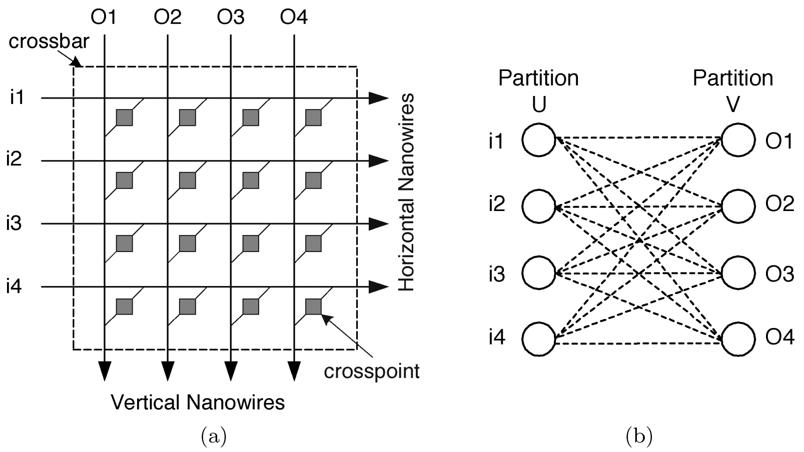


Fig. 1. (a) 4 × 4 2D nano-scale crossbar; (b) bipartite graph representation.

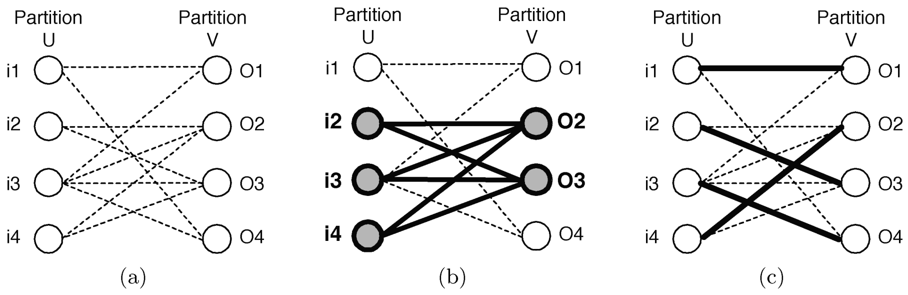


Fig. 2. (a) a bipartite graph $G(U, V, E)$; (b) biclique $(\{i_2, i_3, i_4\}, \{o_2, o_3\})$ in G; (c) a perfect matching in G.

bipartite graph. The maximum complete subset of a bipartite graph is called the maximum biclique.

In some applications, such as logic mapping and special cases of routing, it is required to find a one-to-one mapping between inputs outputs through crosspoints. 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.

Figure 2 shows an example of a 4 × 4 bipartite graph. It also shows the maximum biclique in this graph. This maximum biclique is 3 × 2. Every node is the set $\{i_2, i_3, i_4\}$ is connectable to each and every node in $\{o_2, o_3\}$. A perfect matching of this bipartite graph is also shown.

2.3 Defect Tolerance of Crossbar Architectures

Defect tolerance of nanoarchitectures has been discussed in the literature. Hierarchically 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 in a bipartite graph and a greedy matching algorithm with linear time complexity for logic mapping is presented. It is shown that when 20% of devices (i.e., crossbar diodes) were defective, only a 10% overhead in devices was needed to correctly configure the array around the defects. Manufacturing yield of crossbars for generating defect-free matching, for logic mapping, and defect-free complete subsets (bicliques), in a generalized form, has been discussed in Huang et al. [2004] and Tahoori [2005b], respectively.

Thorough testing and high-resolution diagnosis play major roles in the implementation of defect and fault tolerant systems. Test and diagnosis techniques for crossbar array architectures have been presented in [Mishra and Goldstein 2003; Tahoori and Mitra 2004; Brown and Blanton 2004; Wang and Chakrabarty 2005; Rad and Tehranipoor 2006], which exploit the reconfigurability of these nanoarchitectures to achieve a very high coverage test and high resolution diagnosis. These techniques are mainly categorized as *Built-in Self-test* (BIST) since test vector generation and output response analysis are performed on-chip.

In the conventional application-dependent defect tolerance, the existence and the location of defective elements are identified using test and diagnosis steps. After these steps, each individual resource in the crossbar array (e.g., nanowire, crosspoint, buffer) is precisely identified as defect-free (usable) or defective (unusable). The fault location information is stored in the so-called *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 a defect map. Particularly, placement and routing phases of the physical design use the defect map in order to map the design to the crossbar array by using only defect-free resources. In addition to physical design phases, higher-level design steps, such as logic and architecture design, might need to be modified to incorporate defect map information. Although this approach is able to recover majority of defect-free resources, it has major shortcomings, as follows. 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. Because of these limitations this approach is unsuitable for high-volume production.

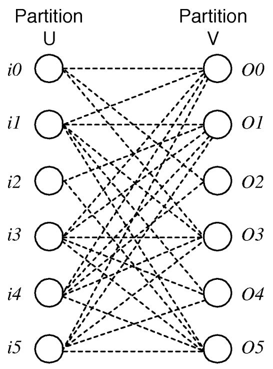
An *application-independent* defect tolerant design flow is presented in Tahoori [2005a, 2005b]. In this flow, defects are bypassed before mapping any particular application to the nanoarchitecture. In other words, defect tolerance is performed once and the same recovered array can be used for all applications mapped to the nanoarchitecture. Almost all design steps, from high-level architecture design to the last step of the physical design, are unaware of the existence and the location of defects within the nanoarchitecture. The key idea in the application-independent defect tolerance is to identify *universal* defect-free subsets of resources within the original partially-defective nanochip. All design steps use these universal defect-free subsets as the physical implementation devices (*design view*). Hence, conventional architectural, logic, and physical design flows and tools can be reused. These defect-free subsets are called “universal” because the size of these subsets is identical for all nanochips fabricated in the same process environment (similar defect densities). Also, these universal defect-free subsets remain unchanged for different applications mapped into the same nanoarchitecture, making this approach application-independent. An additional step, after placement and routing, is required to map the used resources within the universal subset (design view) into 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 nano-chip. The two major tasks of defect-tolerance in this flow is to

(1) identify and extract the defect-free universal subsets from partially-defective nano-chip, and
(2) perform the final mapping phase, which consists of mapping the used resources in the design view to the actual resources in the physical view.

In this flow, test and diagnosis steps identify the size and the location of defect-free $k \times k$ crossbars within partially defective fabricated $n \times n$ crossbars. The size of the maximum defect-free crossbar, k, can be estimated using a yield analysis method [Tahoori 2005b, 2006b] based on a sample of fabricated devices and this value will be used for all chips manufactured in the same process environment. This is done as a preproduction phase, similar to existing *silicon debug* steps.

Once the value of k is fixed, in the production flow, all crossbars with a defect-free subset not smaller than $k \times k$ are considered as “good” (pass) devices, otherwise they are marked as “bad” (fail) devices. The information regarding the location of defect-free subsets of crossbars is stored in a reduced defect map. During the physical design, the original design will be mapped (placed and routed) into an array 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 [Tahoori 2005a]. 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 [Huang et al. 2004; Tahoori 2005a].

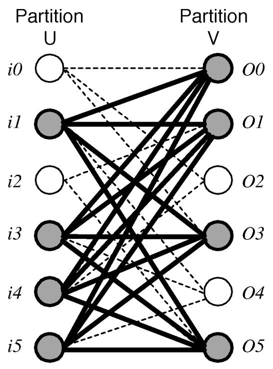
Figure 3 shows how the reduced defect map is generated (reduction from $O(n^2)$ to $O(n)$) and the final mapping is performed. In Figure 3(a), the bipartite graph model of a defective 6×6 crossbar is shown (physical view). This crossbar is defective since the corresponding bipartite graph is not complete. Each missing edge corresponds to a defective switch. The complete defect map, as used in the application-dependent flow, is shown in Figure 3(b) in which a 1(0) entry represents a usable (unusable) switch. The defect-free 4×4 subset of the crossbar is shown in Figure 3(c). The reduced defect map containing only of two vectors of size n is shown in Figure 3(d). 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



(a)

$$\left(\begin{array}{ccccccc} & i_0 & i_1 & i_2 & i_3 & i_4 & i_5 \\ O_0 & 1 & 1 & 0 & 1 & 1 & 1 \\ O_1 & 0 & 1 & 1 & 1 & 1 & 1 \\ O_2 & 1 & 0 & 0 & 0 & 1 & 0 \\ O_3 & 1 & 1 & 0 & 1 & 1 & 1 \\ O_4 & 0 & 1 & 0 & 1 & 0 & 1 \\ O_5 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

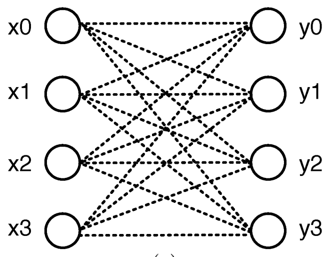
(b)



(c)

$$\begin{array}{l} \mathrm{V} = \begin{array}{cccccc} [1 & 1 & 0 & 1 & 0 & 1\,] \\ [y_0 & y_1 & & y_2 & & y_3] \end{array} \\ \\ \mathrm{H} = \begin{array}{cccccc} [0 & 1 & 0 & 1 & 1 & 1\,] \\ [& x_0 & & x_1 & x_2 & x_3] \end{array} \end{array}$$

(d)



(e)

Fig. 3. (a) physical view: 6×6 crossbar; (b) complete $O(n^2)$ defect map; (c) defect-free 4×4 subset of crossbar; (d) reduced defect map: two vectors of $O(n)$; (e) design view 4×4 crossbar.

partition (U) participating in the defect-free subset whereas the vertical vector corresponds to the output partition (V). The design view, containing only the defect-free 4×4 subset is shown in Figure 3(e). Each resource (programmable switch or nanowire) can be uniquely mapped into the physical view (Figure 3(a)) using the reduced defect map (Figure 3(d)). The inputs (outputs) of the crossbar in the design view correspond to "1" entries in the horizontal (vertical) vector in the reduced defect map, in the same order. For instance, the switch (x_0, y_2) in the defect-free subset corresponds to the switch (i_1, O_3) in the actual 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 necessarily be connected in the crossbar array. Consider the example of two 6×6 crossbars shown in Figure 4. 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

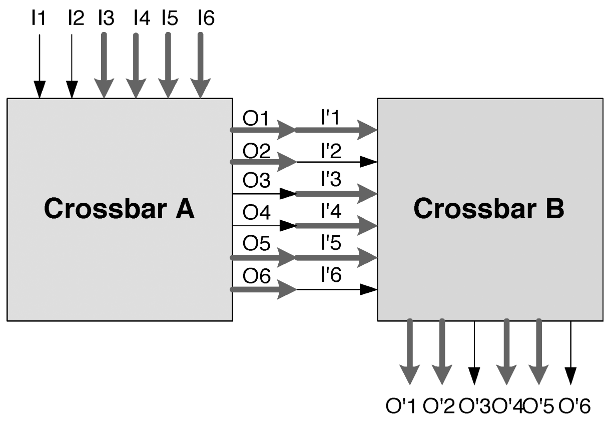


Fig. 4. Nonmatching defect-free subsets of two neighbor crossbars.

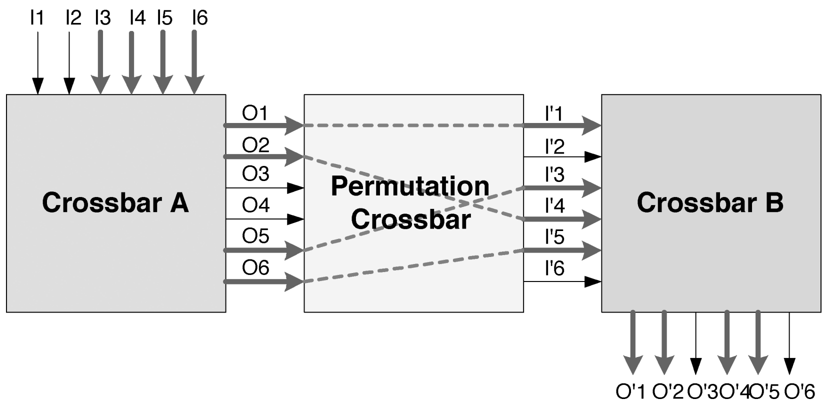


Fig. 5. Using permutation crossbar in connecting two neighbor crossbars.

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.

In general, the defect-free subsets obtained locally 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 Figure 4, a crossbar is used to match the input and outputs nanowires of the defect-free subsets of those two crossbars, as shown in Figure 5. 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 Figure 5, 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 Figure 5, 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 Tahoori [2005a, 2005b]. 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 [Brown and Blanton 2004; Wang and Chakrabarty 2005]. Algorithms for finding a defect-free matching within a crossbar are presented in [Naeimi and DeHon 2004; Huang et al. 2004]. 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 performed per crossbar.

Figure 6(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 Figure 6(b). The design view only consists of an array of $k \times k$ defect-free UCBs. This architecture is consistent with previously proposed crossbar array architectures [Goldstein and Budiu 2001; DeHon and Wilson 2004]. For instance, in the NanoFabric architecture, nanoBlocks are (logic) UCBs and switch blocks are composed of (interconnect) UCBs and PCBs.

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. Nevertheless, in Section 5 we discuss the situations in which some PCBs can be omitted to save some area overhead of PCBs in the fabricated crossbar array.

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 [Tahoori 2005b].

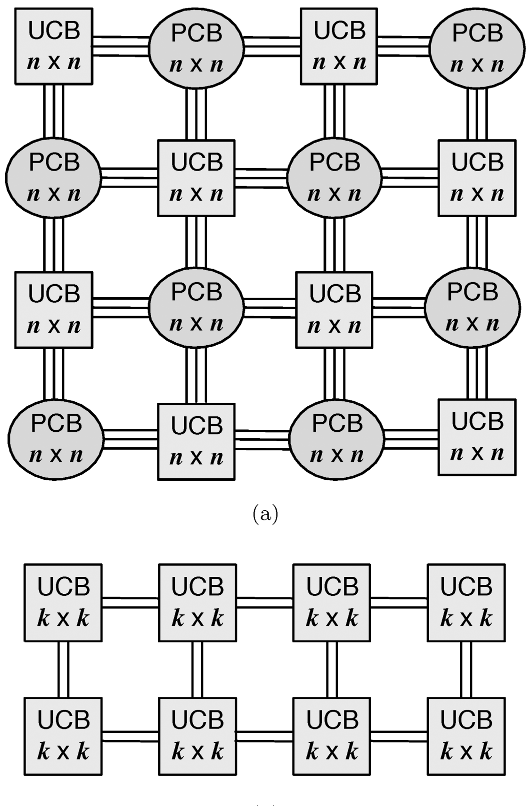


Fig. 6. (a) defect tolerant architecture using UCBs and PCBs; (b) design view.

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 7 compares the probability (yield) 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. In these experiments, $n^2.d$ defects are randomly injected into an $n \times n$ crossbar, where d is the defect density. These defects are injected in the bipartite graph model of the crossbar. Then, it is checked whether a $k \times k$ defect-free biclique exists in this crossbar. This experiment has been repeated for a large sample of crossbars (between 1000 and 5000), for each value of n and d, in order to obtain the probability of success (yield) as the ratio of crossbars with $k \times k$ defect-free biclique over the total number of experiments. The same procedure has been performed for defect-free matching as well. These experiments show that the probability of finding a defect-free matching in a PCB (for any arbitrary k inputs to any arbitrary k outputs in

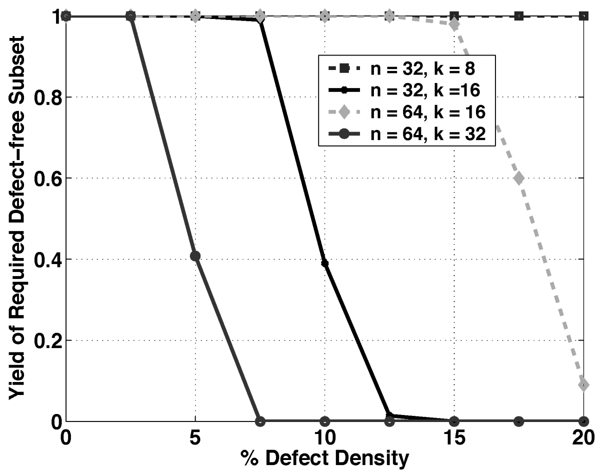


Fig. 7. UCB yield for various values of n and k.

an $n \times n$ PCB) is always 100% when up to 30% of switches are defective (open defects).

This result 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.

As presented in Section 2.2, the "simple" molecular crossbar considered in this article is composed of two sets of orthogonal nanowires, which means that the inputs are provided in one side and the outputs are generated in another side. The simple crossbars can be combined to form a "complex" crossbar able to accept inputs from more than one side and provide outputs in multiple sides. Such structures are similar to those used as switch blocks in the NanoFabric architecture [Goldstein and Budiu 2001]. The matching problem in complex crossbar structures can be modeled with the corresponding matching problems in the underlying simple crossbars. For instance, matching in a permutation crossbar which accepts k inputs from two sides and sends them to k outputs in two other sides can be modeled as two $k_1 \times k_1'$ and $k_2 \times k_2'$ matching problems in the underlying simple permutation crossbars where $k_1 + k_2 = k$ and $k_1' + k_2' = k$. Since $k_1, k_2, k_1', k_2' \leq k$, then *yield*($k_1 \times k_1'$ *matching*), *yield*($k_2 \times k_2'$ *matching*) $\geq$ *yield*($k \times k$ *matching*). Therefore, different connection patterns in complex crossbars can be handled by underlying PCBs, the same way that is achieved in simple crossbars.

In the next section we discuss how to reduce the area overhead of this application-independent defect tolerance flow in the UCBs. Section 5 describes the techniques to reduce the area overhead associated with PCBs in the crossbar array.

4. REDUCING AREA OVERHEAD IN UCBS

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. In other words, the application-dependent defect tolerance tries to maximize the number of defect-free resources recovered from each chip whereas the main goal of the application-independent defect tolerance is to minimize per-chip and per-application post-fabrication design efforts.

In application-dependent defect tolerance, the size of the crossbars used for application mapping (k) is the same as the size of the fabricated crossbars (n), that is, $k = n$. However, the physical design flow is both application-dependent and device-specific, meaning that the physical design needs to be tailored for each application and for each fabricated chip. In application-independent defect tolerance, the size of the crossbars used for application mapping is smaller than the size of the fabricated crossbars ($k < n$) [Tahoori 2006b]. However, the design flow is both application-dependent and device-unspecific, allowing the same physical design to be used for all fabricated chips irrespective of the defect map and distribution of individual chips. Since for high-volume production it is extremely important to minimize the amount of per-chip post-fabrication efforts, the application-independent defect tolerance is best suited in the production flow. Moreover, at the nanoscale integration level, there are plenty of resources on chip (10^{12} to 10^{14} devices per unit area). Therefore, the extra area overhead of the application-independent defect tolerance can be justified.

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, as will be discussed in the rest of this section.

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

4.1 Logic UCB Overhead Reduction

The structure of UCBs, as described in Section 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.

In logic mapping, only a random (arbitrary) defect-free matching is sufficient [Naeimi and DeHon 2004]. 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 8 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

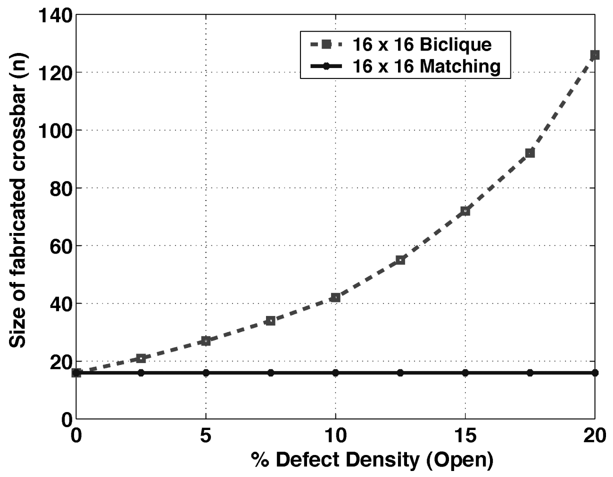


Fig. 8. Size of the fabricated crossbar to achieve defect-free 16×16 biclique and matching.

a defective crossbar are described in Naeimi and DeHon [2004] and Huang et al. [2004]. These results are obtained by randomly injecting $n^2.d$ defects (for a given defect density d) in the bipartite graph model of the $n \times n$ crossbar. For each defect-injected crossbar, it is determined whether the required defect-free $k \times k$ subset can be extracted. The yield, $Y^d_{n,k}$, is obtained as the ratio of successful cases over the total number of fault injected crossbars (a large sample between 1000 and 5000 crossbars for each data point). For the fixed values of k and d, and desirable yield of y (in these experiments, $y = 100\%$), we find an n such that $Y^d_{n,k} = y$ but $Y^d_{n-1,k} < y$. In other words, 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.

With 20% open defect density, it is still 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.

4.2 Interconnect UCB Overhead Reduction

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 lines in a 24×24 switch block.[2] 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 $m \ll k$. An example of a 4×4 2-regular bi-graph is shown in Figure 9.

[2]In *Xilinx Datasheets*, www.xilinx.com.

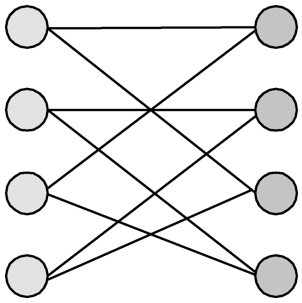


Fig. 9. 4×4 2-regular bi-graph.

```
Function HasRegular($G(U, V, E), m$)
  Obtain $\overline{G}(U, V, \overline{E}), \overline{E} = K_{|U|,|V|} - E$
    Sort $U$ based on $d(u)$ in $\overline{G}$ (decreasing order)
    Sort $V$ based on $d(v)$ in $\overline{G}$ (decreasing order)
    $toggle \leftarrow true$
    $reg_U \leftarrow false, reg_V \leftarrow false$
    Repeat
      if $toggle$ then
        $u \leftarrow$ node in $U$ with maximum degree
        if $d(u) > |V| - m$ then
          $U \leftarrow U - \{u\}$
          for each $v' \in V$ such that $(u, v') \in \overline{E}$ do
              $d(v') \leftarrow d(v') - 1$
          Re-sort $V$ accordingly
        else
          $reg_U \leftarrow true$
      else
        $v \leftarrow$ node in $V$ with maximum degree
        if $d(v) > |U| - m$ then
          $V \leftarrow V - \{v\}$
          for each $u' \in U$ such that $(u', v) \in \overline{E}$ do
              $d(u') \leftarrow d(u') - 1$
          Re-sort $U$ accordingly
        else
          $reg_V \leftarrow true$
      $toggle \leftarrow \neg toggle$
    Until $U = \phi \vee V = \phi \vee (reg_U \wedge reg_V)$
  return $|U| \times |V|$ as the largest $m$-regular subgraph
```

Fig. 10. Algorithm for extracting m-regular subset.

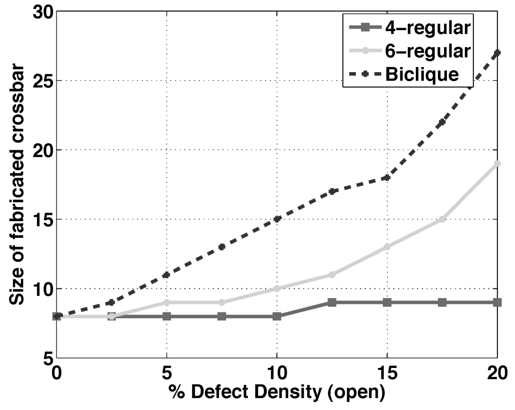
Similar to an FPGA-like switch block structure, instead of extracting defect-free $k \times k$ bicliques for interconnect UCBs, it is possible to use defect-free m-regular bi-graphs where $m \ll n$. Since the number of edges (defect-free switches) in an m-regular bi-graph (mk) is much smaller than that 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.

4.2.1 *Algorithm.* A heuristic greedy algorithm for finding the maximum m-regular subset of a bipartite graph is presented in Figure 10. 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.

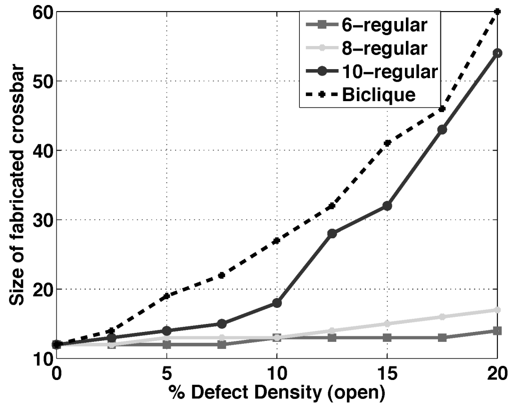
The approach is based on converting this problem to the dual problem in the *complement* graph. The complement of a graph G is a graph $\overline{G}$ with the same set of vertices such that two vertices of $\overline{G}$ are adjacent if and only if they are not adjacent in G. Since the goal is to extract 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 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.

Run-Time Analysis. 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 $O(n \log n)$.

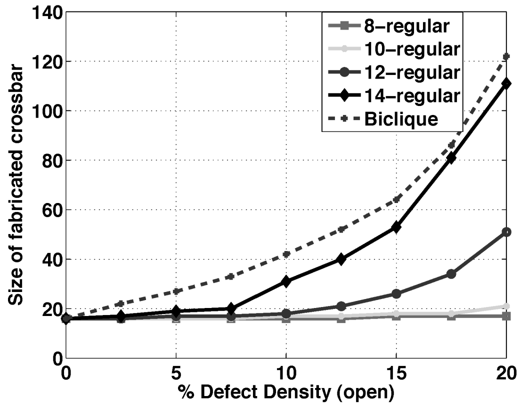
4.2.2 *Experimental Analysis.* A fault injection framework, as explained in Section, 4.1, is used to implement the above algorithm and compare the area overhead of various m-regular structures compared to complete subsets (bicliques). Figure 11 compares the area overhead required to achieve a defect-free $k \times k$ biclique and m-regular $k \times k$ structures (each of k input nanowires is connectable to m output nanowires) for various values of k and m in the presence of open defects. For instance, with 20% defect density of open defects, only an 18×18 crossbar needs to be fabricated to achieve a defect-free 16×16 8-regular structure. The required size for a defect-free 16×16 biclique is 126×126. This shows a considerable area saving of m-regular subsets over bicliques. For $m < k/2$, the overhead is almost zero for all these crossbars; that is, it is always possible to extract a defect-free m-regular $k \times k$ from a $k \times k$ defective crossbar if $m < k/2$ and defect densities less than 25%. Results for larger values



(a) $k = 8,\ m = 4, 6$



(b) $k = 12,\ m = 6, 8, 10$



(c) $k = 16,\ m = 8, 10, 12, 14$

Fig. 11. Size of the fabricated crossbar to achieve defect-free m-regular $k \times k$ structures as well as $k \times k$ bicliques in the presence of open defects.

of m are also shown. As can be seen in this figure, the area overhead becomes comparable to bicliques as m approaches to k. However, it needs to be mentioned that $m > k/2$ are never used in contemporary industrial switch blocks.

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 $k \times k$ subset. In other words, all k^2 switches in the $k \times k$ subset should be free of short defects. This means that the problem changes from finding a defect-free m-regular $k \times k$ subset into finding a defect-free $k \times k$ biclique (only for short defects). The algorithm for finding a defect-free biclique has been discussed in Tahoori [2005a, 2005b].

Figure 12 compares the area overhead of biclique and m-regular structure for both open and short defects considering an equal percentage of open and short defects. As can be seen in this figure, in each case the sizes of the minimum fabricated crossbar for various values of m are the same. This is because the size of the minimum fabricated crossbar is dictated by short defects (which require defect-free biclique) rather than open defects (which require defect-free m-regular structure). In other words, as long as $m \leq k$, the size of fabricated crossbar to achieve $k \times k$ bicliques is always greater than or equal to the size of fabricated crossbar to obtain m-regular $k \times k$ subsets.

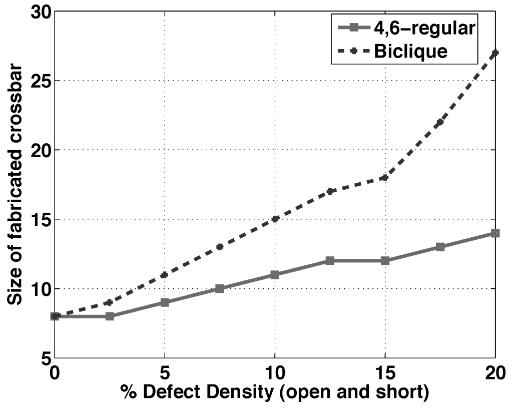
Table I shows area overhead reduction of m-regular UCBs compared to biclique UCBs for some representative defect densities (opens and shorts) and values of k. As can be seen in this table, the overhead reduction drastically improves for larger values of k as well as higher defect densities. Therefore, using m-regular defect-free subsets for interconnect UCBs provides a sufficient degree of routability with much reduced area overhead.

5. PCB OVERHEAD REDUCTION

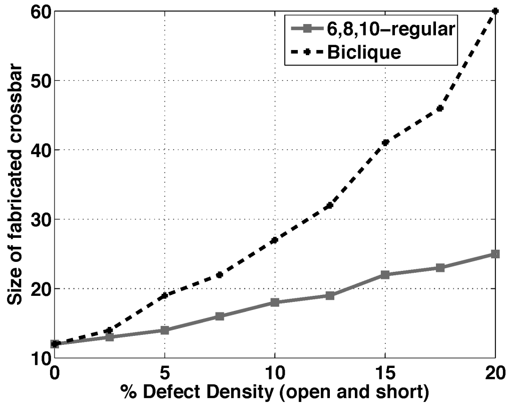
5.1 UCB Direct Match

In the architecture presented in Figure 6(a), there is a PCB between every 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.

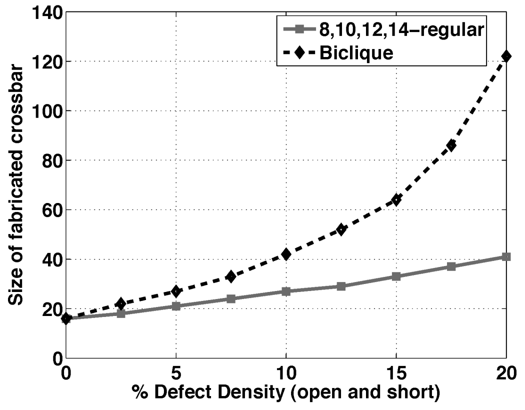
Since the value of k is fixed for all fabricated crossbars within the fabric, the actual maximum defect-free subsets of all crossbars in the fabric can be extracted and compared for a possible match of k signals between neighboring crossbars. Consider that the maximum defect-free subset (biclique, m-regular, 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 13 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



(a) $k = 8,\ m = 4, 6$



(b) $k = 12,\ m = 6, 8, 10$



(c) $k = 16,\ m = 8, 10, 12, 14$

Fig. 12. Size of the fabricated crossbar to achieve defect-free m-regular $k \times k$ structures as well as $k \times k$ bicliques in the presence of both open and short defects.

Table I.
UCB area overhead reduction of m-regular UCBs compared to biclique UCBs for different sizes of UCBs and defect densities (DD)

	5% DD	10% DD	15% DD	20% DD
8×8	1.5x	1.9x	2.3x	3.7x
12×12	1.9x	2.3x	3.5x	5.8x
16×16	1.7x	2.4x	3.8x	8.9x

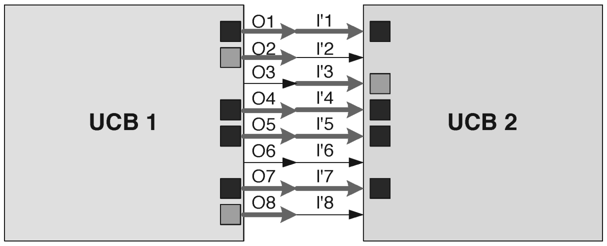


Fig. 13. Connecting UCBs without a PCB.

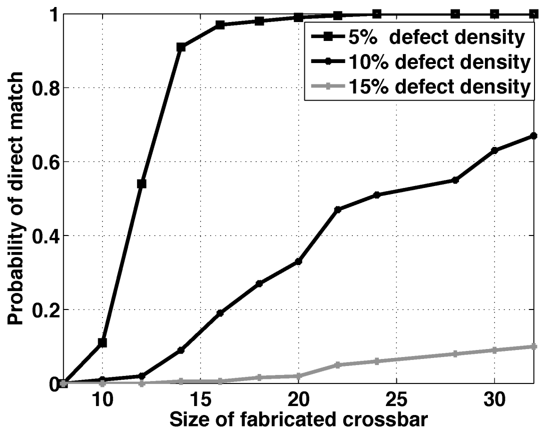
between defect-free inputs and outputs (shown in bold) is 4. Therefore, no PCB is required to connect these two UCBs. As the size of the actual defect-free subset of the crossbar becomes larger than the required defect-free size, the possibility of such direct match increases. This process, although performed in a per-chip basis, is still application-independent, i.e. before mapping applications to the chip.

5.2 Experimental Analysis

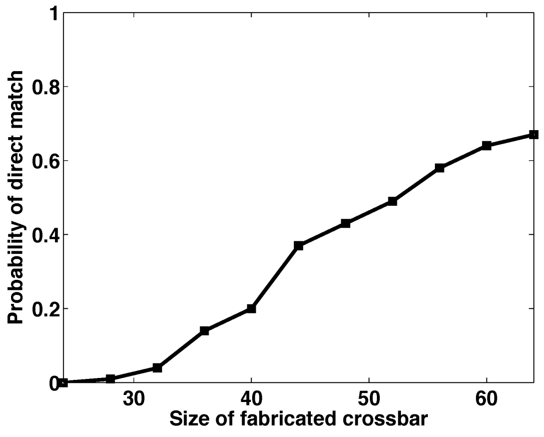
We have performed experiments to evaluate PCB overhead reduction due to direct match between neighbor UCBs. We have used the same experiment framework, as used in the previous section, to inject random defects in the crossbar array, and examine the probability that two adjacent defect-free subsets are already matched, as discussed in Section 5.1.

First, consider the situation in which the defect-free subsets are complete (i.e., bicliques). The results for the probability of direct match for neighboring 8×8 and 16×16 defect-free bicliques are shown in Figure 14. As the size of the fabricated crossbar increases, the probability of direct match increases as well. Moreover, this probability drastically decreases for larger defect densities. For example for 16×16 defect-free bicliques, such probability is very close to zero for 10% defect denisty and higher.

However, the probability of finding direct match in adjacent UCBs is much higher for defect-free m-regular UCBs compared to biclique UCBs. This probability is almost 100% for the cases reported in Figure 14. These results confirm that using m-regular UCBs instead of bicliques (complete defect-free subsets) not only reduces UCB area overhead considerably, but minimizes PCB overhead also.



(a) $k = 8$, Defect density = 5%, 10%, 15%



(b) $k = 16$, Defect density = 5%

Fig. 14. The probability of finding direct match (no PCB) for biclique UCBs.

5.3 Discussion

The overhead reduction techniques presented in Section 4 for reducing UCB area overhead are both application-independent and *device-unspecific*. They are application-independent since they can be used for all designs to be mapped to the crossbar array and they are not tailored for a particular design. These techniques are device-unspecific as the universal defect-free subset is the same for different fabricated chips and the universal defect-free subsets, as the outcome of these techniques, are independent of defect distribution and defect map in each particular chip.

However, the technique presented in this section for PCB overhead reduction is still application-independent although it is *device-specific*. This means that this overhead reduction can be used before mapping any particular design to the crossbar array and hence, can be used for all possible designs. Therefore, this technique is still consistent with the application-independent defect tolerance. However, it is device-specific meaning that the outcome could be different for each fabricated chip depending on defect distribution. Since this technique is application-independent, it has some advantages over the previous application-dependent methods because it still decouples defect handling from the design flow and does not impose difficulties and overhead to the original design flow.

It needs to be mentioned that the PCB overhead reduction technique can be used in addition to the device-unspecific methods of Section 4 in the situations where it is required to further reduce the area overhead.

6. 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 article, an application-independent defect tolerant architecture is presented in which universal (application-independent) defect-free subsets 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 (user crossbars) by using some permutation crossbars in order to achieve a defect-free crossbar array architecture.

Also, techniques to reduce the area overhead of the proposed defect-tolerant flow have been presented. Instead of extracting "complete" defect-free subsets, incomplete but regular defect-free subsets are required. This results in a considerable area overhead reduction without any negative impact on the usability (logic mapping or signal routing) of crossbars. Moreover, the use of regular defect-free subsets instead of complete defect-free subsets increases the chance of direct matching between user crossbars, hence, eliminating the majority of the overhead associated with permutation crossbars. Therefore, it can be concluded that regular yet incomplete defect-free subsets are suitable choices for application-independent defect tolerance with minimal area overhead both at the crossbar level and crossbar array level.

REFERENCES

Bachtold, A., Harley, P., Nakanishi, T., and Dekker, C. 2001. Logic circuits with carbon nanotube transistors. *Science*. *294*. 1317–1320.

Brown, J. G. and Blanton, R. D. S. 2004. CAEN-BIST: Testing the nanofabrics. In *Proceedings of the International Test Conference.* 462–471.

Butts, M., DeHon, A., and Goldstein, S. 2002. Molecular eletronics: Devices, systems and tools for gigagate, gigabit chips. In *Proceedings of the International Conference 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. *Nanotech. 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. Nanotech. 2*, 23–32.

DeHon, A., Lincoln, P., and Savage, J. 2003. Stochastic assembly of sublithographic nanoscale interfaces. *IEEE Trans. Nanotech. 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.

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.

Ma, X., Strukov, D., Lee, J., and Likharev, K. 2005. Afterlife for silicon: CMOL circuit architectures. In *Proceedings of the IEEE Conference on Nanotechnology*. 175–178.

Mishra, M. and Goldstein, S. C. 2003. Defect tolerance at the end of the roadmap. In *Proceedings of the International Test Conference*. 1201–1210.

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.

Rad, R. and Tehranipoor, M. 2006. SCT: An approach for testing and configuring nanoscale devices. In *Proceedings of the VLSI Test Symposium*. 370–377.

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.

Strukov, D. and Likharev, K. 2005. CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotech. 16*, 888–900.

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.

Tahoori, M. 2006a. Application-independent defect-tolerant crossbar nano-architectures. In *Proceedings of the IEEE International Conference on Computer-Aided Design*.

Tahoori, M. 2006b. Application-independent defect tolerance of reconfigurable nanoarchitectures. *J. Emerg. Technol. Comput. Syst. 2*, 3, 197–218.

Tahoori, M. and Mitra, S. 2004. Defect and fault tolerance of reconfigurable molecular computing. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*. 176–185.

Tehranipoor, M. 2005. Defect tolerance for molecular electronics-based nanofabrics using built-in self-test procedure. In *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*. 305–313.

Wang, Z. and Chakrabarty, K. 2005. Using built-in self-test and adaptive recovery for defect tolerance in molecular electronics-based nanofabrics. In *Proceedings of the International Test Conference*. 477–486.

Received July 2008; accepted October 2008 by Vijaykrishnan Narayanan