IEEE TRANSACTIONS ON NANOTECHNOLOGY

Circuit Design Steps for Nano-Crossbar Arrays: Area-Delay-Power Optimization with Fault Tolerance

Journal:	*Transactions on Nanotechnology*
Manuscript ID	TNANO-00011-2019
Manuscript Type:	NANOARCH2018ATHENS
Date Submitted by the Author:	02-Jan-2019
Complete List of Authors:	Morgul, Muhammed Ceylan; Istanbul Teknik Universitesi Elektrik-Elektronik Fakultesi, Electronic and Communication Engineering Frontini, Luca; Universita degli Studi di Milano Tunali, Onur; Istanbul Teknik Universitesi Elektrik-Elektronik Fakultesi, Electronic and Communication Engineering Vatajelu, Ioana; Institut Polytechnique de Grenoble, EE and CS Ciriani, Valentina; Universita degli Studi di Milano anghel, lorena; INPG, EE and CS Moritz, Csaba Andras; U Mass, Amherst, ECE Stan, Mircea; University of Virginia, ECE Alexandrescu, Dan; IROC Technologies Altun, Mustafa
Keywords:	Crossbar Arrays, Logic Synthesis, Defect Tolerance, Fault tolerance, Performance Optimization, Memristor Arrays

ISTANBUL TECNICHAL UNIVERSITY

January 02, 2019

Dear Professor Fabrizio Lombardi,

Attached please find our submission to IEEE Transaction on Nanotechnology for special issue on "14th ACM/IEEE International Symposium on Nanoscale Architectures", titled "Circuit Design Steps for Nano-Crossbar Arrays: Area-Delay-Power Optimization with Fault Tolerance".

A preliminary version of this paper, titled "Integrated Synthesis Methodology for Crossbar Arrays" is presented at the 14th ACM/IEEE International Symposium on Nanoscale Architectures. Nearly 50% of material in this manuscript is new:

- At least 70% of the material in the logic synthesis are new;
    - we have proposed completely new technique for two-level logic synthesis for memristive nanoarrays
    - we have investigated multi-level logic synthesis for four-terminal switch based nanoarrays as well as other types (diode, memristor and FET), and we have compared them, first in this study
- At least 50% of the material in the subsection "Defect Tolerance for Four-terminal" is new, we have improved our technique and described in more detail;
- At least 95% of the material in the performance optimization is new, we have made in-depth analysis oo complexity of delay and power performances of nanoarray types;
- We have added new section for a case study to increase comprehensibility of complete design methodology;

The subject area of this paper is "Developing a complete integrated synthesis methodology for circuits and architectures".

Best Regards,

Mustafa Altun, PhD

Assistant Professor
Director of the Emerging Circuits and Computation Group
Istanbul Technical University

Email: altunmus@itu.edu.tr
Web: www.ecc.itu.edu.tr

ITU Ayazaga Campus 34469 Maslak, Sariyer, Istanbul, Turkey Tel: +90 212 285 3510 Fax: +90 212 285 3565

Circuit Design Steps for Nano-Crossbar Arrays: Area-Delay-Power Optimization with Fault Tolerance

M. Ceylan Morgul, Luca Frontini, Onur Tunali, E. Ioana Vatajel, Valentina Ciriani, Lorena Anghel, Csaba Andras Moritz, Mircea R. Stan, Dan Alexandrescu, and Mustafa Altun

***Abstract*—Nano-crossbar arrays have emerged to achieve high performance computing beyond the limits of current CMOS. They offer area and power efficiency in courtesy of their easy-to-fabricate and dense physical structures consisting of regularly placed crosspoints as computing elements. Depending on the used technology, a crosspoint behaves as a diode, a memristor, a field effect transistor, or a four-terminal switching device. In this study, we comparatively elaborate on these technologies in terms of their capabilities for computing in terms of area, delay, and power consumption. Also, we consider fault tolerance capabilities of the arrays. Due to the stochastic nature of nano-fabrication, nanoarrays have much higher fault rates compared conventional technologies such as CMOS. As a result, this study introduces a synthesis methodology that considers basic technology preference for switching crosspoints and defect or fault rates of the given nanoarray as well as their effects on performance metrics including power, delay, and area.**

***Index Terms*—Crossbar Arrays, Logic Synthesis, Defect Tolerance, Fault Tolerance, Performance Optimization, Memristor Arrays**

I. Introduction

Nano-crossbars have emerged to be an alternative technology to CMOS [30]. They are fabricated with relatively cheap bottom-up nano-fabrication techniques rather than using pure lithography based conventional production. Due to the novel manufacturing techniques, produced fabrics are regular and in dense forms that results in area and power efficient structures [9] [2].

M. Ceylan Morgul and Mustafa Altun are with the Department of Electronics and Communication Engineering, and Onur Tunali is with the Department of Nanoscience and Nanoengineering of Istanbul Technical University, Istanbul, Turkey e-mail: {morgul, onur.tunali, altunmus}@itu.edu.tr

Luca Frontini and Valentina Ciriani are with the Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy e-mail: {luca.frontini, valentina.ciriani}@unimi.it

E. Ioana Vatajelu and Lorena Anghel are with TIMA laboratory, Grenoble-Alpes University, Grenoble, France e-mail: {ioana.vatajelu, lorena.anghel}@imag.fr

Csaba Andras Moritz is with the Department of Electrical and Computer Engineering,University of Massachusetts, Amherst, Massachusetts, USA e-mail: andras@ecs.umass.edu

Mircea R. Stan are with the Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, Virginia, USA e-mail: mircea@virginia.edu

Dan Alexandrescu is with IROC Technologies, Grenoble, France e-mail: dan.alexandrescu@iroctech.com

This work is part of a project that has received funding from the European Union's H2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement #691178, as well as supported by the TUBITAK-Career project #113E760.

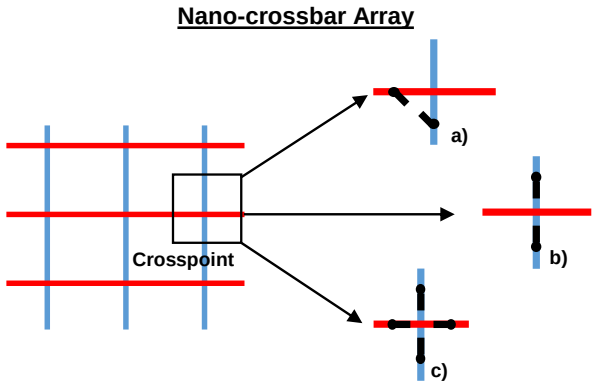

Fig. 1. Switching models of a nano-crossbar array: crosspoint as a) two-terminal switch with terminals in the crossed lines, b) two-terminal switch with terminals in the same line, and c) four-terminal switch.

Currently, computing is achieved with crosspoints behaving like switches, either as two-terminal or four-terminal. This is illustrated in Figure 1. Depending on the used technology, a two-terminal switch behaves either as a diode [13], a resistive/memristive switch [21], or a field effect transistor (FET) [22]. Diode and resistive switches correspond to the crosspoint structure in Figure 1(a); here, the switch is controlled by the voltage difference between the terminals. Figure 1(b) shows a FET based switch; here, the red line represents the controlling input. This is a unique opportunity that allows us to integrate well developed conventional circuit design techniques into nano-crossbar arrays. Finally, a novel four-terminal switch is demonstrated in Figure 1(c) and technology development is presented and analyzed in a recent paper [19] with detailed TCAD simulations. In mentioned paper, switching lattice can be fabricated using CMOS-like technology. It has four terminal either connected each other or not. Preferred state is actualized with a controlling input, which is not shown in the Figure 1(c) and has a separate physical formation from the crossbar, that is thoroughly explained for different technologies in [3] [19].

To illustrate different computing approaches, we show examples for the implementation of $f_{XOR_2} = x_1\overline{x_2} + \overline{x_1}x_2$ in Figure 2. Logic synthesis models for diode and memristor based crossbars are quiet similar to Programmable Logic Array (PLA) as can be seen in Figure 2(a) and 2(b). Memristor based crossbars have one major difference that establishing the output goes through several states/loops (for further information,

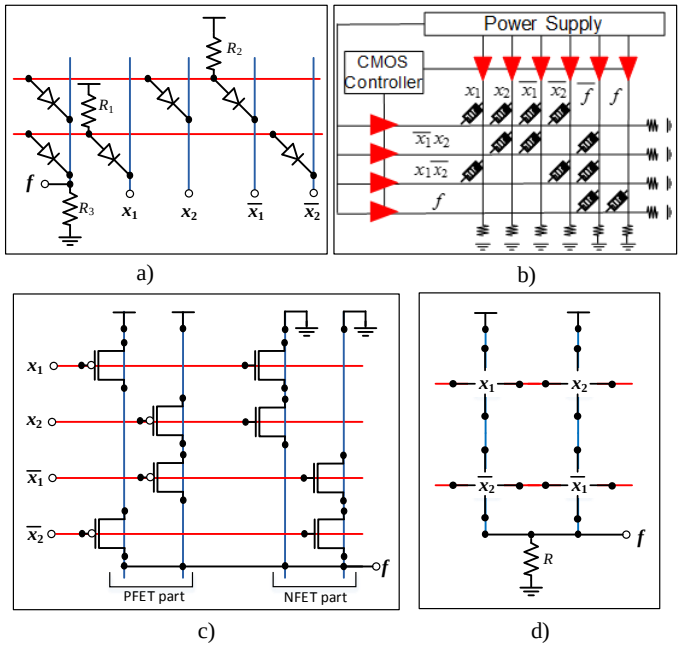

Fig. 2. Implementation of f_{XOR_2} with different nano-crossbar types: crosspoint as a) diode, b) memristor, c) FET, and d) four-terminal switch.

check [29]).

For FET based crossbars, each logic function product and dual function product is realized by a separate column, as seen in Figure 2(c). Each input is assigned to a row for controlling all the FETs on corresponding row.

Finally, a four-terminal based crossbar; here every crosspoint performs switching on all four directions. Control lines of crosspoints are not shown in Figure 2(d), yet detailed explanation of control lines can be found in [3] [19].

Regarding emerging technologies and nano-fabrication, fault rates are much higher for nano-crossbars, as expected, compared to those of conventional CMOS circuits [10]. Therefore, during logic synthesis, consideration of faults and defects is mandatory. This applies for the integration of both diode, FET based or novel four-terminal based logic synthesis methodologies. For this reason, researchers focus on challenges including defect and variance tolerances [25] [17]. Defect and variance tolerant approaches are closely related to logic realization and performance optimization, respectively.

Taking mentioned issues into account, we have developed a complete integration methodology for logic synthesis, defect tolerance and performance optimization. This methodology is designed as a step-by-step guide to combine modular research approaches into an entire production pipeline.

This work combines detailed survey for all steps of the methodology with new approaches and techniques for the missing parts, so a complete synthesis methodology is achieved. Surveyed and newly studied materials are listed below.

Surveyed items:

- Single and multi-output logic synthesis for diode, memristor, and FET based crossbars, and single-output logic synthesis of four-terminal lattice.

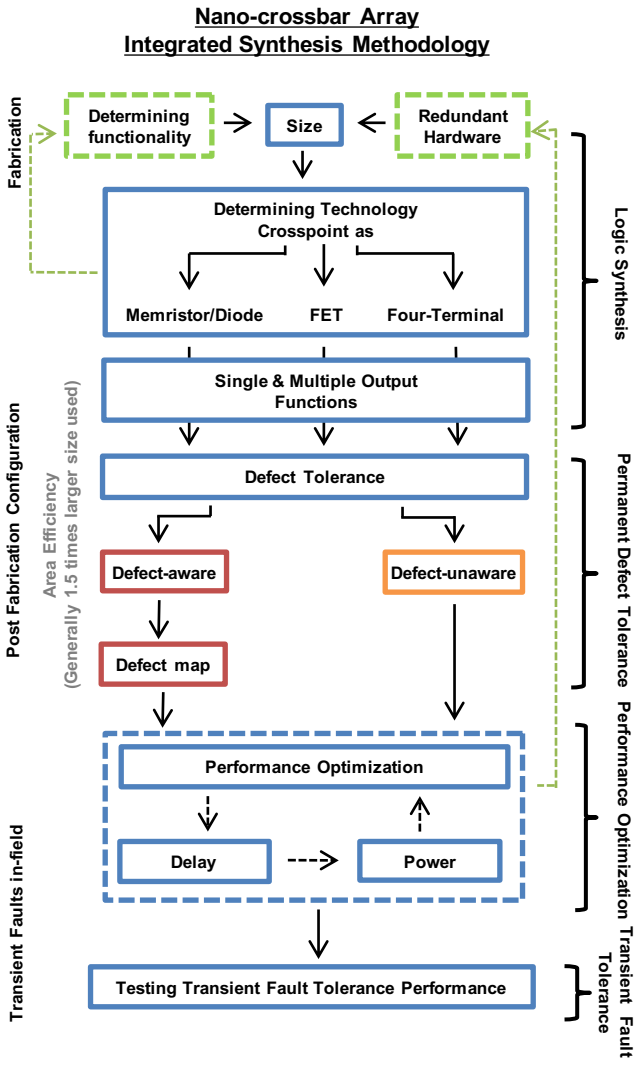

Fig. 3. Integrated synthesis methodology scheme for nano-crossbar arrays.

- Defect tolerance technique for diode, memristor, and FET based crossbars.
- Performance (delay-power) analysis of memristor arrays.

Newly studied items:

- Multi-output logic synthesis for four-terminal lattice, and comparison with others.
- A greedy optimization algorithm for two-level single-output memristor crossbar (logic synthesis).
- Defect tolerance technique for four-terminal lattice.
- Performance (delay-power) analysis of diode, FET and four-terminal arrays

Details of the Integration Methodology

As briefly explained above, nano-fabrication produces switching nano-crossbar arrays with structures or individual components having varied properties. Mentioned factors introduce random characteristics of which need to be carefully considered by synthesis process. For instance, a competent synthesis methodology must consider basic technology preference for switching elements and defect or fault rate of the

given nano-crossbar. Presented synthesis methodology in this study comprehensively covers the all specified factors and provides optimization algorithms for each step of the process. A diagram summary demonstrating every step of the method with annotation showing the certain research tasks is given in Figure 3.

First step of the synthesis process of a nano-crossbar involves the decision of switching technology which will be explained elaborately in Section II. Main purpose is to determine which of the diode/memristor, FET, or four-terminal based components are to be used. This step is one of the most important procedures determining the size of the nano-crossbar. Production with diode/memristor based technologies as well as with FET are explained and then logic synthesis design with four-terminal based switches is given.

Second step of the synthesis process of a nano-crossbar covers the permanent faults (defects forming in the course of fabrication) and the tolerance aspects, which will be described in Section III. Main purpose is to obtain a valid realization of a given logic function using two distinct approaches titled as defect-aware and defect-unaware. First method employs faults existing in nano-crossbar during the realization of logic function hence the name aware. Second method avoids the faults by attempting to find a fault-free region of nano-crossbar at the beginning so realization of given logic function is straightforward at the end.

Third step of the synthesis process of a nano-crossbar covers the performance analysis and optimization, which will be explained in Section IV. Main purpose is to analyse delay and power consumption of arrays by showing their dependencies on the properties of the given function as well as specifics of the used technology.

Final step of the synthesis process of a nano-crossbar involves the analysis of transient faults. Main purpose is to determine the effect of transient faults to the operational capacity of nano-crossbar and calculate fault tolerance performance.

Section V is devoted to further elucidate the proposed synthesis process with a case study.

II. Logic Synthesis

In the beginning of the logic synthesis process, the crossbar technology, *i.e.*, diode, memristor, FET, or four-terminal, needs to be determined based on the following criteria:

- Crossbar size (number of rows and columns)
- Number of outputs (single or multiple function realization)
- Fabrication complexity
- Power and delay specifications
- Application specifications

A decision can be made on the importance and priority of the listed items, depending on the application. For example, if an application includes also memory, then the memristor technology can be chosen for the realization of logic functions, since the memristor can also be used as a memory unit. Thus, these components can be fabricated using the same technique.

On the other hand, the realization of a logic function using a diode or a memristor based crossbar requires less number of crosspoints than those in the FET based crossbar. However, the FET based designs consume less power than the diode/memristor-based designs. Moreover, the four-terminal based crossbar includes less number of crosspoints than other crossbar designs [15].

As follows in the first part, we examine the logic synthesis techniques developed for diode, memristor, and FET based nanoarrays in the literature; we formulate their crossbar sizes needed to implement given functions. We also present results for four-terminal switch based arrays to synthesize multi-output functions.

In the second part, we present a new two-level synthesis technique for memristor based nano arrays and compare it with other techniques in the literature.

A. Area Comparisons for Different Crossbar Technologies

We present the logic synthesis step of the integration methodology, considering only the number of crosspoints in the crossbar arrays, which is actually the size of the crossbar array. The size of an crossbar array including diode, memristor, FET, and four-terminal is given as follows, where n is the number of logic functions (the number of outputs); f_i denotes the i^{th} logic function, and f_i^D stands for its dual with $1 \leq i \leq n$.

- **Diode**:$(\#\ of\ products\ of\ all\ f_i) + n) \times ((\#\ of\ literals\ in\ f) + n)$
- **Memristor**: $((\#\ of\ products\ of\ all\ f_i) + n) \times ((\#\ of\ literals\ in\ f) + 2n)$ **[worst-case]**
- **FET**: $(\#\ of\ literals\ in\ f + n) \times ((\#\ of\ products\ of\ all\ f_i) + (\#\ of\ products\ of\ all\ f_i^D))$
- **Four-terminal**: $(largest\ of\ \#\ of\ products\ \#\ in\ f_i^D s) \times (()\#\ of\ products\ of\ all\ f_i) + n - 1)$ **[worst-case]**

As mentioned in Section I, the logic synthesis on diode and memristive based crossbars is similar to the PLA like synthesis. Thus, the techniques, such as product sharing and phase changing used in the PLA design, are also applicable in these designs. Since, the array sizes can be further reduced using the product sharing, these array size formulations can be considered as an upper bound for the logic synthesis techniques.

For the single and multiple output function realization, the synthesis methodology for FET crossbar does not allow us to produce multi-level logic synthesis, only two-level approach can be used [23]. However, multi-level logic synthesis approach is applicable for the diode and memristive crossbars [27]. Therefore, the optimization on the array size still demands further research for the diode and memristor based designs.

The logic synthesis using four-terminal crossbars, generally known as switching lattices, is a new method. As shown in [3], Altun presented a useful logic synthesis technique for the switching lattices. However, this method cannot find the optimal solution in terms of the lattice size. Therefore, new specific logic synthesis methodologies are needed to be presented. As shown in [12] and [15], optimal synthesis

TABLE I
ARRAY SIZE COMPARISON OF DIODE, MEMRISTOR, FET, AND FOUR-TERMINAL SWITCH BASED NANOARRAYS ON MULTIPLE OUTPUT FUNCTIONS

Benchmarks	Diode	Memristor	FET	Four-Terminal [1]
rd53	442	560	819	120
squar5	594	884	900	**108**
bw	1900	3300	1824	**441**
inc	897	1280	2231	235
rd73	2210	2620	4658	606
misex1	437	600	1334	**126**
sqrt8	840	1032	1340	165
ex5p	10823	19596	32864	2664
rd84	5180	6240	10960	2320
clip	2875	3528	6256	685
apex4	16835	25480	72335	7308
sao2	1488	1764	3672	476
ex1010	8820	11800	57960	5958

TABLE II
AREA COMPARISON OF TWO-LEVEL LOGIC SYNTHESIS ALGORITHMS: INITIAL GREEDY ALGORITHM (GA-INITIAL) AND PROPOSED GREEDY ALGORITHM (PGA) WITH OPTIMAL (BRUTE FORCE) AND BASIC [29] APPROACHES

Benchmarks	Basic [29]	GA-initial [28]	PGA	Optimal
rd53	560	416	416	416
squar5	884	858	832	780
inc	1280	1280	1280	1248
rd73	2620	2620	1940	1940
misex1	600	750	600	600
sqrt8	1032	648	648	648
ex5p	19596	19312	19312	**
rd84	6240	6072	4584	4584
clip	3528	3500	3388	3332
sao2	1764	1176	1372	1176
ex1010	11800	11800	11800	11800
alu4	25696	16544	16544	16192
b12	2544	1872	1776	1776
table5	11136	11136	11136	**
vg2	7854	7854	7854	7854

** Time exceeds 600 seconds

methodologies are provided. In addition, there are decomposition based techniques such as XOR based [14], [7], p-circuit [5] and dimension reducibility [6] decompositions as well.

However, all of these studies were only focused on the realization of a single logic function using switching lattices. On the other hand, in [1], three main steps are presented to realize the multiple functions using switching lattices. These steps are given as follows: 1) find the realization of each logic function using a switching lattice; 2) merge these lattices into a single lattice; 3) check if these lattices can be realized using a smaller number of rows and columns such that the final lattice includes a small number of four-terminal switches.

This article is the first to present the sizes of diode, memristor, FET, and four-terminal based crossbar arrays on the multiple output functions. The results are given in Table I. Note that the results given in bold under the four-terminal column indicates that they are found using the approximate algorithm of [1], following the three steps described above. On the other hand, the other results are found using a divide and conquer method, based on the divide and synthesize (DS) method of [1], following the first two steps described above.

As can be observed from Table I, the four-terminal based crossbar arrays include significantly less number of crosspoints when compared to the diode, memristor, and FET based crossbar arrays.

B. Proposed Algorithms for two-level multi output synthesis for Memristive Arrays

In memristive crossbar arrays, function outputs' and their negations are produced [29]. However production order changes the array size. We have called phase_combination-0 and phase_combination-1 realization, if realization is happed based on an output itself and its negation, respectively (note that in phase_combination-1, first its negation is produced [28]). Based on this property array size of memristive crossbars can be optimized.

In our previous study [28], we have proposed a greedy algorithm, which only considers output, individually and doesn't consider output interrelation (no product sharing at analysis). We can call this algorithm as "initial Greedy Algorithm (GA-initial)". On the other hand our new greedy algorithm considers product sharing with the change of phase.

Proposed Greedy Algorithm **(PGA)**: investigates outputs collectively by changing phase of outputs, one at a time, starting from phase_combination-0 and phase_combination-1. Compares them; if any of the changed phase combination has less # of total product; then keeps that phase and continues searching with changing other outputs phase one at a time. Algorithm steps are as follows:

1) Step: Compare product numbers of phase_combination-0 and phase_combination-1, Set Reference_Number as minimum product number and Save the phase combination as Reference_Phase,
2) Step: Create two sets of candidate phase combination; set0: which has only one phase-1 and set1: which has only one phase-0,
3) Step: Find the phase combinations which yields minimum number of products for the function in set0 and set1.
4) Step: Decide; the phase combination found in step-2 is in set0 or set1. If it is in set0 further changes will be to 1, and vice versa.
5) Step: Decide; if the product number of the phase combination is less than Reference_Number. If Yes, make that number of product as Reference_Number and Save the phase combination as Reference_Phase; If NO, Jump to step-7
6) Step: According to decision in step-3, Create a set includes phase combinations by changing phases of outputs other than the one which have been already changed in Reference_Phase compared to phase_combination-0 or phase_combination-1.
7) Step: While Reference_Number is higher or equal to found in step-5, repeat step-4
8) Step: Finalize phase combinations as Reference_Phase.

For example, if we are given a function which has three outputs. First we check phases 000 and 111 (phase_combination-0 and phase_combination-1). Then calculate product numbers of phase in phase sets; set0: 001,010,100 (changing zeros

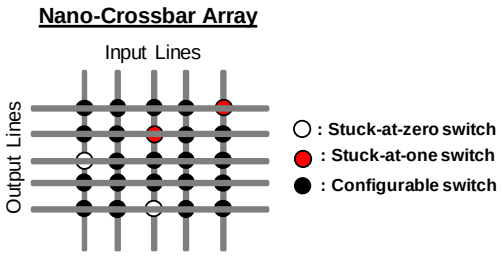

Fig. 4. Nano-crossbar array with faulty/defective crosspoints.

to one), and set1: 111,110,101,011 (Changing ones to zero) (Note that, we change one at a times). Lets say we compared product numbers and found that phase 110 yields the least number product. Then we continue from the phase 110, with changing ones to zero. Means, we check 100,010 and compare the results with result of phase 110. If there is one which yields less product number chose it, or chose the phase 110 for the final phase combination of the function.

To evaluate the algorithms and compare the results, we use espresso and MATLAB on a 3.20 GHz Intel Core i7 CPU (only single core used) with 4GB memory. Results are shown in Table II. Algorithm performances differs function to function. For six of the functions PGA results less area size than GA-initial. Overall, PGA out-competes GA-initial Yet, For two (clip and sao2) of the functions GA-initial results less area size than PGA. For the rest they give same result. GA-initials major advantages is that it has almost no time cost. It can be used for any case.

III. Defect/Fault Tolerance

In this section, we will investigate defects or faults with categorizing them as permanent (naming defects) and transient (naming faults). As mentioned in Section I, crossbars tend to be fabricated with defects. Also, particular transient faults can occur in the field. Defect tolerance basically means finding defect-free region or crosspoint which can still be employed during logic synthesizing. On the other hand, faults can only be tolerated by redundancy, since they occur transiently. Yet sensitivity analysis can be made for both types. Defect model can be found in Figure 4 demonstrating stuck-at-0 (open) and stuck-at-1 (close). Their features can be summarized as:

- **Permanent Faults** occur mostly in fabrication and are tolerated in post-fabrication by redundancy and reconfigurability (mapping).
- **Transient Faults** occur in field and are tolerated in field by only redundancy

A. Defect Tolerance for Diode, Memristor and FET

Defect tolerance is achieved by realizing a target logic functions on a defective crossbar using row and column permutations. This problem is considered as NP-complete [20]. For the worst-case, $N!M!$ permutations are required to find a successful mapping for $N \times M$ crossbar. Algorithms in the literature use defect-unaware or defect-aware approach.

Defect-unaware algorithms aim to find the largest possible $k \times k$ defect-free sub-crossbar from a defective $N \times N$ crossbar where $k \leq N$. The algorithms are inefficient for high fault rates - obtained k values are much smaller than N. In this regard, defect-aware algorithms perform much more satisfactorily [25]. Detailed analysis of both approaches can be found in [26].

Defect-aware considers the defect characteristics (stuck-at-0 or stuck-at-1), then decide which switch to employ during the mapping. In our previous work [25], we have proposed an efficient heuristic algorithms which aims to match defected crossbar and the function solution crossbar. For this, it defines crossbars as matrix. Therefore it can perform sorting, matching and backtracking steps efficiently. It makes repetition for a limit of permutation. This controls heuristic feature of the algorithm.

B. Defect Tolerance for Four-terminal

Four-terminal defect tolerance demands a different approach than the methods we have covered so far. For this reason, we present a novel method, which is firstly introduced in this paper. The proposed method utilizes a prior sensitivity analysis of crossbar (latttice) to specify critical switches, and strengthens them with proposed mitigation factors. The same naming conventions are applicable, regarding defects which are categorized as stuck-at-0 (SA0) and stuck-at-1 (SA1). Furthermore, we describe a new defect that can be considered for crossbars (lattices), that consists in changing a switching literal in a cell $c_{i,j}$ of the lattice with a literal that is in a adjacent cell (i.e., $c_{i-1,j}$, $c_{i+1,j}$,$c_{i,j-1}$, or $c_{i,j+1}$). We denote this fault model as Adjacent Cellular Fault Model (ACFM). In addition, we follow the same terminology adopted in [3] and [12] by addressing crossbar as lattice and switch as cell to be consistent and emphasize the distinction of four-terminal approach. Finally, it should be noted that as opposed the previous sections, we provide a more detailed explanation due to original technical contribution presented in this section.

Defect Injection Methodology

This section introduces the algorithms for the defect injection in the SAFM and in the ACFM.

The two procedures are repeated for each cell of the lattice. Once the "defective" lattice is obtained, both the algorithms generate all the possible 2^n inputs (where n is the number of variables). For each input, the simulation algorithms compare the given output with the correct one. The two defect injection algorithms (for the SAFM and the ACFM) differ in the calculation of number of defective outputs in the two fault models, as explained in the following subsections.

Fault Model 1: Stuck at Faults (SAFs)

The sensitivity of the decomposition algorithm on a given crossbar is analyzed face to SA0 and SA1 as being the model widely adopted today for memristive type of crossbar. As there is no consensus today on the fault distribution, we have, first of all, chosen a uniform distribution for each type of SA0 and SA1 [8]. The fault density considered may be up to 10% of the crossbar, all faults being independent, as reported in [8]. The fault injection in the above lattices is performed substituting

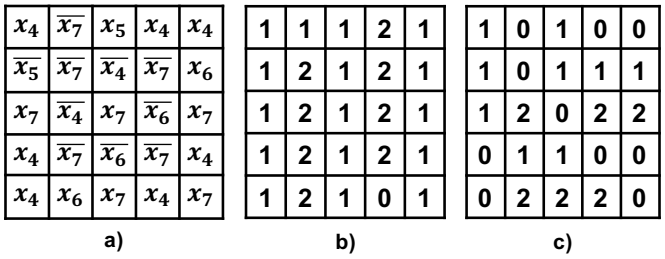

Fig. 5. a) Lattice design for the example function f and its sensitivity map for b) SAO and c) SA1.

a single cell with an always stuck-at 1 (SA1) or stack-at 0 (SA0) cell.

Let E^0_{ij} (resp., E^1_{ij}), with $1 \leq i \leq r$, $1 \leq j \leq s$, be the number of defective outputs with a SA0 (resp., SA1) in the cell (i, j) of the given lattice. Note that $0 \leq \{E^0_{ij},\ E^1_{ij}\} \leq 2^n$. Moreover, when E^0_{ij} (resp., E^1_{ij}) is equal to 0 we have that, for any possible input, the lattice output is never changed by the SAF in the cell $c_{i,j}$. In this case, we call the cell $c_{i,j}$ robust w.r.t. SA0 (resp., SA1). Let R^0 (resp., R^1) be the total number of robust cells w.r.t. SA0 (resp., SA1) in the lattice. Finally, let $E^0 = \sum_{i=1}^{i=r}\sum_{j=1}^{j=s} E^0_{ij}$ (resp., $E^1 = \sum_{i=1}^{i=r}\sum_{j=1}^{j=s} E^1_{ij}$) be the total number of defective outputs with SA0 (resp. SA1) in the simulation. For an example of function $f = x_4\overline{x_5}x_7 + \overline{x_4}x_6\overline{x_7} + \overline{x_4}x_5\overline{x_6}x_7 + x_4\overline{x_6x_7} + x_4x_6x_7$ realized in Figure 5(a) (with the method in [3]), in the Figure 5(b) (resp., 5(c) shows the map containing E^0_{ij} (resp., E^1_{ij}) in each cell.

Fault Model 2: Adjacent Cellular Faults (ACFs)

In the classical CFM [11] for CMOS circuits it is assumed that a fault modifies the behavior of exactly one node v in a given circuit C and that the modified behavior is still combinational. This fault for a switching lattice L can be described as follows: a *cellular fault* in L is a tuple $(c_{i,j}, l_c, l_f)$, where $c_{i,j}$ is the cell of the lattice L (i.e., fault location), l_c is the correct controlling literal in $c_{i,j}$, and $l_f (\neq l_c)$ is the faulty controlling literal. We denote *adjacent cellular fault* a cellular fault where the faulty literal l_f is in an adjacent cell. More precisely:

Definition 1: Let $l_{h,k}$ be the literal in the cell $c_{h,k}$ of a lattice L. We have that:

1) A *Left Adjacent Cellular Fault (L-ACF)* is the cellular fault $(c_{i,j}, l_{i,j}, l_{i,j-1})$,
2) A *Right Adjacent Cellular Fault (R-ACF)* is the cellular fault $(c_{i,j}, l_{i,j}, l_{i,j+1})$,
3) A *Bottom Adjacent Cellular Fault (B-ACF)* is the cellular fault $(c_{i,j}, l_{i,j}, l_{i-1,j})$,
4) A *Top Adjacent Cellular Fault (T-ACF)* is the cellular fault $(c_{i,j}, l_{i,j}, l_{i+1,j})$.

Let E^L_{ij} (resp., E^R_{ij}, E^B_{ij}, and E^T_{ij}), with $1 \leq i \leq r$, $1 \leq j \leq s$, be the number of defective outputs with a L-ACF (resp., R-ACF, B-ACF, and T-ACF) in the cell $c_{i,j}$ of the given lattice. Let R^a (with $a \in \{L, R, B, T\}$) be the total number of robust cells w.r.t. a-ACF in the lattice. Finally, let $E^a = \sum_{i=1}^{i=r}\sum_{j=1}^{j=s} E^a_{ij}$ be the total number of defective outputs with a-ACF in the simulation.

Metrics used for Sensitivity Analysis

In order to evaluate the sensitivity of a lattice to SAF and ACNF defects, we propose two metrics. The first one measures the average number of defective outputs considering sensitive cells to defects only. The second one measures the average number of defective outputs in the entire lattice. Note that the total number of cells is the area of the lattice (i.e., $r \cdot s$), the number of non-robust cells for SA0 (resp., SA1) is $r \cdot s - R^0$ (resp., $r \cdot s - R^1$), and 2^n is the total number of inputs. Moreover, the number of non-robust cells for a-ACF (with $a \in \{L, R, B, T\}$) is $r \cdot s - R^a$.

The *sensitivity of defective cells* is the total number of inputs that give an uncorrected output (E^i, with $i \in \{0, 1, L, R, B, T\}$) divided by the total number of inputs (2^n), for each non-robust cell. The metric can be expressed as: $S^i_C = E^i/(2^n \cdot (r \cdot s - R^i))$ for each fault $i \in \{0, 1, L, R, B, T\}$.

The *sensitivity of lattice* is the total number of inputs that give an uncorrected output divided by the total number of inputs for each cell: In particular, $S^i_L = E^i/(2^n \cdot r \cdot s)$, with $i \in \{0, 1, L, R, B, T\}$.

Benchmarks and Simulations

The defect simulations have been run on a machine with two AMD Opteron 4274HE for a total of 16 CPUs at 2.5 GHz and 128 GByte of main memory, running Linux CentOS 7. The benchmarks functions are expressed in PLA form and are taken from a subset of LGSynth93 [31]. A total of about 580 functions were considered, and each output of a function is implemented as a separate Boolean function.

The software used for simulations is written in C++. We used ESPRESSO to implement the method described in [3], and a collection of Python scripts for computing minimum-area lattices by transformation to a series of SAT problems, to simulate the results reported in [12]. Each SAT execution is stopped after ten minutes.

In Table III, we report a sample of benchmark functions and their sensitivity values, according to the metrics presented before. In particular, Table III refers to lattice synthesized as described in [3] and [12]. The benchmarks that are present in Table III with dual method were stopped after ten minutes of SAT execution, but that was not the case for the rest.

More precisely, in both methods, the first column reports the name and the number of the considered output of each function. The following columns report dimension ($r \times s$) required for the synthesis of a given function according to each decomposition method, and the number of input variables n. Columns from 4 to 11 refers to Stuck At fault model, columns from 12 to 27 to Adjacent Cellular fault model showing the total number of errors E, the Sensitivity of defective cells S_C, the Sensitivity of lattice S_L and the percentage of robust cells $\%R/r \times s$.

Table IV describes the overall results for the benchmarks we have considered. It also shows the average values for the considered metrics. We can note that the percentage of cells that are considered robust according to our metrics is higher in the first approach [3]. This is due to the more constrained structure of the lattices produced by the first synthesis method. Indeed, the method proposed in [3] computes a lattice for f

TABLE III
A SAMPLE OF BENCHMARK FUNCTIONS SYNTHESIZED WITH [3] AND [12] APPROACHES AND THEIR SENSITIVITY VALUES

name	Benchmark Size		Stuck at Fault Model								Adjacent Cellular Fault Model													
	$r \times s$	n	E^0	S_C^0	S_L^0	$\%\frac{R^0}{r\times s}$	E^1	S_C^1	S_L^1	$\%\frac{R^1}{r\times s}$	E^L	S_C^L	S_L^L	$\%\frac{R^L}{r\times s}$	E^R	S_C^R	S_L^R	$\%\frac{R^R}{r\times s}$	E^T	S_C^T	S_L^T	$\%\frac{R^T}{r\times s}$	E^B	S_C^B	S_L^B	$\%\frac{R^B}{r\times s}$
Synthesis with Dual Method [3]																										
add6(1)	6×6	4	19	0.06	0.03	47%	9	0.06	0.02	75%	13	0.04	0.02	47%	9	0.04	0.02	58%	6	0.03	0.01	66%	9	0.04	0.02	58%
alu2(2)	11×10	8	462	0.03	0.02	35%	121	0.02	0.01	80%	155	0.01	0.01	54%	187	0.01	0.01	51%	110	0.02	0	76%	32	0.01	0	80%
b11(1)	3×6	7	28	0.02	0.01	44%	73	0.03	0.03	6%	23	0.01	0.01	11%	23	0.01	0.01	11%	65	0.03	0.03	0%	59	0.03	0.03	5%
dc2(0)	4×6	7	117	0.05	0.04	17%	162	0.08	0.05	33%	78	0.01	0	80%	99	0.01	0	79%	160	0.01	0	79%	102	0.01	0	79%
exam(5)	6×11	9	1868	0.07	0.06	17%	131	0.02	0.01	74%	1114	0.04	0.03	15%	2079	0.06	0.06	6%	1208	0.04	0.04	8%	1120	0.04	0.03	5%
rd53(2)	16×16	5	144	0.03	0.02	44%	66	0.03	0.01	74%	77	0.03	0.01	64%	77	0.03	0.01	64%	59	0.025	0.01	71%	59	0.025	0.01	71%
z4(2)	12×12	5	70	0.03	0.02	51%	14	0.03	0	90%	40	0.02	0.01	64%	38	0.02	0.01	65%	12	0.02	0	83%	14	0.02	0	82%
Synthesis with Quantified Boolean Logic [12]																										
add6(1)	5×3	4	31	0.15	0.13	13%	32	0.14	0.13	7%	13	0.13	0.13	0%	9	0.13	0.13	0%	6	0.13	0.13	0%	9	0.13	0.13	0%
alu2(2)	7×3	8	464	0.1	0.09	14%	384	0.08	0.07	5%	326	0.06	0.06	5%	316	0.06	0.06	10%	291	0.05	0.05	0%	278	0.05	0.05	0%
b11(1)	3×5	7	45	0.03	0.02	7%	128	0.07	0.07	7%	44	0.03	0.02	13%	35	0.02	0.02	13%	125	0.07	0.07	7%	121	0.06	0.06	0%
dc2(0)	4×4	7	104	0.06	0.05	13%	132	0.07	0.06	13%	100	0.05	0.05	6%	54	0.03	0.03	6%	121	0.06	0.06	6%	66	0.03	0.03	6%
mlp4(6)	4×3	4	19	0.12	0.10	17%	30	0.19	0.16	17%	25	0.14	0.13	8%	22	0.11	0.11	0%	18	0.10	0.09	8%	17	0.11	0.09	17%
p3(8)	7×3	8	359	0.07	0.07	5%	192	0.04	0.04	10%	229	0.05	0.04	10%	239	0.05	0.04	5%	150	0.03	0.03	0%	140	0.03	0.03	0%
prom2(0)	6×4	8	148	0.02	0.02	0%	492	0.08	0.08	0%	297	0.05	0.05	8%	295	0.05	0.05	8%	332	0.05	0.05	0%	358	0.06	0.06	0%

TABLE IV
OVERALL RESULTS OF THE SIMULATIONS

Synthesis Method	Average area	Average n	S_C^0	S_L^0	$\%\frac{R^0}{r\times s}$	S_C^1	S_L^1	$\%\frac{R^1}{r\times s}$	S_C^R	S_L^R	$\%\frac{R^R}{r\times s}$	S_C^L	S_L^L	$\%\frac{R^L}{r\times s}$	S_C^T	S_L^T	$\%\frac{R^T}{r\times s}$	S_C^B	S_L^B	$\%\frac{R^B}{r\times s}$
[3]	30	6	0.05	0.05	20%	0.06	0.05	29%	0.02	0.02	18%	0.02	0.02	18%	0.04	0.03	16%	0.03	0.03	16%
[12]	15	7	0.07	0.06	9%	0.07	0.07	8%	0.06	0.06	5%	0.06	0.06	5%	0.08	0.07	4%	0.08	0.07	4%

and its dual that is in general less compact than the lattice given by [12] (see, the column Average area in Table IV). Moreover, we can note that the sensitivity of the lattice is quite low for both methods. In fact, the experiments show that, in general, non-robust cells compute a defective output for a very limited number of inputs. In particular Adjacent Cellular faults have a lower sensitivity with respect to the Stuck At faults. This is due to the fact that if two adjacent cells contains the same value this kind of fault does not occur.

Mitigation by Defect Avoidance

From the above results, it can be seen that the two analyzed mapping algorithms show different sensitivities of the output of a given function. As a matter of fact, the more restrictive an algorithm is in terms of area (closer to optimal solution), the higher the defect sensitivity of the output to cell defect. It is mandatory to include the mapping algorithm defect-avoidance heuristics, but hardware-level defect tolerant scheme may also be necessary, especially in the case of high defect densities. Redundancy schemes that can be used are inspired from memory testing and repairing structures published in the early 2000 [16]. They can be used at a column level, or block level, as the basic computation unit of memristor array is not the memristive cell as in classical CMOS based memories, but an entire column. Therefore, several redundant columns can be added to the initial design to be able to detect and replace the memristive RAM affected columns but also to overcome the potential SAD that affect the redundant parts. Mapping of logic functions on crossbar arrays are thus divided into two main phases: mapping phase to write the parameters of functions in the memristive array and a read operation to check the results from the crossbar. The objective here is to identify at the writing time, if common literals and other multiple-choice literals of the function have to be mapped on highly critical cells.

Mitigation for Stuck at Faults:

In order to mitigate the sensitivity of a lattice to SAD, we propose the following possible strategy applied to the synthesis method proposed in [3] which has been proven as less sensitive to SAD impact on the output functions: (1) For a given mapped function, if a potential SA0, SA1 defect affects a robust cell identified by the defect injection campaign, the lattice still computes the correct output, thus we do not need any mitigation with defect tolerant design. (2) However, if an injected defect occurs in a multiple-choice cell, if a different literal can be chosen to make the cell robust, we change the literal with the new one. (3) Otherwise, if the injected SA0 defect is proven as being critical for the output value, the column that contains that defective cell has to be replaced by spare columns. In case of an SA1 the row that contains the defective cell has to be replaced by a spare row. Note that, in this case, the output still provides a correct function f from top to bottom, but the function from left to right could be changed and become a function which will not be dual of f anymore.

As an example, consider the lattice synthesized in Figure 6(a) with $f = x_4\overline{x}_5x_7 + \overline{x}_4x_6\overline{x}_7 + \overline{x}_4x_5\overline{x}_6x_7 + x_4\overline{x}_6\overline{x}_7 + x_4x_6x_7$ by using synthesis method presented in [3]. The example shows one case of mitigation of 3 independent SAD affecting the lattice implementing the function, yielding an approximate 10% defects. In Figure 6, SA1 cells are marked in blue and SA0 cells a remarked in red.

To avoid output errors due to these SAD we have used the following strategy:

1) Identify robust cells for a given function mapping. Example: the defect in first row, fifth column is non-influent on the value of the output (robust cell), the sensitivity map, obtained through defect injection campaign, shows that this cell is not sensitive to SA1 for the mapped function.
2) Identify the swapping of literals during synthesis process on a column of an high sensitive cell. Example: the defect

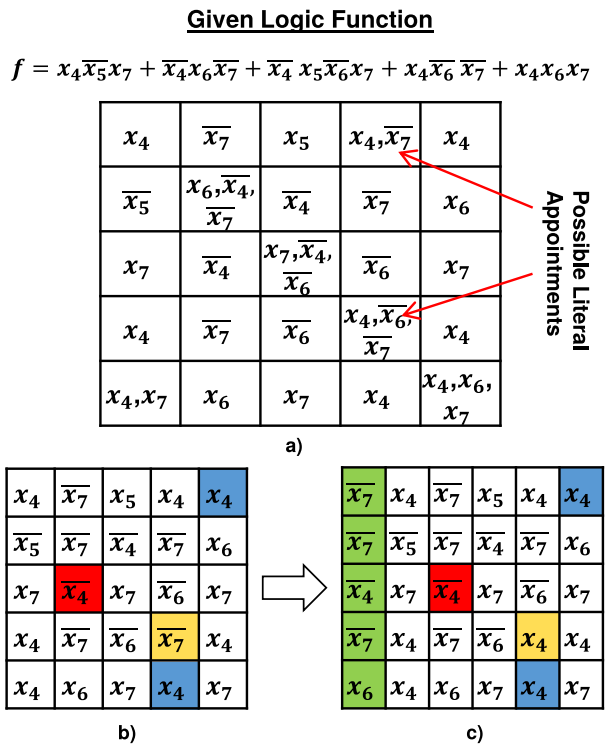

Fig. 6. a) defect-free lattice; b) lattice with defects: SA0 in red and SA1 in blue; and c) lattice with the defect fixed.

in fifth row, fourth column is sensitive to SA1. The former choice in the yellow cell was $\overline{x_7}$, choosing x_4 the fifth row, fourth column cell, if it will be affected by a SA1 at fabrication time, will not affect the output of the function.

3) Identify the critical cell for the output value and add a spare column per critical cell. Example: the defect in the third row and second column will influence the value of the output and no swapping operands is possible, thus the only solution remains to add a spare column (in green) identical to the column containing the SA0 defect, and perform the spare and replace strategy. By using spare columns, the mapping algorithm can eliminate columns of the lattice susceptible to affect the output value of the function in case SA defect appear at fabrication or in the field.

Mitigation for Adjacent Cellular Faults:

We can note that adjacent cells containing the same controlling literal are robust to adjacent cellular faults. For this reason, in this fault model, it is of huge importance to maximize the number of adjacent cells with the same controlling literal. For this purpose, the columns and rows, obtained by the synthesis method proposed in [3], can be permuted. The properties of the lattice synthesized with [3] guarantee that row and column permutations do not effect the Boolean function computed by the switching lattice.

C. Transient Fault Tolerance

Regarding transient faults, there are two approaches: redundancy based and manipulation of logic function to obtain a more fault tolerant design. Nevertheless, it should be noted that since nano-crossbars are in the early stage of development, there is a lack of in-field data regarding transient faults. For this reason, second approach towards transient fault tolerance is methodology independent and rather focuses on the intrinsic features of the given logic functions. As mentioned, transient faults can be tolerated with redundancy. Inserting redundant components can be constructed with adding extra rows and/or columns as shown [18] [4].

Furthermore, provided certain conditions, particular logic functions are inherently tolerant to transient faults limited to certain switches of crossbar as show in [25]. Mentioned inherent tolerance capability varies depending on the design. For example, consider a design having low logic inclusion ratio (IR), meaning less number of crosspoints of a crossbar are used, that means this design is more tolerant to stuck-at-0 faults. Logic synthesis (design) should be made with regard to this, since there are multiple design solution.

Similarly, if the target function can be realized with high IR, then a technology, the one which tends to have stuck-at-1 faults, should be preferred. for example function $f = x_1x_2 + x_2x_3 + \overline{x_3}x_4$ can also be written as $f' = x_1x_2\overline{x_3} + x_2x_3 + \overline{x_3}x_4 = f$, therefore IR can be increased.

Fault sensitivity analysis can be made using Monte Carlo analysis, yet it is costly. Since we know the dynamics of the fault tolerance we can calculate sensitivity (fault tolerance performance) directly with algebraic equations. Equation parameters consists of crossbar dimensions, inclusion ratio, fault occurrence possibility and number of tolerable crosspoint. For further information please refer to [25].

IV. Performance Optimization

Area performance of crossbar is previously investigated in Section II. Here, we will analyze the delay and power depending on number of products and/or literals. In our previous study [28], we have concluded this analysis for memristive crossbars.

Here we have used an approach of multiplying resistive and capacitive loads for $delay$. This helps us to calculate maximum frequency with $1/delay$. In the delay analysis for diode, we see that "number of columns" and "load resistor" dominate the capacitive load and resistive load respectively. For memristive crossbars, it is proportional to constant 7. Considering FET, we need to calculate resistive loads and capacitive loads from the longest path for the worst-case scenario. Lastly, four-terminal delay is directly proportional to the longest path on the lattice, same as FET. Longest path on the lattice can be calculated as explained in [3]:

$$L_{long\_path} = \begin{cases} R & R \leq 2 \bigcup C \leq 1 \\ 3\frac{R-2}{2}\frac{C}{2} + \frac{2+(-1)^R+(-1)^C}{2} & R > 2 \bigcap C > 1 \end{cases}$$

where R: number of rows, C: number of columns, L_{long_path}: length of the longest path on the lattice.

Delay Analysis:

- **Diode**: $\propto (load\_resistor) \times (\#\ of\ literals\ in\ f)$
- **Memristor**: $\propto constant : 7$ [28]

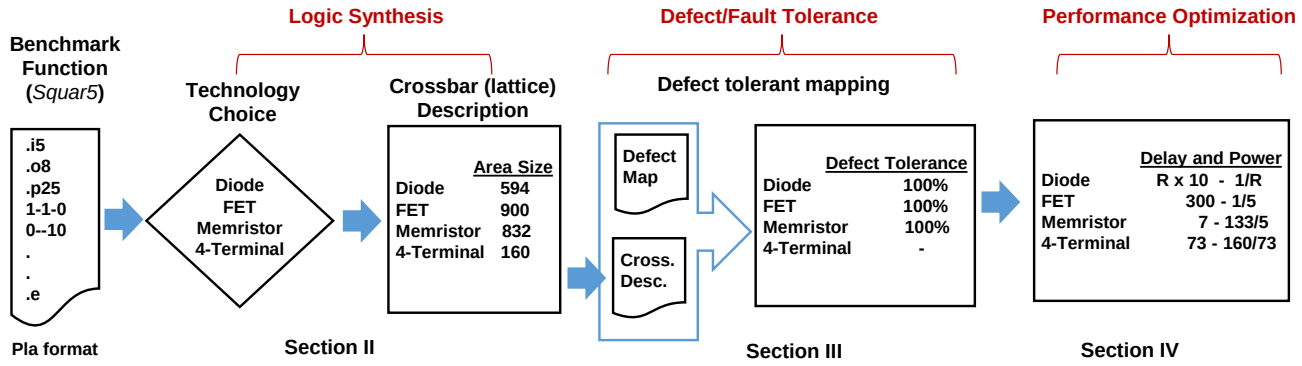

Fig. 7. Whole integrated synthesis pipeline of benchmark function *squar5*. R denotes the $load\_resistor$.

- **FET**: $\propto (degree\ of\ the\ largest\ product\ in\ f\ and\ f^D) \times ((\#\ of\ products\ in\ f) + (\#\ of\ products\ in\ f^D) + \#\ of\ literals\ in\ f)$
- **Four-terminal**: $\propto L_{long\_path}$

Power consumption is total energy used in unit time. Therefore we estimate consumed total energy in a period, afterwards we divided it with $delay$ ($delay$ is proportional with one period maximum frequency). Because we have estimated circuit will used in maximum frequency.

For power of diode crossbar is dominated by static power. Thus, it is inversely proportional only with load resistor. Load resistor can be considered as 10 times of diode's inner resistance. Memristor's total energy consumption is proportional to total memristor count, because it is assumed that all memristors are changes their states one time only [28]. Total energy consumption of FET crossbars is directly proportional to the capacitance of output node and nodes related to the output node. For four-terminal, we assume all crosspoints consume energy at worst-case scenario. Finally, we divided energyy consumption by the delay to find the complexity of power consumption.

Power Analysis:

- **Diode**:$\propto 1\ /\ (load\_resistor)$
- **Memristor**: $\propto (total\ memristor\ count\ in\ crossbar)/7$ [28] [24]
- **FET**: $\propto 1\ /\ (degree\ of\ the\ largest\ product\ in\ f\ and\ f^D)$ (@ maximum frequency)
- **Four-terminal**: $\propto\ ((\#\ of\ products\ in\ rows) \times (\#\ of\ products\ in\ columns))\ /\ L_{long\_path}$

V. A Case Study for Benchmark Function *Squar5*

In this section, we demonstrate the whole process with an example. We have chosen the benchmark function *Squar5* which has 5 inputs, 25 products and 8 outputs in pla format. Diagram of the process is given in Figure V.

First, we need to evaluate different technologies such as diode, FET, memristor or four-terminal. Using logic synthesis, we can generate function descriptions in crossbar (lattice) form and obtain area sizes utilizing equations in Section II. Results are as follows:

Diode: $(\#\ of\ products\ of\ all\ f_{oi}) + n) \times ((\#\ of\ literals\ in\ f) + n)$ = (25 + 8) × (10 + 8) = 594.

Memristor: $((\#\ of\ products\ of\ all\ f_{oi}) + n) \times ((\#\ of\ literals\ in\ f) + 2n)$ = (25 + 8) × (10 + 16) = 858. However, our proposed greedy algorithm **PGA** decreases the size to 832.

FET: $(\#\ of\ literals\ in\ f + n) \times ((\#\ of\ products\ of\ all\ f_{oi}) + (\#\ of\ products\ of\ all\ f_{oi}^D))$ = (10 + 8) × (25 + 25) = 900. Coincidently, its dual also has the same number of products.

four-terminal: $(degree\ of\ the\ largest\ product\ in\ f_{oi}^D) \times (\#\ of\ products\ of\ all\ f_{oi} + n - 1)$ = 5 x (25 + 8-1) = 160. First term is chosen according to the product which has the maximum number of literals.

Secondly, we use function descriptions and defect map of a crossbar. Applying the defect tolerant logic mapping methods in Section III, it is possible to measure defect/fault performance. Key point is that diode, FET and memristor have a rich literature of methods for reaching 100% defect tolerance for defect rates up to 10% [26] [28]; nevertheless four-terminal is at its infancy in terms of defect tolerance. In this paper, defect performance of only single output functions are studied and we are planning to extend the work into multi output functions as well in the future.

Lastly, we conduct a performance optimization specific to technology dependent delay and power parameters of crossbars. Since related equations are presented in immediate section, only result are given:

Diode: Delay is $R\times$ 10 ($Load\_resistor$ is shown with R). Power is $1/R$. If assume R is 10 times than a diode's inner resistance, then Delay is 100 and Power is $1/10$.

Memristor: Delay is constant and 7. Power is 133/7 $\approx$ 19. The number 133 denotes the number of memristors used in the crossbar.

FET: Delay is 300 and power is 1/5. Including the next four-terminal, FET has the largest delay.

Four-terminal: The longest path is 73, so is the delay. Power is 160/73 $\approx$ 2.

Note that, values only lights the complexity of delay and power, they are not real elapsed time and power consumption. Here we assume, they all fabricated with same technology. For

instance, in order to calculate the delay of diode crossbars, we need to know value of R and the source voltage.

To provide an overall evaluation, four-terminal is the most advantageous choice in terms of area size. However, defect tolerance is poor especially regarding the complexity and computation power needed to conduct experiments for four-terminal. Delay of memristive crossbars is predictable, it doesn't depend on function. Therefore, it could be considered as the most advantageous choice in terms of delay. For power consumption, diode seems to have least consumption, yet it is static power also depends source voltage. On the other hand, FET crossbars has only dynamic power consumption with complementary architecture.

VI. Conclusion and Discussion

In this study, we present a synthesis methodology for crossbar arrays having crosspoints working as FET, diode/resistive/memristive, or four-terminal switch based devices. We cover "logic synthesis", "defect/fault tolerance", and "area-power-delay performance optimization" steps. Presented synthesis methodology provides optimization algorithms for each step of the process as well as their relations and trade-offs.

As a future work, a fully automated software tool will be developed. It takes the target function, the used technology, and the performance specifications (area, delay and/or power consumption) as input and return the optimized crossbar-arrays structure as an output. This software tool to be developed will benefit from the optimization algorithms introduced in this study. It will also make the technology mapping using technology data.

Acknowledgement: We thank Levent Aksoy for his helps to synthesize multi-output functions with four-terminal switch based arrays.

References

[1] L. Aksoy and M. Altun, "A satisfiability-based approximate algorithm for logic synthesis using switching lattices." Design, Automation and Test 2019 (DATE'19), accepted.

[2] D. Alexandrescu, M. Altun, L. Anghel, A. Bernasconi, V. Ciriani, L. Frontini, and M. Tahoori, "Synthesis and performance optimization of a switching nano-crossbar computer," in *Digital System Design (DSD), 2016 Euromicro Conference on*. IEEE, 2016, pp. 334–341.

[3] M. Altun and M. D. Riedel, "Logic synthesis for switching lattices," *IEEE Transactions on Computers*, vol. 61, no. 11, pp. 1588–1600, 2012.

[4] S. Baranov, I. Levin, O. Keren, and M. Karpovsky, "Designing fault tolerant fsm by nano-pla," in *On-Line Testing Symposium, 2009. IOLTS 2009. 15th IEEE International*. IEEE, 2009, pp. 229–234.

[5] A. Bernasconi, V. Ciriani, L. Frontini, V. Liberali, G. Trucco, and T. Villa, "Logic synthesis for switching lattices by decomposition with p-circuits," in *Digital System Design (DSD), 2016 Euromicro Conference on*. IEEE, 2016, pp. 423–430.

[6] A. Bernasconi, V. Ciriani, L. Frontini, and G. Trucco, "Synthesis on switching lattices of dimension-reducible boolean functions," in *Very Large Scale Integration (VLSI-SoC), 2016 IFIP/IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[7] A. Bernasconi, V. Ciriani, L. Frontini, and G. Trucco, "Composition of switching lattices and autosymmetric boolean function synthesis," in *Digital System Design (DSD), 2017 Euromicro Conference on*. IEEE, 2017, pp. 137–144.

[8] C.-Y. Chen, H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu, and F. T. Chen, "Rram defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 180–190, 2015.

[9] Y. Chen, G.-Y. Jung, D. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams, "Nanoscale molecular-switch crossbar circuits," *Nanotechnology*, vol. 14, no. 4, p. 462, 2003.

[10] A. DeHon and B. Gojman, "Crystals and snowflakes: building computation from nanowire crossbars," *Computer*, no. 2, pp. 37–45, 2011.

[11] A. Friedman, "Easily testable iterative systems," *IEEE Transactions on Computers*, vol. C-22, pp. 1061–1064, 1973.

[12] G. Gange, H. Søndergaard, and P. J. Stuckey, "Synthesizing optimal switching lattices," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 1, p. 6, 2014.

[13] Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K.-H. Kim, and C. M. Lieber, "Logic gates and computation from assembled nanowire building blocks," *Science*, vol. 294, no. 5545, pp. 1313–1317, 2001.

[14] M. C. Morgül and M. Altun, "Anahtarlamalı nano dizinler ile lojik devre tasarımı ve boyut optimizasyonu logic circuit design with switching nano arrays and area optimization," in *ELECO*, 2014.

[15] M. C. Morgul and M. Altun, "Synthesis and optimization of switching nanoarrays," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2015 IEEE 18th International Symposium on*. IEEE, 2015, pp. 161–164.

[16] M. Nicolaidis, L. Anghel, and N. Achouri, "Memory defect tolerance architectures for nanotechnologies," *Journal of Electronic Testing*, vol. 21, no. 4, pp. 445–455, 2005.

[17] F. Peker and M. Altun, "A fast hill climbing algorithm for defect and variation tolerant logic mapping of nano-crossbar arrays," *IEEE Transactions on Multi-Scale Computing Systems*, vol. *-accepted-*, 2018.

[18] W. Rao, A. Orailoglu, and R. Karri, "Logic level fault tolerance approaches targeting nanoelectronics plas," in *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*. IEEE, 2007, pp. 1–5.

[19] S. Safaltin, O. Gencer, M. Morgul, L. Aksoy, S. Gurmen, C. Moritz, and M. Altun, "Realization of four-terminal switching lattices: Technology development and circuit modeling." Design, Automation and Test 2019 (DATE'19), accepted.

[20] A. M. S. Shrestha, S. Tayu, and S. Ueno, "Orthogonal ray graphs and nano-pla design." in *ISCAS*, 2009, pp. 2930–2933.

[21] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A: Materials Science & Processing*, vol. 80, no. 6, pp. 1165–1172, 2005.

[22] G. Snider, P. Kuekes, T. Hogg, and R. S. Williams, "Nanoelectronic architectures," *Applied Physics A*, vol. 80, no. 6, pp. 1183–1195, 2005.

[23] G. Snider, P. Kuekes, and R. S. Williams, "Cmos-like logic in defective, nanoscale crossbars," *Nanotechnology*, vol. 15, no. 8, p. 881, 2004.

[24] M. Traiola, M. Barbareschi, and A. Bosio, "Estimating dynamic power consumption for memristor-based cim architecture," *Microelectronics Reliability*, vol. 80, pp. 241–248, 2018.

[25] O. Tunali and M. Altun, "Permanent and transient fault tolerance for reconfigurable nano-crossbar arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 5, pp. 747–760, 2017.

[26] O. Tunali and M. Altun, "A survey of fault-tolerance algorithms for reconfigurable nano-crossbar arrays," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 79:1–79:35, Nov. 2017.

[27] O. Tunali and M. Altun, "Logic synthesis and defect tolerance for memristive crossbar arrays," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2018*, 2018.

[28] O. Tunali, M. C. Morgul, and M. Altun, "Defect-tolerant logic synthesis for memristor crossbars with performance evaluation," *IEEE Micro*, vol. 38, no. 5, pp. 22–31, 2018.

[29] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 335–342.

[30] H. Yan, H. S. Choe, S. Nam, Y. Hu, S. Das, J. F. Klemic, J. C. Ellenbogen, and C. M. Lieber, "Programmable nanowire circuits for nanoprocessors," *Nature*, vol. 470, no. 7333, pp. 240–244, 2011.

[31] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.

Integrated Synthesis Methodology for Crossbar Arrays*

1st M. Ceylan Morgul
3rd Onur Tunali
10th Mustafa Altun
Istanbul Technical University
Istanbul, Turkey
{morgul, onur.tunali, altunmus}@itu.edu.tr

2nd Luca Frontini
5th Valentina Ciriani
Università degli Studi di Milano
Milan, Italy
{luca.frontini, valentina.ciriani}@unimi.it

4th E. Ioana Vatajelu
6th Lorena Anghel
TIMA laboratory
Grenoble-Alpes University
Grenoble, France
{ioana.vatajelu, lorena.anghel}@imag.fr

7th Csaba Andras Moritz
University of Massachusetts, Amherst
Massachusetts, USA
andras@ecs.umass.edu

8th Mircea R. Stan
University of Virginia
Charlottesville, Virginia, USA
mircea@virginia.edu

9th Dan Alexandrescu
IROC Technologies
Grenoble, France
dan.alexandrescu@iroctech.com

ABSTRACT

Nano-crossbar arrays have emerged as area and power efficient structures with an aim of achieving high performance computing beyond the limits of current CMOS. Due to the stochastic nature of nano-fabrication, nano arrays show different properties both in structural and physical device levels compared to conventional technologies. Mentioned factors introduce random characteristics that need to be carefully considered by synthesis process. For instance, a competent synthesis methodology must consider basic technology preference for switching elements, defect or fault rates of the given nano switching array and the variation values as well as their effects on performance metrics including power, delay, and area. Presented synthesis methodology in this study comprehensively covers the all specified factors and provides optimization algorithms for each step of the process.

CCS CONCEPTS

• **Hardware** → *Emerging architectures*; *Emerging tools and methodologies*;

KEYWORDS

Crossbar Arrays, Logic Synthesis, Defect Tolerance, Fault Tolerance, Performance Optimization, Memristor Arrays

1 INTRODUCTION

Nano-crossbars are emerged to be an alternative technology besides CMOS [25]. They are fabricated with relatively cheap bottom-up nano-fabrication techniques rather than using purely lithography based conventional production. Due to the novel manufacturing techniques, fabrics yield to be in regular and dense form [7]. Because of their structure and technology, they are area and power efficient [1].

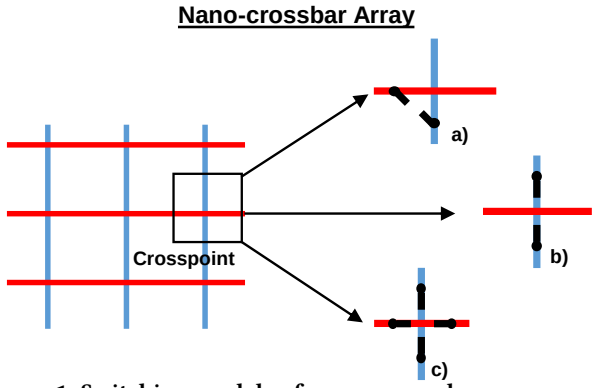

Figure 1: Switching models of a nano-crossbar array: crosspoint as a) two-terminal switch with terminals in the crossed lines, b) two-terminal switch with terminals in the same line, and c) four-terminal switch.

Currently, computing is achieved with crosspoints behaving like switches, either as two-terminal or four-terminal. This is illustrated in Figure 1. Depending on the used technology, a two-terminal switch behaves either as a diode [11], a resistive/memristive switch [18], or a field effect transistor (FET) [19]. Diode and resistive switches correspond to the crosspoint structure in Figure 1(a); here, the switch is controlled by the voltage difference between the terminals. Figure 1(b) shows a FET based switch; here, the red line represents the controlling input. This is a unique opportunity that allows us to integrate well developed conventional circuit design techniques into nano-crossbar arrays. Finally, a novel four-terminal switch is given in Figure 1(c). The controlling input, not shown in the figure, has a separate physical formation from the crossbar that is thoroughly explained for different technologies in [2].

To illustrate their computing approaches, we show examples for the implementation of $f_{XOR_2} = x_1\overline{x_2} + \overline{x_1}x_2$ in Figure 2. Logic

*This work is part of a project that has received funding from the European Union's H2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement #691178, as well as supported by the TUBITAK-Career project #113E760.

NANOARCH '18, July 17–19, 2018, Athens, Greece

ACM ISBN 978-1-4503-5815-6/18/07…$15.00
https://doi.org/10.1145/3232195.3232211

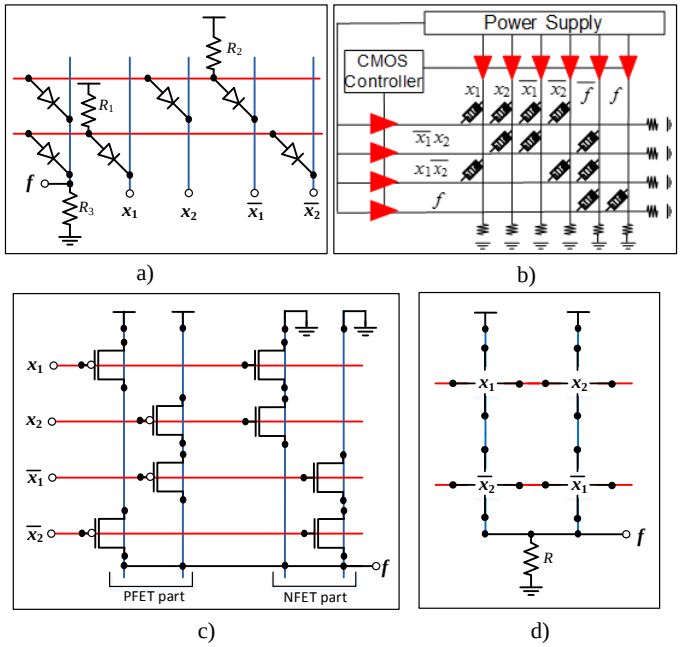

Figure 2: Implementation of f_{XOR_2} with different nano-crossbar types: crosspoint as a) diode, b) memristor, c) FET, and d) four-terminal switch.

synthesis models for diode and memristor based crossbars are very much PLA-like as can be seen in Figure 2(a) and 2(b). Memristor based crossbars have one major difference that logic computation is made through several states/loops (for further information, check [24]). For FET based crossbars, each product of the function or function's dual is realized by a separate column, as seen in Figure 2(c). Each inputs is assigned to a row to control the FETs on the corresponding row. Another type is a four-terminal based crossbar; here every crosspoint performs switching on all four directions. Crosspoints' control lines are not shown in Figure 2(d), yet detailed explanation of control lines can be found in [2].

Regarding emerging technologies and nano-fabrication, fault rates are much higher for nano-crossbars, as expected, compared to those of conventional CMOS circuits [8]. Therefore, during logic synthesis, consideration of faults and defects is mandatory. This applies for the integration of both diode, FET based or novel 4-terminal based logic synthesis methodologies. For this reason, researchers focus on challenges including defect and variance tolerances [21] [10]. Defect and variance tolerant approaches are closely related to logic realization and performance optimization, respectively.

Taking mentioned issues into account, we have developed a complete integration methodology for logic synthesis, defect tolerance and performance optimization with variance tolerance. This methodology is designed as a step-by-step guide to combine modular research approaches into an entire production pipeline. The overview of proposed integrated methodology is explained in Section 2. Following sections, we have demonstrated and reviewed the current methods with the exception of Section 4.2. Since four-terminal design parts from the rest in terms of defect tolerance, we present a preliminary and novel approach for defect tolerance of

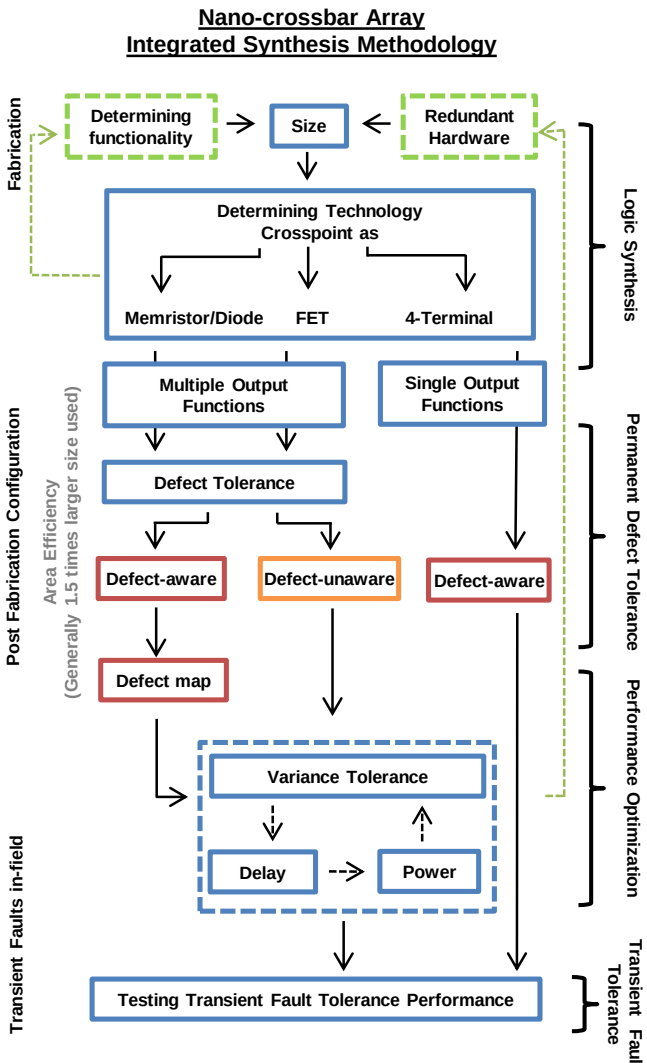

Figure 3: Integrated synthesis methodology scheme for nano-crossbar arrays.

four-terminal crossbars which is anticipated as an initial step in further research.

2 PROPOSED INTEGRATION METHODOLOGY

As briefly explained in the previous section, nano-fabrication delivers switching nano-crossbar arrays with structures or individual components having varied properties. Mentioned factors introduce random characteristics of which need to be carefully considered by synthesis process. For instance, a competent synthesis methodology must consider basic technology preference for switching elements, defect or fault rate of the given nano-crossbar and the variation values. Presented synthesis methodology in this study comprehensively covers the all specified factors and provides optimization algorithms for each step of the process. A schematic summary demonstrating every step of the method with annotation showing the certain research tasks is given in Figure 3.

First step of the synthesis process of a nano-crossbar involves the decision of switching technology which will be explained elaborately in Section 3. Main purpose is to determine which of the diode/memristor, FET, or four-terminal based components are to be used. This step is one of the most important procedures determining the size of the nano-crossbar. Production with diode/memristor based technologies as well as with FET are explained and then logic synthesis design with four-terminal based switches is given.

Second step of the synthesis process of a nano-crossbar covers the permanent faults (defects forming in the course of fabrication) and the tolerance aspects, which will be described in Section 4. Main purpose is to obtain a valid realization of a given logic function using two distinct approaches titled as defect-aware and defect-unaware. First method employs faults existing in nano-crossbar during the realization of logic function hence the name aware. Second method avoids the faults by attempting to find a fault-free region of nano-crossbar at the beginning so realization of given logic function is straightforward at the end.

Third step of the synthesis process of a nano-crossbar covers the variation minimization, which will be explained in Section 5. Main purpose is to minimize the overall delay by considering the individual variation values of components used in logic synthesis. Heuristic algorithms in the literature produce results very close to theoretical lower bound.

Delay minimization stage of the fourth step can be modified to include power optimization and fault tolerance as well. An objective function added to the variation tolerant algorithms guides the process to minimize both delay and power values of a nano-crossbar. Nevertheless, since this is a multi-level optimization, performance of the algorithm diminishes to an extent. In order to execute a parametric optimization, trade-offs must be decided considering technology dependent features and other conditions for the objective function. Furthermore, fault tolerance mechanism can be appended to the delay minimization process. It is possible to avoid faulty elements by assigning infinite variation values to them. Since the heuristic is based on minimizing the variation values of the nano-crossbar, fault avoidance is inherently established.

Final step of the synthesis process of a nano-crossbar involves the analysis of transient faults. Main purpose is to determine the effect of transient faults to the operational capacity of nano-crossbar and calculate fault tolerance performance.

3 LOGIC SYNTHESIS

At the beginning of logic synthesis process, crossbar technology must be determined based on certain fundamental criteria:

- Crossbar size limits (area) and Function size
- Fabrication Complexity
- Function output number (Multi or single output realization)
- Power and Delay Specifications
- Application specification (memory based etc.)

Decision should be made on the importance of the listed items; this could change depending on the application. For example, if an application has also memory unit, then it will be smart to choose memristor technology for logic unit. Since memristor could be used on memory unit as well, then they can interact more smoothly and the same fabrication technique could be used for both.

On the other hand, realization of a function with diode or memristor based crossbar requires less area than FET based option. However FET has better power performance over other technologies. In addition to all, four-terminal based crossbar performs better results on most of the function in terms of area [13].

In this section, we survey logic synthesis step of the integration methodology by considering only area size of the crossbar arrays. Array size formulations, which are given below, could be a guideline to this.

Array size formulations for single output function f (where f's dual is f^D):

- for **Diode**:$(number\ of\ products\ in\ f) + 1) \times ((number\ of\ literals\ in\ f) + 1)$
- for **Memristor**: $((number\ of\ products\ in\ f) + 2) \times ((number\ of\ literals\ in\ f) + 2)$
- for **FET**: $(number\ of\ literals\ in\ f) \times ((number\ of\ products\ in\ f) + (number\ of\ products\ in\ f^D))$
- for **4-terminal**: $(number\ of\ products\ in\ f^D) \times (number\ of\ products\ in\ f)$

For the single and multi output function realization, synthesis methodology for FET crossbar does not allow us to produce multi-level logic synthesis, only two-level approach can be used [20]. However, multi-level logic synthesis approach is applicable for diode and memristive crossbars [23]. Therefore, area optimization still demands further research for FET systems. Furthermore, as mentioned in Section 1, logic synthesis on diode and memristive crossbars is similar to PLA like synthesis. So the same approaches (which for the PLA) are applicable such as product sharing, phase changing etc.

Logic synthesis on four-terminal crossbars (lattices) is relatively a new method and technology. As shown in [2], Altun presented a useful logic synthesis technique for four-terminal crossbars (lattices). Yet mentioned method does not warrant the fact that produced lattice has optimal solution in terms of area. Therefore, new specific logic synthesis methodologies are needed to be presented. As shown in [9] and [13], optimal synthesis methodologies are provided. In addition, there are decomposition based techniques such as XOR based [12] [6], p-circuit [4] and dimension reducibility [5] decompositions as well.

4 DEFECT/FAULT TOLERANCE

In this section, we will investigate defects or faults with categorizing them as permanent (naming defects) and transient (naming faults). As mentioned in Section 1, crossbars tend to be fabricated with defects. Also, particular transient faults can occur in the field. Defect tolerance basically means finding defect-free region or crosspoint which can still be employed during logic synthesizing. On the other hand, faults can only be tolerated by redundancy, since they happens transiently. Yet sensitivity analysis can be made for both types. Defect model can be found in Figure 4 demonstrating stuck-at-0 (open) and stuck-at-1 (close). Their features can be summarized as:

- **Permanent Faults** occur mostly in fabrication and are tolerated in post-fabrication by redundancy and reconfigurability (mapping).

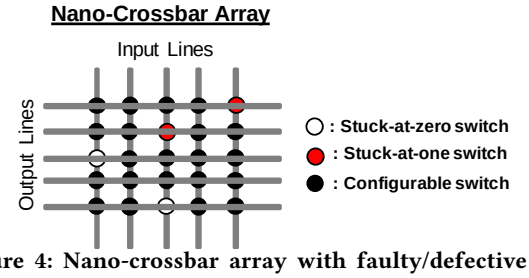

Figure 4: Nano-crossbar array with faulty/defective cross-points.

- **Transient Faults** occur in field and are tolerated in field by only redundancy

4.1 Defect Tolerance for Diode, Memristor and FET

Defect tolerance is achieved by realizing a target logic functions on a defective crossbar using row and column permutations. This problem is considered as NP-complete [17]. For the worst-case, $N!M!$ permutations are required to find a successful mapping for $N \times M$ crossbar. Algorithms in the literature use defect-unaware or defect-aware approach.

Defect-unaware algorithms aim to find the largest possible $k \times k$ defect-free sub-crossbar from a defective $N \times N$ crossbar where $k \leq N$ [27]. The algorithms are inefficient for high fault rates - obtained k values are much smaller than N [27]. In this regard, defect-aware algorithms perform much more satisfactorily [21]. We have performed detailed analysis of algorithms in [22].

Defect-aware considers the defect characteristics (stuck-at-0 or stuck-at-1), then decide which switch to employ during the mapping. In our previous work [21], we have proposed an efficient heuristic algorithms which aims to match defected crossbar and the function solution crossbar. For this, it defines crossbars as matrix. Therefore it can perform sorting, matching and backtracking steps efficiently. It makes repetition for a limit of permutation. This controls heuristic feature of the algorithm.

4.2 Defect Tolerance for Four-terminal

Four-terminal defect tolerance demands a different approach than the methods we have covered so far. For this reason, we present a novel method, which is firstly introduced in this paper. The Method utilizes a prior sensitivity analysis of crossbar to specify critical switches, and strengthens them with proposed mitigation factors. The same naming conventions are applicable, regarding defects which are categorized as stuck-at-0 (SA0) and stuck-at-1 (SA1). In addition, we follow the same terminology adopted in [2] and [9] by addressing crossbar as lattice and switch as cell to be consistent and emphasize the distinction of four-terminal approach. Finally, it should be noted that as opposed the previous sections, we provide a more detailed explanation due to original technical contribution presented in this section.

4.2.1 Defect Injection Methodology. We perform a defect injection with uniform distribution to lattice reaching defect densities up to 10%. Every cell (a four-terminal switch) is presumed to have only SA0 or SA1. Once the "defective" lattice is obtained, the algorithm

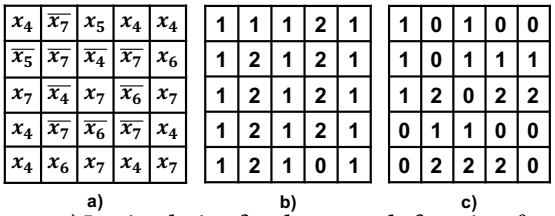

Figure 5: a) Lattice design for the example function f and its sensitivity map for b) SAO and c) SA1.

generates all the possible 2^n inputs (where n is the number of variables). For each input, the simulation algorithm compares the given output with the correct one. Let E^0_{ij} (resp., E^1_{ij}), with $1 \leq i \leq r$, $1 \leq j \leq s$, be the number of defective outputs with a SA0 (resp., SA1) in the cell (i, j) of the given lattice. Note that $0 \leq \{E^0_{ij},\ E^1_{ij}\} \leq 2^n$. Moreover, when E^0_{ij} (resp., E^1_{ij}) is equal to 0 we have that, for any possible input, the lattice output is never changed by the SAF in the cell (i, j). In this case, we call the cell (i,j) robust w.r.t. SA0 (resp., SA1). Let R^0 (resp., R^1) be the total number of robust cells w.r.t. SA0 (resp., SA1) in the lattice. Finally, let $E^0 = \sum_{i=1}^{i=r} \sum_{j=1}^{j=s} E^0_{ij}$ (resp., $E^1 = \sum_{i=1}^{i=r} \sum_{j=1}^{j=s} E^1_{ij}$) be the total number of defective output with SA0 (resp. SA1) in the simulation. For an example of function $f = x_4\overline{x_5}x_7 + \overline{x_4}x_6\overline{x_7} + \overline{x_4}x_5\overline{x_6}x_7 + x_4\overline{x_6}\overline{x_7} + x_4x_6x_7$ realized in Figure 5(a) (with the method in [2]), in the Figure 5(b) (resp., 5(c) shows the map containing E^0_{ij} (resp., E^1_{ij}) in each cell.

4.2.2 Metrics used for Sensitivity Analysis. In order to evaluate the sensitivity of a lattice to SA0 and SA1 defects, we propose two metrics. The first one measures the average number of defective outputs considering sensitive cells to SA0 or SA1 only. The second one measures the average number of defective outputs in the entire lattice. Note that the total number of cells is the area of the lattice (i.e., $r \times s$), the number of non-robust cells for SA0 (resp., SA1) is $r \times s - R^0$ (resp., $r \times s - R^1$), and 2^n is the total number of inputs. (1) Sensitivity of defective cells is the total number of inputs that give an uncorrected output (E^0 and E^1) divided by the total number of inputs (2^n), for each non-robust cell ($r \times s - R^0$ or $r \times s - R^1$). In the case of SA0 the metric can be expressed as: $S^0_C = E^0/(2^n(r \times s - R^0))$. The same reasoning can be done for SA1 defect sensitivity. (2) Sensitivity of lattice is the total number of inputs that give an uncorrected output divided by the total number of inputs for each cell, in the case of SA0 is: $S^0_L = E^0/(2^n(r \times s))$. The SA1 case is analogous.

4.2.3 Benchmarks and Simulations. The defect simulations have been run on a machine with two AMD Opteron 4274HE for a total of 16 CPUs at 2.5 GHz and 128 GByte of main memory, running Linux CentOS 7. The benchmarks functions are expressed in PLA form and are taken from a subset of LGSynth93 [26]. A total of about 580 functions were considered, and each output of a function is implemented as a separate Boolean function.

The software used for simulations is written in C++. We used ESPRESSO to implement the method described in [2], and a collection of Python scripts for computing minimum-area lattices by transformation to a series of SAT problems, to simulate the results reported in [9]. Each SAT execution is stopped after ten minutes.

Table 1: A sample of benchmark functions synthesized with [2] and [9] approaches and their sensitivity values

name	$r \times s$	n	E^0	S_C^0	S_L^0	$\%\frac{R^0}{r\times s}$	E^1	S_C^1	S_L^1	$\%\frac{R^1}{r\times s}$
Synthesis with Dual Method [2]										
add6(1)	6×6	4	19	0.06	0.03	47%	9	0.06	0.02	75%
alu2(2)	11×10	8	462	0.03	0.02	35%	121	0.02	0.01	80%
b11(1)	3×6	7	28	0.02	0.01	44%	73	0.03	0.03	6%
dc2(0)	4×6	7	117	0.05	0.04	17%	162	0.08	0.05	33%
exam(5)	6×11	9	1868	0.07	0.06	17%	131	0.02	0.01	74%
z4(2)	12×12	5	70	0.03	0.02	51%	14	0.03	0	90%
Synthesis with Quantified Boolean Logic [9]										
add6_G_1	5×3	4	31	0.15	0.13	13%	32	0.14	0.13	7%
alu2_G_2	7×3	8	464	0.1	0.09	14%	384	0.08	0.07	5%
b11_G_1	3×5	7	45	0.03	0.02	7%	128	0.07	0.07	7%
dc2_G_0	4×4	7	104	0.06	0.05	13%	132	0.07	0.06	13%

Table 2: Overall results of the simulations

Synthesis Method	Average area	Average n	S_C^0	S_L^0	$\%\frac{R^0}{r\times s}$	S_C^1	S_L^1	$\%\frac{R^1}{r\times s}$
[2]	30	6	0.05	0.05	20%	0.06	0.05	29%
[9]	15	7	0.07	0.06	9%	0.07	0.07	8%

In Table 1, we report a sample of benchmark functions and their sensitivity values, according to the metrics presented before. In particular, Table 1 refers to lattice synthesized as described in [2] and [9]. The benchmarks that are present in Table 1 with dual method were stopped after ten minutes of SAT execution, but that was not the case for the rest.

More precisely, in both methods, the first column reports the name and the number of the considered output of each function. The following columns report dimension ($r \times s$) required for the synthesis of a given function according to each decomposition method, and the number of input variables n. Columns from 4 to 7 refers to SA0 defect metrics (resp., columns from 8 to 11 to SA1 metrics) showing the total number of errors E^0, the Sensitivity of defective cells S_C^0, the Sensitivity of lattice S_L^0 and the percentage of robust cells $\%R^0/r \times s$.

Table 2 describes the overall results for the benchmarks we have considered. It also shows the average values for the considered metrics. We can note that the percentage of cells that are considered robust according to our metrics is higher in the first approach [2]. This is due to the more constrained structure of the lattices produced by the first synthesis method. Indeed, the method proposed in [2] computes a lattice for f and its dual that is in general less compact than the lattice given by [9] (see, the column Average area in Table 2). Moreover, we can note that the sensitivity of the lattice to stuck-at-defects (SAD) is quite low for both methods. In fact, the experiments show that, in general, non-robust cells -in presence of a SAD- compute a defective output for a very limited number of inputs.

4.2.4 Mitigation by Defect Avoidance. From the above results, it can be seen that the two analyzed mapping algorithm shows different sensitivities of the output of a given function. As a matter of fact, the more restrictive an algorithm is in terms of area (results closer to optimal solution), the higher the defect sensitivity of the output to cell defect of SA0 or SA1. It is mandatory to include in the mapping algorithm defect-avoidance heuristics.

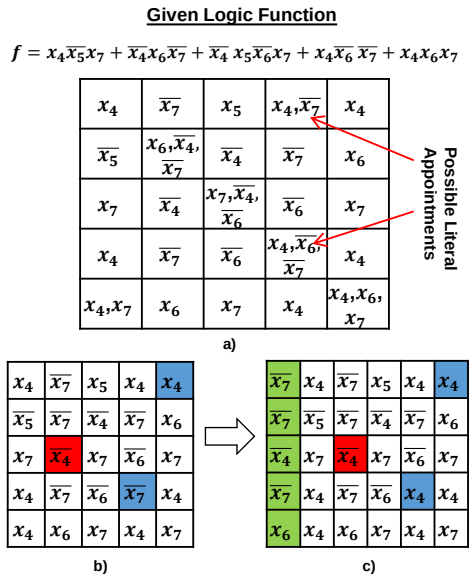

Figure 6: a) defect-free lattice; b) lattice with defects: SA0 in red and SA1 in blue; and c) lattice with the defect fixed.

In order to mitigate the sensitivity of a lattice to SAD, we propose the following possible strategy applied to the synthesis method proposed in [2] which has been proven as less sensitive to SAD impact on the output functions: (1) For a given mapped function, if a potential SA0, SA1 defect affects a robust cell identified by the defect injection campaign, the lattice still computes the correct output, thus we do not need any mitigation with defect tolerant design. (2) However, if an injected defect occurs in a multiple-choice cell, if a different literal can be chosen to make the cell robust, we change the literal with the new one. (3) Otherwise, if the injected SA0 defect is proven as being critical for the output value, the column that contains that defective cell has to be replaced by spare columns. In case of an SA1 the row that contains the defective cell has to be replaced by a spare row. Note that, in this case, the output still provides a correct function f from top to bottom, but the function from left to right could be changed and become a function which will not be dual of f anymore.

As an example, consider the lattice synthesized in Figure 6(a) with $f = x_4\overline{x_5}x_7 + \overline{x_4}x_6\overline{x_7} + \overline{x_4}x_5\overline{x_6}x_7 + x_4\overline{x_6x_7} + x_4x_6x_7$ by using synthesis method presented in [2]. The example shows one case of mitigation of 3 independent SAD affecting the crossbar implementing the function, yielding an approximative 10% defects. In Figure 6, SA1 cells are marked in blue and SA0 cells a remarked in red.

4.3 Transient Fault Tolerance

Regarding transient faults, there are two approaches: redundancy based and manipulation of logic function to obtain a more fault tolerant design. Nevertheless, it should be noted that since nano-crossbars are in the early stage of development, there is a lack of in-field data regarding transient faults. For this reason, second approach towards transient fault tolerance is methodology independent and rather focuses on the intrinsic features of the given logic functions.

As mentioned, transient faults can be tolerated with redundancy. Inserting redundant components can be constructed with adding extra rows and/or columns as shown [16] [3].

Furthermore, provided certain conditions, particular logic functions are inherently tolerant to transient faults limited to certain switches of crossbar as show in [21].

Mentioned inherent tolerance capability varies depending on the design. For example, consider a design having low logic inclusion ratio (IR), meaning less number of crosspoints of a crossbar are used, that means this design is more tolerant to stuck-at-0 faults. Logic synthesis (design) should be made with regard to this, since there are multiple design solution.

Similarly, if the target function can be realized with high IR, then a technology, the one which tends to have stuck-at-1 faults, should be preferred. for example function $f = x_1x_2 + x_2x_3 + \overline{x_3}x_4$ can also be written as $f' = x_1x_2\overline{x_3} + x_2x_3 + \overline{x_3}x_4 = f$, therefore IR can be increased.

Fault sensitivity analysis can be made using Monte Carlo analysis, yet it is costly. Since we know the dynamics of the fault tolerance we can calculate sensitivity (fault tolerance performance) directly with algebraic equations. Equation parameters consists of crossbar dimensions, inclusion ratio, fault occurrence possibility and number of tolerable crosspoint. For further information please refer to [21].

5 VARIANCE TOLERANCE AND PERFORMANCE OPTIMIZATION

Another aspect of the crossbar fabrication is that every crosspoint does not have the same property in terms of dimension, doping, etc. [14]. It is called variance of crosspoints. This affects threshold voltages and ON and OFF resistances as well as capacitance values. It means that delay and power performances are changing. Decision of which crosspoint switches are going to be used during logic design plays a crucial role in performance optimization. To achieve variation tolerant delay values, different optimization algorithms have been proposed [28] [15]. These algorithms aim to optimize the worst-case delay values in logic mapping. They use Gaussian distribution to model variances.

On the other hand, the literature lacks variation tolerant power optimization algorithms. Considering that fault tolerance mechanism can be appended to the delay minimization process, variation-power-delay optimizations are needed. This can be considered as a future direction. Another future direction is performing sensitivity analysis for switching components of the arrays.

6 CONCLUSION AND DISCUSSION

In this study, we present a synthesis methodology for crossbar arrays having crosspoints working as FET, diode/resistive/memristive, or four-terminal switch based devices. We cover "logic synthesis", "defect/fault tolerance", and "variation-area-power-delay performance optimization" steps. Presented synthesis methodology provides optimization algorithms for each step of the process as well as their relations and trade-offs.

REFERENCES

[1] Dan Alexandrescu, Mustafa Altun, Lorena Anghel, Anna Bernasconi, Valentina Ciriani, Luca Frontini, and Mehdi Tahoori. 2016. Synthesis and Performance Optimization of a Switching Nano-Crossbar Computer. In *Digital System Design (DSD), 2016 Euromicro Conference on*. IEEE, 334–341.

[2] Mustafa Altun and Marc D Riedel. 2012. Logic synthesis for switching lattices. *IEEE Trans. Comput.* 61, 11 (2012), 1588–1600.

[3] Samary Baranov, Ilya Levin, Osnat Keren, and M Karpovsky. 2009. Designing fault tolerant FSM by nano-PLA. In *On-Line Testing Symposium, 2009. IOLTS 2009. 15th IEEE International*. IEEE, 229–234.

[4] Anna Bernasconi, Valentina Ciriani, Luca Frontini, Valentino Liberali, Gabriella Trucco, and Tiziano Villa. 2016. Logic Synthesis for Switching Lattices by Decomposition with P-Circuits. In *Digital System Design (DSD), 2016 Euromicro Conference on*. IEEE, 423–430.

[5] Anna Bernasconi, Valentina Ciriani, Luca Frontini, and Gabriella Trucco. 2016. Synthesis on switching lattices of Dimension-reducible Boolean functions. In *Very Large Scale Integration (VLSI-SoC), 2016 IFIP/IEEE International Conference on*. IEEE, 1–6.

[6] Anna Bernasconi, Valentina Ciriani, Luca Frontini, and Gabriella Trucco. 2017. Composition of Switching Lattices and Autosymmetric Boolean Function Synthesis. In *Digital System Design (DSD), 2017 Euromicro Conference on*. IEEE, 137–144.

[7] Yong Chen, Gun-Young Jung, Douglas AA Ohlberg, Xuema Li, Duncan R Stewart, Jan O Jeppesen, Kent A Nielsen, J Fraser Stoddart, and R Stanley Williams. 2003. Nanoscale molecular-switch crossbar circuits. *Nanotechnology* 14, 4 (2003), 462.

[8] Andre DeHon and Benjamin Gojman. 2011. Crystals and snowflakes: building computation from nanowire crossbars. *Computer* 2 (2011), 37–45.

[9] Graeme Gange, Harald Søndergaard, and Peter J Stuckey. 2014. Synthesizing optimal switching lattices. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 20, 1 (2014), 6.

[10] Benjamin Gojman and André DeHon. 2009. VMATCH: Using logical variation to counteract physical variation in bottom-up, nanoscale systems. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE, 78–87.

[11] Yu Huang, Xiangfeng Duan, Yi Cui, Lincoln J Lauhon, Kyoung-Ha Kim, and Charles M Lieber. 2001. Logic gates and computation from assembled nanowire building blocks. *Science* 294, 5545 (2001), 1313–1317.

[12] Muhammed Ceylan Morgül and Mustafa Altun. 2014. Anahtarlamalı Nano Dizinler ile Lojik Devre Tasarımı ve Boyut Optimizasyonu Logic Circuit Design with Switching Nano Arrays and Area Optimization. In *ELECO*.

[13] Muhammed Ceylan Morgul and Mustafa Altun. 2015. Synthesis and optimization of switching nanoarrays. In *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2015 IEEE 18th International Symposium on*. IEEE, 161–164.

[14] Muhammed Ceylan Morgul, Furkan Peker, and Mustafa Altun. 2016. Power-Delay-Area Performance Modeling and Analysis for Nano-Crossbar Arrays. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE, 437–442.

[15] Furkan Peker and Mustafa Altun. 2018. A Fast Hill Climbing Algorithm for Defect and Variation Tolerant Logic Mapping of Nano-Crossbar Arrays. *IEEE Transactions on Multi-Scale Computing Systems* (2018), 1–1.

[16] Wenjing Rao, Alex Orailoglu, and Ramesh Karri. 2007. Logic level fault tolerance approaches targeting nanoelectronics plas. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*. IEEE, 1–5.

[17] Anish Man Singh Shrestha, Satoshi Tayu, and Shuichi Ueno. 2009. Orthogonal Ray Graphs and Nano-PLA Design.. In *ISCAS*. 2930–2933.

[18] G Snider. 2005. Computing with hysteretic resistor crossbars. *Applied Physics A: Materials Science & Processing* 80, 6 (2005), 1165–1172.

[19] Greg Snider, P Kuekes, T Hogg, and R Stanley Williams. 2005. Nanoelectronic architectures. *Applied Physics A* 80, 6 (2005), 1183–1195.

[20] Greg Snider, Philip Kuekes, and R Stanley Williams. 2004. CMOS-like logic in defective, nanoscale crossbars. *Nanotechnology* 15, 8 (2004), 881.

[21] Onur Tunali and Mustafa Altun. 2017. Permanent and transient fault tolerance for reconfigurable nano-crossbar arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 5 (2017), 747–760.

[22] Onur Tunali and Mustafa Altun. 2017. A Survey of Fault-Tolerance Algorithms for Reconfigurable Nano-Crossbar Arrays. *ACM Comput. Surv.* 50, 6, Article 79 (Nov. 2017), 35 pages.

[23] Onur Tunali and Mustafa Altun. 2018. Logic Synthesis and Defect Tolerance for Memristive Crossbar Arrays. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2018*.

[24] Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, and Koen Bertels. 2015. Fast boolean logic mapped on memristor crossbar. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 335–342.

[25] Hao Yan, Hwan Sung Choe, SungWoo Nam, Yongjie Hu, Shamik Das, James F Klemic, James C Ellenbogen, and Charles M Lieber. 2011. Programmable nanowire circuits for nanoprocessors. *Nature* 470, 7333 (2011), 240–244.

[26] Saeyang Yang. 1991. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC).

[27] Bo Yuan and Bin Li. 2014. A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 10, 3 (2014), 25.

[28] Bo Yuan, Bin Li, Huanhuan Chen, and Xin Yao. 2016. Defect-and Variation-Tolerant Logic Mapping in Nanocrossbar Using Bipartite Matching and Memetic Algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 9 (2016), 2813–2826.