Defect Tolerance in Diode, FET, and Four-Terminal Switch Based Nano-Crossbar Arrays

Onur Tunalı
Nanoscience and Nanoengineering Department
Istanbul Technical University
onur.tunali@itu.edu.tr

Mustafa Altun
Electronics and Communication Engineering Department
Istanbul Technical University
altunmus@itu.edu.tr

Abstract—**In this paper, defect tolerance performance of switching nano-crossbar arrays is extensively studied. Three types of nanoarrays where each crosspoint behaves as a diode, FET, and four-terminal switch, are considered. For each crosspoint, both stuck-open and stuck-closed defect probabilities are independently taken into consideration. A fast heuristic algorithm using indexing and mapping techniques is proposed. The algorithm measures defect tolerance performances of the crossbar arrays that are expected to implement a certain given function. The algorithm's effectiveness is demonstrated on standard benchmark circuits that shows 99% accuracy compared with an exhaustive search. The benchmark results also show that not only the used technology, the nanoarray type, but more significantly the specifics of given functions affect defect tolerance performances.**

Keywords—*defect tolerance, nano-crossbar, reconfigurable arrays*

I. Introduction

Developments in nano-crossbar fabrication techniques have made it possible to use each crosspoint as a conventional electronic component such as a diode, transistor, or switch [1] [2]. This is a unique opportunity that allows to integrate well developed conventional circuit design techniques into nano-crossbar arrays. As expected, the integration comes with some challenges and defect tolerance is one of the significant ones. Defect rates are much higher for nano-crossbars compared to conventional CMOS circuits [3]. Therefore developing new defect tolerance techniques for nano-crossbars is a must, especially for high defect rates up to 20%. In this study, we assess and compare defect tolerance performances of different nano-crossbar architectures/technologies that is conducted through finding a valid mapping in accordance with the proposed algorithm and defect maps in case of randomly distributed defects.

Crossbar nanotechnologies are favorably achieved by nanotubes or nanowires [4] [5] such that each crosspoint behaves as a diode, FET, and four-terminal switch [10] [12] [13]. This is illustrated in Figure 1. Regarding crosspoints as switching elements, both static and reconfigurable crossbars are presented. A predetermined design with static crossbars is not capable of defect tolerance because it is not possible to create alternative routes for defective crosspoints. On the contrary, reconfigurable designs can be manipulated to tolerate defects. This study focuses on reconfigurable crossbars by considering randomly occurred stuck-open and stuck-closed crosspoint defects. We consider diode, FET, and four-terminal switch based crossbar arrays. While diode and FET based crossbars use conventional resistive diode and CMOS logic [1] [2] [12] [13], four-terminal switch based arrays use a novel logic synthesis methodology that is extensively studied in [10]. Figure 2 shows implementations of a specific Boolean function with these three synthesis methods.

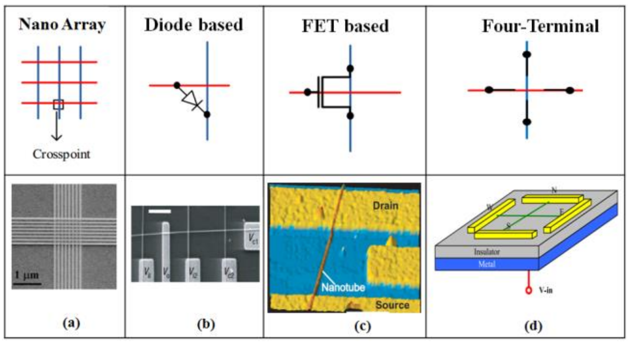


Fig. 1 **(a)** Nano-crossbar [12] based on **(b)** diode [12], **(c)** FET [13], **(d)** four-terminal switch [10] crosspoints.

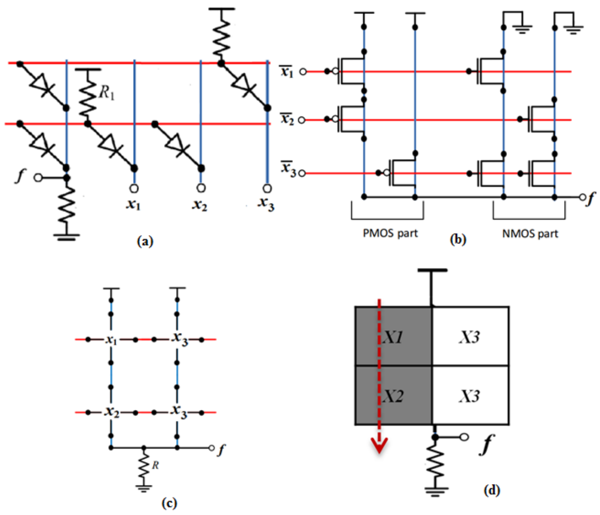


Fig. 2. Implementation of $f = x_1 x_2 + x_3$ with **(a)** diode, **(b)** CMOS, and **(c)** four-terminal switch based logic; **(d)** assigning logic 1's to x_1 and x_2 makes a top-to-bottom connection that results in f=1.

Mapping a target logic function on a defective crossbar is an NP-complete problem [6]. In the worst-case scenario, an *N x M* crossbar has *N!M!* permutations that is intractable for a reasonable computing time. Different algorithms and heuristics are presented to tackle this issue. Graph based models are proposed in [7] and [8] that use a fan-out embedding heuristic and a maximum flow algorithm, respectively. In addition to graph based approaches, "Integer Linear Programming" is used in [9] and [14] that employs a pruning centered approach with certain constraints. It is shown in [14] that defect tolerance results might dramatically vary with the chosen algorithm correlated to the algorithm's accuracy. We test and compare our algorithm with an exhaustive search to establish an accuracy of 99%. However, it should be noted that large crossbars are computationally intractable to be included in exhaustive search so we only consider crossbars up to 7x7 size for comparison. In addition, mentioned approaches so far are using pre-determined crossbar sizes to find a mapping for a chosen logic function. We use an optimal crossbar to realize a logic function which means that both the function and the crossbar matrices have the same size. We test logic functions in Irredundant Sum-of-Products (ISOP) form that is consistent with using optimal crossbar sizes. Also we include 3 different logic families for comparison which departs from the mentioned studies in the literature.

We propose a heuristic algorithm that creates and compares index representations of a given function and a defective certain sized crossbar to be used to implement the function. We show that if the index representations are not matched then the defective crossbar cannot be used to implement the function; otherwise, it can be used. We prove that this is a necessary and sufficient condition. Our algorithm eliminates considerable amount of crossbar mapping permutations that is the main headache for the mentioned studies in the literature. Furthermore, our viewpoint is comprehensive that is concerned both with the types of the crossbar technologies and the characteristics of given functions. The proposed indexing based algorithm is direct used for diode and CMOS based logic of nano-crossbars. For four-terminal switch based logic, the proposed algorithm is partially used; a conventional matrix based matching is mainly performed. Also we include 3 different logic families for comparison which departs from stated studies. It is important to determine features of different families due to post-production selection according to inherent defect types.

Organization of the paper is as follows. In section II, we give detailed explanation of key concepts used in the algorithm. In section III, we explain the proposed algorithm used for diode and CMOS based logic of nano-crossbars. In section IV, we briefly explain the defect tolerance technique used for four-terminal switch based logic. In section V, we present proof of correctness for theorem used in the algorithm. In section VI, we present experimental results and elaborate on them. In section VII, we discuss our contributions and future works.

II. PRELIMINARIES

We aim to find out whether it is possible or not to map a logic function on a given defective crossbar. For this purpose we deal with matrix representations of a given logic function and a defective crossbar as shown in Figure 3.

Function Matrix: Literals and products are appointed to the columns and rows, respectively. If a literal occurs in a product, it is denoted with 1; otherwise 0 is assigned.

Crossbar Matrix: Functional switches of crossbars are denoted with x; defective stuck-open and stuck-closed switches are denoted with 0 and 1, respectively. It should be noted that functional switches can be 0 or 1, so they can be matched with either element of a function matrix.

In order to find a valid mapping, defective switches of a crossbar matrix which are denoted as 0 (stuck-open) or 1 (stuck-closed) must be matched with 0's (unused) and 1's (used) in the function matrix. Here, an important property is that row and column permutations do not alter the implemented function, so there are exponentially many different implementations in terms of the crossbar size. This is an important advantage and

$f = x_1x_2x_3 + x_2x_5 + x_3x_4 + x_1x_5$

$$\begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline p1 & 1 & 1 & 1 & 0 & 0 \\ p2 & 0 & 1 & 0 & 0 & 1 \\ p3 & 0 & 0 & 1 & 1 & 0 \\ p4 & 1 & 0 & 0 & 0 & 1 \end{array}$$

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline \alpha & x & x & x & 0 \\ \beta & x & 0 & x & 0 \\ \gamma & x & x & 0 & 0 \end{array}$$

x: Functional Switches
0:Defective Switches
(Stuck-open)

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline \alpha & x & 1 & x & 1 \\ \beta & x & 1 & x & x \\ \gamma & 1 & x & x & x \end{array}$$

x: Functional Switches
1:Defective Switches
(Stuck-closed)

Fig. 3 Function matrix and crossbar matrices for different defect types.

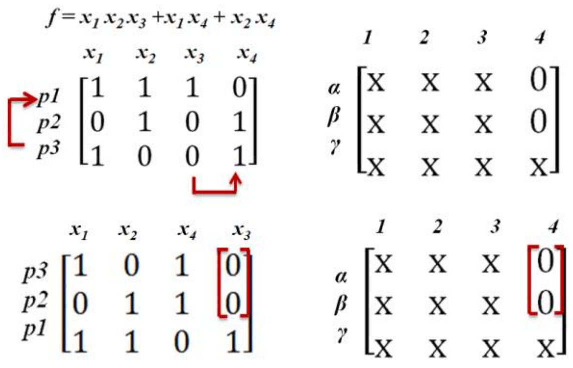


Fig. 4 Row and column permutation of the matrix to obtain valid mapping.

feature for defect tolerance as illustrated in Figure 4. Our algorithm exploits this feature to develop its indexing based methodology. We use index concept in a distinct way for our algorithm. Index means number of same matrix elements for a chosen value in a row or column.

Row Index: Number of same elements in a row for a chosen value. For example, p_4 in figure 3 has a row index of 3 for a chosen value of 0.

Column Index: Number of same elements in a column for a chosen value. For example x_4 in figure 3 has a column index of 1 for a chosen value of 1.

Double Index: It is found for "*one*" element of a matrix rather than for a row or a column. Double index is a 2-tuple denoted with (a, b). First number of tuple is a row index corresponding to the selected element' occurrence in the row and second number of tuple is a column index with the same principle. For example, double index of a_{44} (0) element of function matrix in Figure 3 is (3, 3). It lies in p_4 row and x_4 column; p_4 has a row index of 3 and x_4 has a column index of 3 for 0 value.

In the proposed algorithm, we use set of mentioned indices to eliminate impossible mappings (row and column indices) and find valid mapping without considering all permutations (double index). Definition of sets as follows:

Set of Row Indices: A set of all row indices of a matrix for a chosen value. In Figure 3, rows of the function matrix p_1, p_2, p_3 *and* p_4 have row indices 2, 3, 2 and 3, respectively, for a chosen value of 0. So its set of row indices is denoted as $I_{R,F} = \{2, 3, 2, 3\}$ where R stands for row and F stands for function.

Set of Column Indices: A set of all column indices of a matrix for a chosen value. In Figure 3, columns of the function matrix x_1, x_2, x_3, x_4 and x_5 have column indices 2, 2, 2, 3 and 2, respectively for a chosen value of 0. So its set of column indices is denoted as $I_{C,F} = \{2, 2, 2, 3, 2\}$ where C stands for column and F stands for function.

Set of Double Indices: A set of all double indices of matrix elements having a same value. In Figure 3, set of double indices for 0 is $D_F = \{(2,3), (2,2), (3,2), (3,2), (3,2), (3,2), (3,2), (3,2), (3,2), (3,2), (3,3)\}$ where F stands for function and in case of crossbar C is used.

It should be noted that, set of double indices stays the same after row and column permutation of a matrix. By checking equality of two sets, it is possible to conclude if there is a mapping between function and crossbar matrices. We will prove correctness of these noted conclusion in section V.

III. PROPOSED ALOGRITHM FOR DIODE AND CMOS BASED LOGIC

The outline of our four-step algorithm is shown below.

Input: Function matrix and crossbar (defective) matrix

Output: If there is a matching "YES"; otherwise "NO"

Step 1: If the number of defective switches is greater than the corresponding elements in the function matrix, then return "NO".

Step 2: Sort matrices according to the row and column index, if the crossbar matrix has at least one row or column index greater than a row or column index of the function matrix, return "NO".

Step 3: If the number of defects is equal to or smaller than the worst-case limit W_C, return "YES".

Step 4: Find the reduced matrix and find set of double indices. Start subarray search. If a subarray is found with the equal set of double indices as the reduced matrix with 20,000 trials, return "YES"; otherwise return "NO".

We will then explain each step in details. The algorithm will be demonstrated with an example in Figure 5. It should be noted that the example and the following explanations are for stuck-open defects, nevertheless they can be applied easily for stuck-closed defects by considering defects as 1s (as opposed to 0s) to be matched with 1s (as opposed to 0s) in the function matrix.

The explanations of the algorithm steps for stuck-open defects:

1. Step: *If the number of defective switches is greater than the corresponding elements in the function matrix, then return "NO".*

We consider stuck-open defects so algorithm checks 0s in the crossbar matrix to match 0s in the function matrix. If 0s in thecrossbar matrix is greater than 0s in the function matrix, then it is not possible to find mapping.

2. Step: *Sort matrices according to the row and column index, if crossbar matrix has at least one row or column index greater than a row or column index of the function matrix, return "NO".*

We sort matrices according to the row and column indices. For example, in Figure 5 index sets of the sorted function and crossbar matrices for 0 are:

$I_{R,F} = \{2, 2, 2, 1\}$ $I_{C,F} = \{2, 2, 2, 1\}$
$I_{R,C} = \{2, 1, 0, 0\}$ $I_{C,C} = \{2, 1, 0, 0\}$.

There is a perfect matching between sets. However, if a member of a crossbar set would be greater than corresponding member in a function set, there would not be matching. This would mean there are excessive defective element for matching. For the example in Figure 5, this is not an issue so we proceed to the 3.step.

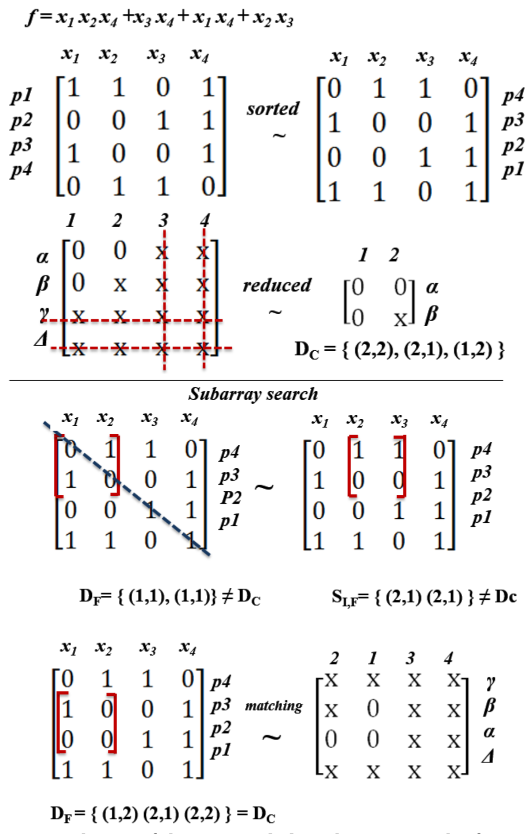
3. Step: *If the number of defects is equal to or smaller than the worst-case limit W_C, return "YES".*

Worst-case limit of a function matrix is the maximum number of tolerable defects in any defect distribution related to the row and column index. We find W_C with using sets of row and column indices. First we choose minimum members in sets of row and column indices, separately. After that we choose the minimum between two members obtained in the first step. W_C gives us minimum row and column index. If crossbar matrix has defective elements less than or equal to W_C, then defects can be matched with any row and column in a function matrix. Let's show finding W_C of the function matrix in Figure 5. Index sets of the function are as follows:

$I_{R,F} = \{2, 2, 2, 1\}$ $I_{C,F} = \{2, 2, 2, 1\}$
$\min\{I_{R,F}\} = 1$ and $\min\{I_{C,F}\} = 1$ so $W_C = 1$

In Figure 5, crossbar matrix has 3 defective elements, so we cannot conclude if there is a matching without checking the fourth step.

4. Step: *Find the reduced matrix and find set of double indices. Start subarray search. If a subarray is found with the equal set of double indices as the reduced matrix with 20,000 trials, return "YES"; otherwise return "NO".*



$f = x_1x_2x_4 + x_3x_4 + x_1x_4 + x_2x_3$
sorted
reduced
$D_C = \{(2,2), (2,1), (1,2)\}$
Subarray search
$D_F = \{(1,1), (1,1)\} \neq D_C$
$S_{I,F} = \{(2,1)(2,1)\} \neq D_c$
matching
$D_F = \{(1,2)(2,1)(2,2)\} = D_C$

Fig. 5 Fourth step of the proposed algorithm: sorting the function matrix, crossbar reducing, and subarray search.

In a crossbar matrix, Xs corresponding to functional switches do not change the double index of a matrix element. For this reason, we erase columns and rows consisting of only such elements (Xs) for compactness. The acquired matrix keeps the same set of double indices. Figure 6 shows an example for this.

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline \alpha & 0 & x & 0 & x \\ \beta & x & x & x & x \\ \gamma & 0 & x & x & 0 \end{array} \qquad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline \alpha & 0 & x & 0 & x \\ \beta & x & x & x & x \\ \gamma & 0 & x & x & 0 \end{array} \qquad \begin{array}{c|ccc} & 1 & 3 & 4 \\ \hline \alpha & 0 & 0 & x \\ \gamma & 0 & x & 0 \end{array}$$

Fig. 6 Reduced matrix for subarray search

In the next step, we use a subarray search to find a matching between a reduced matrix and a subarray. Then, the function and crossbar matrices are sorted according to the sets of indices. We use this to increase the chance of finding matrices with the same set of double indices. Since matching elements are collected to the one side, search progresses diagonally. It can be seen from Figure 5 that seeking a subarray checks only set of double indices of a chosen subarray due to the Double Index Theorem, presented in the next section. As long as they have the same set of double indices, permutation of matrices is not necessary. Once two matrices are found with the same set, it means there is a mapping, so the algorithm returns "YES".

Subarray search has a trial limit of 20,000. We choose this value because when compared with exhaustive search, it gives 99% accuracy. When we run the algorithm by removing this limitation, we see that there is almost no change in the values. This approves the proposed heuristic algorithm's success. If a mapping could not be found in this range, it is interpreted as a negative result meaning that there is no mapping. In subarray search, double index theorem is only valid for matrices with the same number of elements to be matched. In case of unequal number of elements, new defective elements should be introduced to the reduced matrix to equalize the number of elements. If there are N missing elements for matching, 2^N possibilities occurs for consideration. Instead of this, functional switches denoted with x are shown with 0 and defective switches with 1. Same is applied to the subarray in the function matrix. Next, element by element multiplication of matrices is executed. Since functional switches can be matched with 0s and 1s in the function matrix, the resulting matrix can be compared with the reduced matrix. If they are equal, there is a matching.

In CMOS based design two matrices are used to model a logic function. All principles used for diode based design are valid with the exception of input permutations. In diode based design both inputs (columns) and products (rows) can be permutated with respect to the crossbar which are checked for a mapping. In CMOS based design, the first matrix is for the function itself and the second matrix for its complement. Important distinction is that inputs are in the same order for both matrices. For this reason while a mapping is searched for a crossbar, rows of matrices can be permutated independently; however inputs must be in the same order for both matrices

IV. DEFECT TOLERANCE FOR FOUR-TERMINAL SWITCH BASED LOGIC

For four-terminal switch based design, we partially use the proposed algorithm. We use matrix based defect maps similar to those used in the proposed algorithm. Our defect tolerance technique mainly depends on permutation trials and its technical

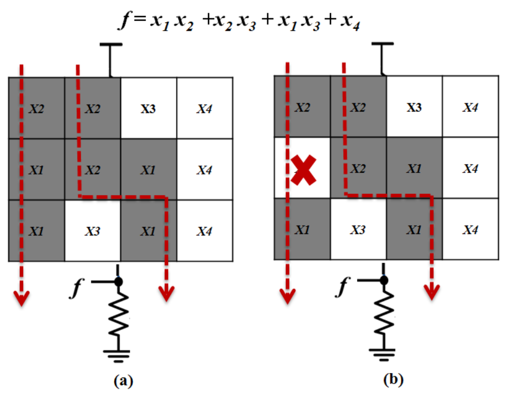


Fig. 7 **(a)** Four-terminal implementation of *f* with two connections between top and bottom plates; **(b)** In case of a stuck-open defect, first connection is broken down, however since there is a second connection *f* evaluates correctly.

contribution is limited for the technical content of this paper. Therefore, we briefly mention about it.

Defect Map: Four-terminal switch based design [10] uses every switch on the crossbar due to its layout method. Therefore there is no unused switch like those in diode and CMOS based designs. However, certain functions yield redundant paths or extra connections between top and bottom plates. Figure 7 shows occurrence of extra connections. If a defect existing in a crossbar appears only on one of the connections, then it can be compensated with the other connection and the correct result can still be achieved. A defect map of a function implemented with four-terminal switch based design displays these type of connections.

V. PROOF OF CORRECTNESS OF THE PROPOSED ALGORITHM

For the proposed algorithm in Section II, checking only set of row and column indices is not sufficient in case of equal index sets. When the sets of row and column indices of two different matrices are equal to each other, it is not possible to decide whether there is a mapping or not definitely. An example in Figure 8 is shown in case of equal sets without a matching. For this reason we introduce set of double indices to obtain necessary and sufficient cased for valid mapping. Before presenting the Double Index Theorem, let's explain relating concepts.

Matching one-to-one: Obtaining an identical matrix which is derived from another matrix with row and column permutations. In matching defined before, it is sufficient to match only defective elements. One-to-one matching provides an identical matrix. Importance of the distinciton lies in the connection of set of double indices to the mapping which will be explained in the proof of the theorem concerning the double indices.

Function Matrix **Array Matrix**

$$\begin{array}{c|cccc} & x_1 & x_2 & x_3 & x_4 \\ \hline p1 & 1 & 0 & 1 & 0 \\ p2 & 0 & 1 & 1 & 0 \\ P3 & 1 & 0 & 0 & 1 \end{array} \qquad \begin{array}{cccc|c} 1 & 2 & 3 & 4 & \\ \hline x & 0 & x & 0 & \alpha \\ x & 0 & x & 0 & \beta \\ 0 & x & 0 & x & \gamma \end{array}$$

$I_{R,F} = \{2, 2, 2\}$ $I_{C,F} = \{1, 2, 1, 2\}$

$I_{R,C} = \{2, 2, 2\}$ $I_{C,C} = \{1, 2, 1, 2\}$

Fig. 8. Both matrices have the same set of row and column indices for 0. However, there is no matching between them.

Double Index Theorem: *There is a one-to-one matching between two matrices if and only if their set of double indices are equivavelent.*

Lemma 1: *Row and column permutations do not alter the double index of a matrix element.*

Numbers of the double index are defined with the row and column indices in which the element is found. Therefore even though permutations changes the position of a row or a column, element is still in the same row and column with the same row and column indices which defines the double index.

Lemma 2: *Set of double indices is unique for a given matrix.*

The proof is by contradiction. Let's assume such a matrix that has 2 different set of double indices. Therefore sets should have different double indices for one element or more. Double indices in a matrix are determined according to a row and column index. Since matrix is not altered and has same number of elements in rows and columns for a chosen value of 0 or 1, it is not possible to have different row and column indices that comprises double indices. This contradicts with the assumption of having different double index for an element or more.

Lemma 3: *Set of double indices for a matrix does not change with row and column permutations.*

Rows and columns of a matrix is consisted of matrix elements which have the same double index after permutations according to Lemma 1. Therefore set stays the same since its members are not changed.

Proof of the Double Index Theorem:

Sufficiency: If there is a one-to-one matching between two matrices, then they are identical by definition. Therefore their set of double indices is equivavelent according to Lemma 2.

Necessity: Lemma 2 states that set of indices is unique for a certain matrix and Lemma 3 states that set of double indices stays the same after row and column permutation. Therefore, if two matrices have the same set of double indices, then they are either identical or permutation of one another; in both case they can be matched one-to-one.

Let's explain set of double indices of two matrices with an example in which both a function matrix and an crossbar matrix have the same set of double indices.

Example 1) Figure 9 shows a function matrix and an crossbar matrix which have the set of indices as follows:

D_F = {(2,2), (2,1), (2,1), (2,2), (2,2), (2,2)}

D_C = {(2,2), (2,1), (2,1), (2,2), (2,2), (2,2)}

Since their sets of double indices are the same there must be a permutation to acquire one-to-one matcing. Necessary manipulation is shown in Figure 9.

VI. Experimental results

We use standard benchmark circuits [11] to measure defect tolerance performances of different nano-crossbar technologies. We consider stuck-open and stuck-closed defect probabilities/rates of 10% and 20% for each crosspoint independently. Simulations are conducted in Matlab. Crossbars with random defects are produced with Matlab's predetermined function generator. To obtain defect tolerance values, a sample size of around 600 is used. At this level defect tolerance fluctuation stabilizes. All experiments run on a 1.70-GHz Intel Core i5 CPU (only single core used) with 4.00 GB memory. It takes 0.2 seconds for each sample in average to check a valid mapping that satisfies an accuracy of 99% compared with an exhaustive search.

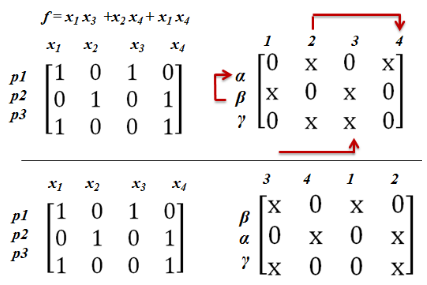


Fig. 9. Row and column permutations to obtain valid mapping

Table 1 shows the results of benchmark functions with respect to defect rates and defect types as well as the crossbar technologies. Considering the technologies and the related logic synthesis methodologies, diode based logic always has a better defect tolerance performance compared with the CMOS based one. Reason behind this is directly connected to the number of matchings necessitated for valid mapping. Since CMOS based logic uses two different planes for function realization, it needs to satisfy two matchings instead of one. Another important conclusion is that four-terminal switch based design yields better results for stuck-closed defects than stuck-open ones since the design generally requires to assign same literals to multiple switches on the same conduction path. Characteristics of the functions also play an important role in defect tolerance. Since stuck-open defects are tolerated with zeros in the functions' matrices, functions with relatively higher number of products compared to their number of literals have a better chance for tolerating stuck-open defects. On the contrary, functions with relatively higher number of literals compared to their number of products have a better chance for tolerating stuck-closed defects. For example Ex33 in Table 1, has a 14% defect tolerance for stuck-closed and 99% for stuck-open type defects. Significant difference between values occurs because of the mentioned literal related properties. Using Ex33 is more favorable for a crossbar with higher probability of stuck-open type defects.

VII. Conclusion

In this study, we propose a novel heuristic algorithm to evaluate defect tolerance performances of diode, CMOS, and four-terminal switch based logic families for nano-crossbar arrays. The algorithm uses indexing and mapping techniques that eliminates considerable amount of crossbar mapping permutations which is the main headache for related studies in the literature. The algorithm's effectiveness is demonstrated on standard benchmark circuits. The results show that not only the used technology, the nanoarray type, but more significantly the specifics of given functions affect defect tolerance performances. As a future work, we will extend our proposed methodology to be applicable to different nano-crossbar technologies including magnetic and organic switch based nanoarrays. We also have a plan to consider transient defects and optimum usage of redundancies to tolerate them.

ACKNOWLEDGEMENT

This work is supported by TUBITAK (The Scientific and Technological Research Council of Turkey) Career #113E760 and 2210-C programs.

Circuit Name	**Diode**				**CMOS**				**Four-Terminal**			
	stuck-closed		**stuck-open**		**stuck-closed**		**stuck-open**		**stuck-closed**		**stuck-open**	
	10%	***20%***	***10%***	***20%***	***10%***	***20%***	***10%***	***20%***	***10%***	***20%***	***10%***	***20%***
Alu 0	85%	54%	99%	96%	63%	50%	93%	72%	86%	64%	52%	26%
Alu 1	85%	54%	99%	96%	83%	53%	97%	73%	86%	64%	52%	26%
Alu 2	85%	54%	99%	96%	83%	53%	97%	73%	86%	64%	52%	26%
Alu 3	85%	54%	99%	96%	83%	53%	94%	73%	86%	64%	52%	26%
B12 0	98%	80%	98%	74%	46%	5%	95%	59%	28%	7%	28%	7%
B12 1	92%	33%	99%	75%	58%	1%	99%	91%	36%	9%	58%	19%
B12 3	96%	77%	96%	78%	90%	58%	93%	68%	42%	17%	43%	16%
B12 4	84%	40%	99%	96%	79%	25%	93%	74%	42%	17%	43%	16%
B12 6	68%	2%	95%	47%	14%	1%	99%	85%	27%	7%	21%	11%
B12 7	44%	2%	99%	95%	34%	1%	82%	75%	42%	11%	22%	3%
B12 8	32%	1%	99%	97%	24%	1%	99%	99%	82%	40%	22%	4%
C17 0	95%	78%	99%	94%	92%	70%	98%	87%	53%	26%	53%	26%
C17 1	96%	77%	96%	78%	91%	64%	92%	69%	43%	16%	43%	16%
Clpl 0	97%	69%	99%	98%	78%	23%	99%	92%	62%	29%	53%	20%
Clpl 1	98%	84%	99%	95%	98%	83%	98%	82%	39%	14%	48%	20%
Clpl 2	97%	82%	99%	94%	93%	79%	98%	92%	67%	40%	65%	42%
Clpl 3	87%	53%	99%	81%	49%	1%	50%	21%	18%	3%	41%	10%
Clpl 4	91%	41%	99%	97%	74%	6%	63%	50%	18%	3%	41%	12%
Dc1 1	99%	97%	95%	75%	84%	52%	93%	73%	52%	25%	52%	25%
Dc1 2	93%	55%	99%	96%	68%	9%	99%	96%	28%	6%	28%	6%
Dc1 5	99%	95%	97%	85%	96%	84%	84%	53%	65%	38%	53%	26%
Dc1 6	95%	79%	99%	88%	94%	70%	98%	86%	53%	25%	53%	25%
Ex5 31	56%	5%	99%	95%	30%	1%	83%	64%	35%	7%	22%	2%
Ex5 33	14%	1%	99%	98%	9%	1%	60%	43%	66%	28%	25%	4%
Ex5 46	45%	5%	99%	99%	38%	1%	84%	65%	17%	1%	28%	6%
Ex5 49	3%	0%	99%	99%	1%	1%	84%	65%	90%	32%	28%	6%
Ex5 50	23%	1%	99%	99%	22%	1%	87%	45%	93%	75%	22%	4%
Ex5 61	29%	2%	99%	99%	25%	1%	98%	78%	90%	32%	43%	16%
Ex5 62	28%	1%	98%	85%	23%	1%	96%	74%	95%	74%	37%	12%
Misex1 1	99%	96%	92%	66%	65%	17%	52%	9%	44%	16%	43%	16%
Misex1 2	78%	18%	99%	92%	30%	1%	98%	87%	29%	6%	36%	10%
Misex1 3	94%	38%	99%	86%	10%	1%	96%	67%	8%	1%	38%	11%
Misex1 4	93%	44%	99%	94%	8%	1%	99%	89%	27%	5%	38%	10%
Misex1 5	86%	45%	97%	80%	63%	3%	95%	64%	26%	4%	42%	14%
Misex1 6	89%	28%	99%	86%	26%	1%	93%	73%	12%	2%	29%	5%
Misex1 7	95%	57%	99%	92%	50%	1%	99%	93%	21%	3%	49%	15%
Newtag	59%	4%	99%	98%	52%	1%	96%	52%	62%	22%	30%	7%

Table 1 Defect tolerance performances of 3 different nano-crossbar based logic families.

References

[1] G. Snider, P. Kuekes, T. Hogg and R. Williams, 'Nanoelectronic architectures', *Appl. Phys. A*, vol. 80, no. 6, pp. 1183-1195, 2005

[2] A. Dehon, 'Nanowire-based programmable architectures', *ACM Journal on Emerging Technologies in Computing Systems*, vol. 1, no. 2, pp. 109-162, 2005.

[3] C. Collier, 'Electronically Configurable Molecular-Based Logic Gates', *Science*, vol. 285, no. 5426, pp. 391-394, 1999..

[4] M. Ziegler and M. Stan, 'CMOS/nano Co-design for crossbar-based molecular electronic systems', *IEEE Trans. Nanotechnology*, vol. 2, no. 4, pp. 217-230, 2003.

[5] S. Goldstein and M. Budiu, 'NanoFabrics', *ACM SIGARCH Computer Architecture News*, vol. 29, no. 2, pp. 178-191, 2001.

[6] Shrestha, A.M.S.; Tayu, S.; Ueno, S., "Orthogonal ray graphs and nano-PLA design," *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on* , vol., no., pp.2930,2933, 24-27 May 2009

[7] Wenjing Rao; Orailoq, A.; Karri, R., "Topology aware mapping of logic functions onto nanowire-based crossbar architectures," *Design Automation Conference, 2006 43rd ACM/IEEE* , vol., no., pp.723,726, 0-0 0

[8] Jing Huang; Tahoori, M.B.; Lombardi, Fabrizio, "On the defect tolerance of nano-scale two-dimensional crossbars," *Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings. 19th IEEE International Symposium on* , vol., no., pp.96,104, 10-13 Oct. 2004.

[9] Joon-Sung Yang; Datta, R., "Efficient Function Mapping in Nanoscale Crossbar Architecture," *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on* , vol., no., pp.190,196, 3-5 Oct. 2011..

[10] Altun, M.; Riedel, M.D., "Logic Synthesis for Switching Lattices," *Computers, IEEE Transactions on* , vol.61, no.11, pp.1588,1600, Nov. 2012.

[11] K. McElvain, "IWLS93 benchmark set: Version 4.0, distributed as part of the IWLS93 benchmark distribution, http://www.cbl.ncsu.edu:16080/benchmarks/lgsynth93/," 1993.

[12] W. Lu and C. Lieber, 'Nanoelectronics from the bottom up', *Nat Mater*, vol. 6, no. 11, pp. 841-850, 2007..

[13] P. Avouris, 'Molecular Electronics with Carbon Nanotubes', *Acc. Chem. Res.*, vol. 35, no. 12, pp. 1026-1034, 2002.

[14] M. Zamani and H. Mirzaei and M.B. Tahoori. "*ILP Formulations for Variation/Defect Tolerant Logic Mapping on Crossbar Nano-Architectures*". In ACM Journal on Emerging Technologies in Computing Systems, 2013.