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

M. Ceylan Morgul*, Luca Frontini, Onur Tunali, Lorena Anghel, Valentina Ciriani, E. Ioana Vatajelu, 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 with the drawback of higher fault rates. They offer area and power efficiency in terms of their easy-to-fabricate and dense physical structures. They consist of regularly placed crosspoints as computing elements, which behave as diode, memristor, field effect transistor, or novel four-terminal switching devices. In this study, we establish a complete design framework for crossbar circuits explaining and analyzing every step of the process. We comparatively elaborate on these technologies in the sense of their capabilities for computation regarding area including a new logic synthesis technique for memristors, fault tolerance including a novel paradigm for four-terminal devices, delay, and power consumption. As a result, this study introduces a synthesis methodology that considers basic technology preference for switching crosspoints and fault rates of the given crossbar 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 arrays have emerged to be an alternative/complementary technology to CMOS [46]. In their fabrication, relatively cheap bottom-up nano-fabrication techniques are used rather than pure lithography based conventional production. Because of the novel manufacturing techniques, end products have regular and dense forms with area and power efficient structures [15] [4].

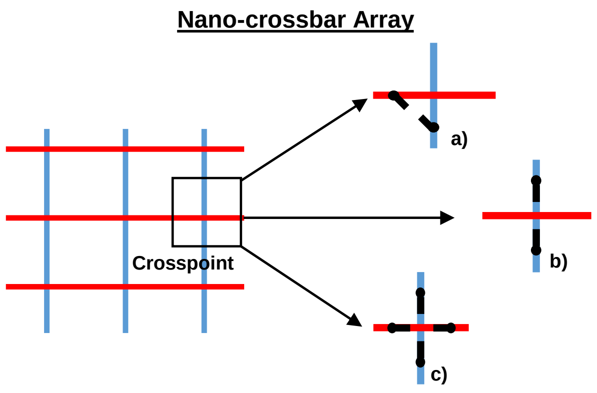


Fig. 1. Switching models of a nano-crossbar array: crosspoint as a) two-terminal switch with terminals in the crossed lines, b) two-terminal switch with terminals in the same line, and c) four-terminal switch.

Main principle behind crossbar based computing is to utilize crosspoints to behave like switches, either as two-terminal or novel four-terminal [6]. This is illustrated in Figure 1. Depending on the used technology, a two-terminal switch behaves either as a diode [18], a resistive/memristive switch [33] [20], or a field effect transistor (FET) [34]. 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). Four terminal architecture has either all of its terminals connected or disconnected. The desired state is actualized with a controlling input, which is not present in the Figure 1(c) and has a separate physical formation from the crossbar which is thoroughly explained for different technologies in [6] [30]. Detailed TCAD simulations and technology development are presented in [30]. In addition, a realization with standard CMOS process is demonstrated in [13].

Contrary to the conventional technologies, circuit design

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

Morgul and Mircea R. Stan are with the Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, Virginia, USA e-mail: {mm4uz, mircea}@virginia.edu

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

*: The work is primarily done when he was with Istanbul Technical University; he is currently affiliated with University of Virginia

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 and TUBITAK-2501 project #218E068.

steps of nano-crossbar arrays are not fully incorporated due to their emerging nature. Motivated by this, we expand and update our preliminary integrated synthesis methodology in [24], and present it in length. Main steps of the methodology are logic synthesis, defect/fault tolerance and performance optimization. In Section II, overview of the methodology is stated with the background information. Details of the design steps are stated in the following sections III IV V. We present a case study to elaborate the methodology conclusively in section VI. For sake of clarity, the significant experimental results are shown in the related sections. Main contributions are as follows:

- Expanded version of integrated synthesis methodology presented in [24].
- Multi-output logic synthesis for four-terminal lattices 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.

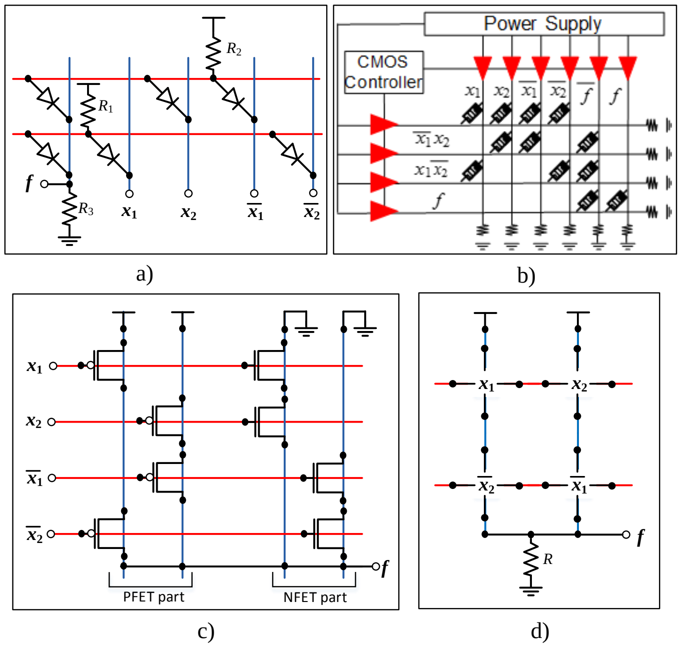


Fig. 2. Realization of f_{XOR_2} with different nano-crossbar types: crosspoint as a) diode, b) memristor, c) FET, and d) four-terminal switch.

II. Overview of Circuit Design Steps

A. Background

Nano-crossbar arrays are first shown to be realizable conceptually in [33], [36] and then physically in [46], [48] as an integrated circuit. After that, research mostly follows the order of technology demonstration such as logic synthesis with ideal (non-defective) arrays, logic synthesis with defective arrays, performance-aware design (performance analysis/optimization), and technology development. This study targets to integrate the current researches with completing missing parts.

1) Logic Synthesis: Main goal of logic synthesis process is to optimize the area size of the circuit through formalizing the circuit size specific to underlying technology. To illustrate different approaches, we show examples for the realization 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 quite 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 refer to [44]). We chose the approach in [44] for the memristive crossbars, because the proposed architecture design does not lose its functionality due to the sneak path issues [20]. 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 and connection between top and bottom yields 1 as output and 0 otherwise. Control lines of crosspoints are not shown in Figure 2(d), and detailed explanation of control lines can be found in [6] [30].

2) Defect/Fault Tolerance: Anticipated fault rates are much higher for nano-crossbars, as expected, compared to those of conventional CMOS circuits [35]. Therefore, during logic mapping, consideration of faults is mandatory. This applies to the all technology types such as diode, FET based or novel four-terminal based arrays. In short, contrasting the conventional CMOS approach, certain switches cannot be used in mapping process, therefore mapping procedure becomes an assignment problem. Early attempts to overcome this issue consists of locating a fault-free region so no crossbar specific assignment is necessary as in [37]. However, it has quickly become apparent that fault free region is not satisfactory in terms of area size [41]. For this reason, researchers focus on challenges including defect and variance tolerances [40] [28]. In this study, we apply a new fault tolerance technique for four-terminal crossbars (lattices).

3) Performance Optimization: With the process variability data (as an extended concept of defects) of crossbars, performance-aware design can be accomplished. Given that targeted technologies have different performance characteristics, to perform fair comparison, their dependencies on target function should be analyzed. Previously, we have analyzed and extracted delay and power characteristic with dependencies on given function for memristor crossbars [43]. In this study, we present delay and power model for diode, FET, and four-terminal lattice. These models characterize the performance of crossbar technologies for given target function. Idea is to optimize a crossbar with given design specification. However, with the lack of experimental results, optimization and comparison studies are limited with modeled characteristics.

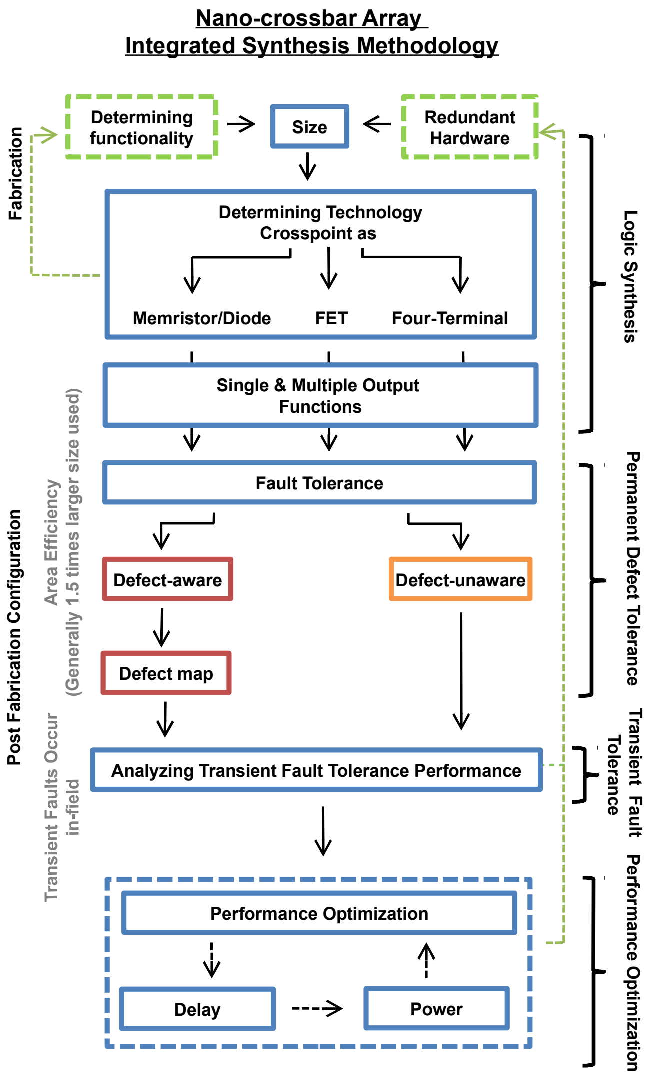

Nano-crossbar Array
Integrated Synthesis Methodology
Fabrication
Determining functionality
Size
Redundant Hardware
Determining Technology Crosspoint as
Memristor/Diode
FET
Four-Terminal
Single & Multiple Output Functions
Fault Tolerance
Defect-aware
Defect-unaware
Defect map
Analyzing Transient Fault Tolerance Performance
Performance Optimization
Delay
Power
Post Fabrication Configuration
Area Efficiency
(Generally 1.5 times larger size used)
Transient Faults Occur in-field
Logic Synthesis
Permanent Defect Tolerance
Transient Fault Tolerance
Performance Optimization

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

B. Overview of Integrated Methodology

As briefly stated, nano-fabrication produces switching nano-crossbar arrays with varying properties, structurally and/or component-wise. These random characteristics need to be considered carefully by the synthesis process. For example, a competent methodology must regard 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. The diagram, given in Figure 3, summarizes the methodology with demonstrating every step including annotated research tasks.

The first step (logic synthesis) covers the decision of switching technology and generate logic function description in crossbar form. The main goal 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 area size of the nano-crossbar that is the chief optimization metric. Logic synthesis with diode/memristor, FET, or four-terminal switching technology is given in section III.

The second step (fault tolerance) covers the permanent faults (defects forming in the course of fabrication) and the transient faults (which occur in-field). The main goal is to obtain a valid mapping of a given logic function in crossbar form, produced by logic synthesis step, and defect map. There are two distinct approaches titled as $defect-aware$ and $defect-unaware$. The first approach employs faults existing in nano-crossbar during the mapping ofthe function, hence the name $aware$. The second approach avoids the defects by attempting to find a defect-free region on the nano-crossbar at the beginning, so that the mapping of the logic function is straightforward. Finally, transient fault analysis is performed with respect to fault rates/types and fed back to logic synthesis step. Detailed explanations are given in Section IV.

The third step (performance optimization) covers final delay and power issues. The main goal is to analyze delay and power consumption of arrays by showing their dependencies on the properties of a given logic function as well as specifics of the used technology. Detailed explanations are given in Section V.

To demonstrate the whole process in work, a case study is provided to elucidate the proposed synthesis mechanism in Section VI.

III. Logic Synthesis

At the start, crossbar (or array used interchangeably) switching technology, *i.e.*, diode, memristor, FET, or four-terminal, needs to be chosen 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 requirements

A decision can be made on the importance and priority of the listed items, depending on the preference. For example, if an application demands in-memory computing, then the memristor technology can be chosen for the realization of logic functions, since the memristor can also be used as a memory unit.

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 of the FET based crossbars. However, the FET based designs consume less power than the diode/memristor-based design. Moreover, the four-terminal based crossbar includes less number of crosspoints than other crossbar designs [26].

Initially, we examine the logic synthesis techniques developed for diode, memristor, and FET based crossbars in the literature and then we formulate their crossbar sizes required to implement given logic 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 crossbars and compare them with other techniques in the literature.

A. Area Comparisons for Different Crossbar Technologies

We present the Logic Synthesis step of the integrated methodology, considering only the number of crosspoints in the crossbar. The size of a crossbar array including diode, memristor, FET, and four-terminal is given as follows:

- **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]**

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 $1 \leq i \leq n$.

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 [35]. However, multi-level logic synthesis approach is feasible for the diode and memristive crossbars [42]. Therefore, the optimization of 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 [6], 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 [17] and [26], optimal synthesis methodologies are provided. Moreover, decomposition based approximate techniques are presented such as XOR based [25] [9], p-circuit [10], and dimension reducibility [11] decompositions. Furthermore, a very efficient (20x faster) technique for very large functions is recently developed to make logic synthesis of four-terminal nano-crossbar arrays more feasible [3].

However, all of these studies focus on the realization of a single logic function using switching lattices. On the other hand, in [2] [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 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 [2] [1] (JANUS), 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 [2] [1], following the first two steps described above.

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

Benchmarks	Diode	Memristor	FET	Four-Terminal [2] [1]
squar5	594	884	900	**108**
rd53	442	560	819	120
misex1	437	600	1334	**126**
sqrt8	840	1032	1340	165
b12	2028	2544	3003	**180**
inc	897	1280	2231	235
bw	1900	3300	1824	**405**
sao2	1488	1764	3672	476
rd73	2210	2620	4658	606
clip	2875	3528	6256	685
rd84	5180	6240	10960	2320
ex5p	10823	19596	32864	2664
ex1010	8820	11800	57960	5958
apex4	16835	25480	72335	7308

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.

For the logic synthesis of memristor based crossbars, there exists more area efficient techniques than in [44]; such as [8] (SIMPLER MAGIC), [38], or [31] (which are essentially improved versions of the MAGIC [21]). For example, $misex1$ can be realized in an array which has the size of 33, 88 and 52 with [8], [38] and [31], respectively. However they require much larger number of operation cycles: 83, 52 and 31 with [8], [38] and [31], respectively. Note that, required number of cycles is only 7 for [44]. This number can even be reduced to 5 with the proposed technique [43]. On the other hand, as stated in [45], the method in [44] can support the method MAGIC. Furthermore, since the method of MAGIC (and improved versions of it) uses all crosspoints in the array (density is 100%), its defect intolerance is much higher than [44]. Therefore, we used the approach in [44] and implemented the improved version to our integrated synthesis methodology.

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

In memristive crossbar arrays, function outputs and their negations are produced [44]. However production order changes the array size. We have called $phase\_combination$-0 and $phase\_combination$-1 realization, if realization is happened based on an output itself and its negation, respectively. For instance, for the realization of phase_combination-1, first its negation is produced [43] and at last original function outputs. Based on this property, array size of mem-

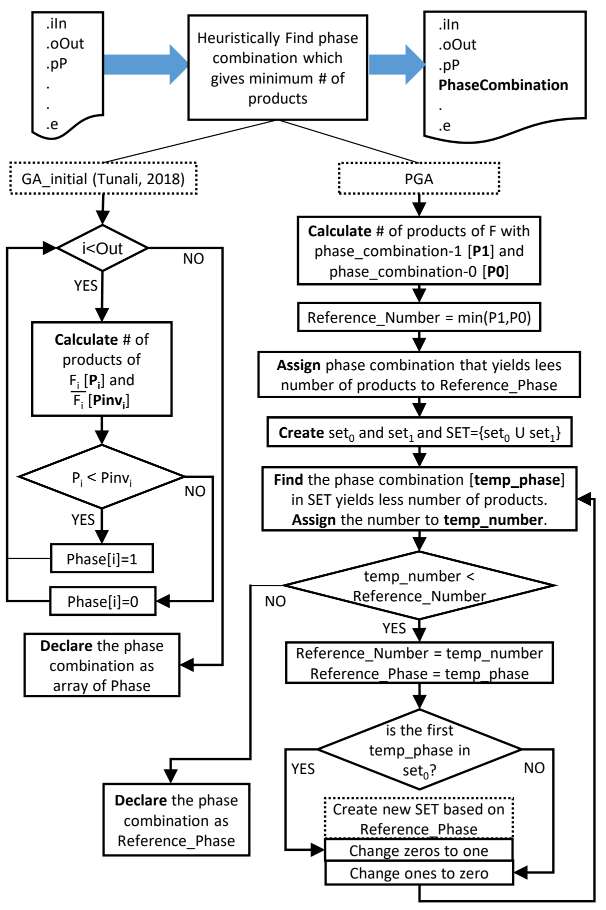

.iIn
.oOut
.pP
.
.
.e
Heuristically Find phase combination which gives minimum # of products
.iIn
.oOut
.pP
PhaseCombination
.
.e
GA_initial (Tunali, 2018)
PGA
i<Out
NO
YES
Calculate # of products of F_i [P_i] and F̄_i [Pinv_i]
P_i < Pinv_i
NO
YES
Phase[i]=1
Phase[i]=0
Declare the phase combination as array of Phase
Calculate # of products of F with phase_combination-1 [P1] and phase_combination-0 [P0]
Reference_Number = min(P1,P0)
Assign phase combination that yields lees number of products to Reference_Phase
Create set0 and set1 and SET={set0 U set1}
Find the phase combination [temp_phase] in SET yields less number of products. Assign the number to temp_number.
temp_number < Reference_Number
NO
YES
Reference_Number = temp_number
Reference_Phase = temp_phase
is the first temp_phase in set0?
YES
NO
Create new SET based on Reference_Phase
Change zeros to one
Change ones to zero
Declare the phase combination as Reference_Phase

Fig. 4. Block diagrams of two-level multi output logic synthesis algorithms: initial Greedy Algorithm (GA-initial) [43] and Proposed Greedy Algorithm (PGA)

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 [44] APPROACHES FOR MEMRISTIVE ARRAYS

Benchmarks	Basic [44]	GA-initial [43]	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

ristive crossbars can be optimized. The realization in [44] is phase_combination-1 realization.

In our previous study [43], we have proposed a greedy algorithm, which only considers outputs individually, and doesn't consider outputs interrelation (i.e. no product sharing at the analysis). We can call this algorithm as "initial Greedy Algorithm (GA-initial)". In this algorithm, only product numbers of individual outputs are considered. On the other hand our new greedy algorithm considers product sharing with the change of phase, meaning it considers total number of products of whole (multi-output) function. We illustrate these two algorithms in Figure 4 side-by-side.

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 (PGA) is as follows:

1: **Calculate and Compare** number of products of $phase\_combination$-0 (00..0) and $phase\_combination$-1 (11..1), **Assign** minimum number of products to $Reference\_Number$, and the phase combination to $Reference\_Phase$,
2: **Create** two sets of candidate phase combinations; set_0: includes phase combinations that only one output is in "phase-1" and set_1: includes phase combinations that only one output is in "phase-0", and **Assign** $SET = set_0 \cup set_1$,
3: **Find** the phase combinations ($temp\_phase$) which yields minimum number of products ($temp\_number$) for the function in SET.
4: **Decide**; if $temp\_number$ is less than $Reference\_Number$. If YES, **Assign** $temp\_number$ to $Reference\_Number$, and ($temp\_phase$) to $Reference\_Phase$; If NO, **Jump** to 8 (end of the algorithm),
5: **Decide**; If the found very first $temp\_phase$ is in set_0,
5a: If YES (in set_0), **Create** a new SET, with phase combinations, by changing "0"s of $Reference\_Phase$ to "1"s,
5b: If NO (in set_1), **Create** a new SET, with phase combinations, by changing "1"s of $Reference\_Phase$ to "0"s,
(Note that, every phase combination in SET has only one alteration with $Reference\_Phase$)
6: **Jump** to 3,
7: **Declare** 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; set_0: "001, 010, and 100" (changing zeros to one), and set_1: "110, 101, and 011" (Changing ones to zero) (Note that, we change one at a time). Let's say we compared product numbers and found that phase "110" yields the minimum number of products. Then we continue from the phase "110", with changing ones to zero. Means, we check "100 and 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. (Notice that, for a function, which has only three outputs, looking to sets of

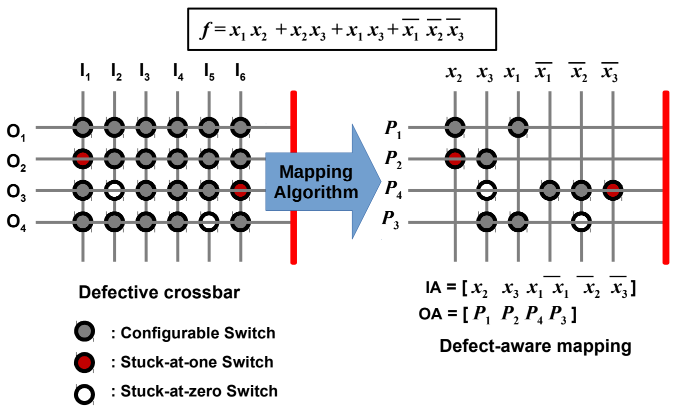

$f = x_1x_2 + x_2x_3 + x_1x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$
I1 I2 I3 I4 I5 I6
O1 O2 O3 O4
Mapping Algorithm
P1 P2 P4 P3
Defective crossbar
IA = [x_2 x_3 x_1 $\overline{x_1}$ $\overline{x_2}$ $\overline{x_3}$]
OA = [P_1 P_2 P_4 P_3]
Defect-aware mapping
: Configurable Switch
: Stuck-at-one Switch
: Stuck-at-zero Switch

Fig. 5. Nano-crossbar array with faulty/defective crosspoints and mapping process.

set_0 and set_1 is enough)

To evaluate the algorithms and compare the results, we use espresso and MATLAB^{TM} on a 3.20 GHz Intel Core i7 CPU with 4GB memory. Results are shown in Table II. Algorithm performances differs from function to function. For six of the examples PGA results less area size than GA-initial. Overall, PGA surpasses GA-initial, yet for two (clip and sao2) of the functions GA-initial results less area size than PGA. For the rest they produce the same result. GA-initial's major advantages is that it has almost no time cost. It can be used for any case.

IV. Defect/Fault Tolerance

In this section, we investigate the effects of permanent and transient faults on the behavior of a logic function mapped on a given crossbar array. We categorize faults into 2 classes: permanent faults generated by physical defects and transient faults. As mentioned in Section I, actual nano-crossbar arrays being fabricated with emerging technologies, are affected mostly by defects, due to the poor technology yields. Indeed, immature process technologies used for nanowires, memristive, etc. elements generate high defect density crossbars. At the same time, these elements also show instabilities and variabilities (behaving like transient or intermittent faults) and they are also affected by transient phenomena occurring in the field.

Permanent fault (defect) tolerance, also called defect avoidance, basically means finding defect-free regions or defect free crosspoints that can still be employed during logic mapping, and it is usually done by the logic function remapping (at the algorithm level) and/or by reconfiguration, at algorithm or hardware level. Permanent fault model can be seen in Figure 5 demonstrating only stuck-at-0 (open) and stuck-at-1 (close) faults as being the most representative permanent faults observed. This categorization is based on actual physical realization of nano-crossbar array as reported in [22] and [12] which state that fault rates reach up to 10% and the most common faults are stuck-at-0 (open).

Transient and intermittent faults, on the other hand, manifest themselves due to particular combinations of topological, environmental factors or process mismatches and instabilities. They can be tolerated by proper architectural reconfiguration, but also by instability and variation-aware design. That is to say by using properly sized, or well estimated hardware redundancy in term of spare cells, critically sensitive cells can be circumvented or protected. Transient fault model can have multiple forms meaning they can be seen as parasitic, 0-to-1 (1-to-0) transitions, parasitic positive (negative) pulses, neighbor cells pattern dependent transitions, etc. Transient fault domain is very closely related to a specific technology. Transient faults are more efficiently covered by fault tolerance strategies, as fault avoidance techniques will yield to a much higher area, but also due to the complexity of the technology dependence analysis.

For both type of faults, it is mandatory to perform crossbar sensitivity analysis as the mapping algorithm can be enhanced with fault avoidance properties, and also can be used to further drive the fault tolerance strategy.

A. Defect Tolerance for Diode, Memristor and FET

Defect tolerance is achieved by mapping a target logic function on a defective crossbar using a distinct input and output assignment. This problem is considered as NP-complete [32]. For the worst-case, $N! \times M!$ permutations are required to find a successful mapping for $N \times M$ crossbar. Algorithms in the literature use defect-unaware or defect-aware approach.

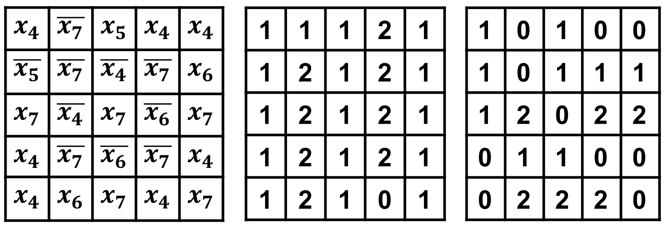
Defect-unaware algorithms aim to find the largest possible $k \times k$ defect-free sub-crossbar from a defective $N \times N$ crossbar where $k \leq N$. The algorithms are inefficient for high fault rates - obtained k values are much smaller than N. In this regard, defect-aware algorithms perform much more satisfactorily [40]. Detailed analysis of both approaches can be found in [41].

Defect-aware algorithms consider the defect characteristics (stuck-at-0 or stuck-at-1), then decide which switch to employ during the mapping. In our previous work [40], we have proposed an efficient heuristic algorithms which aims to match defective crossbar and the given logic function. For this, it denotes crossbars as matrices. 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.

Defect tolerance for nano-crossbars is a well established field with numerous research papers and for this reason we focus on four-terminal architectures which is, to our knowledge, exclusive to this study.

B. Defect Tolerance for Four Terminals Devices

Four-terminal defect tolerance demands a different approach than the architectures 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), called Stuck at Fault Model (SAF). Furthermore, we describe



x_4	$\overline{x_7}$	x_5	x_4	x_4
$\overline{x_5}$	$\overline{x_7}$	$\overline{x_4}$	$\overline{x_7}$	x_6
x_7	$\overline{x_4}$	x_7	$\overline{x_6}$	x_7
x_4	$\overline{x_7}$	$\overline{x_6}$	$\overline{x_7}$	x_4
x_4	x_6	x_7	x_4	x_7

a)

1	1	1	2	1
1	2	1	2	1
1	2	1	2	1
1	2	1	2	1
1	2	1	0	1

b)

1	0	1	0	0
1	0	1	1	1
1	2	0	2	2
0	1	1	0	0
0	2	2	2	0

c)

Fig. 6. a) Lattice design for the example function f and its sensitivity map for b) SAO and c) SA1.

a new model 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 (ACF). In addition, we follow the same terminology adopted in [6] and [17] 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.

1) Defect Injection Methodology: The two fault injections SAF and ACF are repeated for each cell of the lattice. Once a ”defective” lattice is obtained, both 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 fault injection algorithms (one for SAF and one for ACF) 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 lattice is analyzed to face SA0 and SA1, as a model widely adopted today for memristive crossbars. As there is no consensus currently on the fault distribution, we have chosen a uniform distribution for each type of SA0 and SA1 [14]. The fault rates considered may be up to 10% of the crossbar, all faults being independent, as reported in [14]. The fault injection in the above lattices is performed substituting 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 $f = x_4\overline{x_5}x_7 + \overline{x_4}x_6\overline{x_7} + \overline{x_4}x_5\overline{x_6}x_7 + x_4\overline{x_6}\overline{x_7} + x_4x_6x_7$ realized in Figure 6(a) (with the method in [6]), in the Figure 6(b) (resp., 6(c) shows the map containing E^0_{ij} (resp., E^1_{ij}) in each cell. Consider SA0 case (matrix in Figure 6(b)). Each cell of the matrix contains the number of faulty outputs due to a SA0 in the corresponding cell of the lattice 6(a). When a cell of the matrices from Figures 6(b) (SA0) and 6(c) (SA1), contains the value 0, the cell of the corresponding lattice is robust. In this example, the overall number of robust cells for the SA0 (resp., SA1) is $R^0 = 1$ (resp., $R^1 = 10$).

Fault Model 2: Adjacent Cellular Faults (ACFs)

In the classical CFM [16] used 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. In the case of a switching lattice L, the fault model 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 the literal contained 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, with $1 \leq h \leq r$, $1 \leq k \leq s$. 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 $(c_{i,j}, l_{i,j}, l_{i-1,j})$.

For example, consider the lattice depicted in Figure 6(a) and the cell $c_{2,1}$ containing the literal $\overline{x}_5$ in the second row from top and first column from left. In this cell we can have three possible adjacent Cellular Faults: 1) R-ACF: $(c_{2,1}, \overline{x}_5, \overline{x}_7)$ that makes $\overline{x}_7$ the faulty literal of the cell $c_{2,1}$; 2) B-ACF: $(c_{2,1}, \overline{x}_5, x_7)$ that makes x_7 the faulty literal of the cell $c_{2,1}$; 3) T-ACF: $(c_{2,1}, \overline{x}_5, x_4)$ that makes x_4 the faulty literal of the cell $c_{2,1}$. Notice that the cell $c_{2,1}$ cannot be affected by a Left Adjacent Cellular Fault.

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.

2) Metrics used for Sensitivity Analysis: Let us consider a lattice with n input Boolean variables, the main aim of our sensitivity analysis is to understand how many inputs combinations (out of all the possible ones, which are 2^n in number) give an incorrect output value. For this purpose, we inject one error in a cell of the lattice (one error at time), for all cells. This way, we can compute the total number of defective outputs (E^i, with $i \in \{0, 1, L, R, B, T\}$) and the total number of robust cells (R^i, with $i \in \{0, 1, L, R, B, T\}$) as described in the previous section. In order to evaluate the sensitivity of a lattice to SAF and ACF, we propose two metrics. The first one measures the average number of defective outputs face to defect-sensitive cells. The second metric measures the average number of defective outputs in the entire lattice, considering robust and defective, non-robust cells. 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 possible distinct 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 to 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 $i \in \{0, 1, L, R, B, T\}$.

In summary, the first metric measures the impact of one cell on the probability of failure of a logic function. The second one allows us to evaluate the integrity of the entire lattice and understand if it is possible to map strong, critical functions (for a specific operation) in such lattice, which have high sensitivity to faults or overall high robustness. For instance, if we have a lattice where half of the cells are non robust we can understand that we may not want to use it for critical functions (e.g., control, decision making, etc).

For example, consider the lattice depicted in Figure 6(a) and the map in Figure 6(c) showing the values of E^1_{ij} in each cell in the SA1 model. Note that the number of rows and columns is $r = s = 5$ and the number of different variables in the lattice is $n = 4$ (i.e., x_4, x_5, x_6, and x_7). Moreover, the total number of incorrect outputs is the sum of all the errors, i.e., $E^1 = 21$ and the number of robust cells (i.e., cells with no errors) is $R^1 = 10$. Therefore, the sensitivity of defective cells is $S^1_C = E^1/(2^n \cdot (r \cdot s - R^1)) = 21/(2^4 \cdot (5 \cdot 5 - 10)) = 21/240$. This means that out of all possible errors, which are 240 in number, we have 21 errors. The second metrics does not take into account the robust cells, therefore we have that $S^1_L = E^1/(2^n \cdot (r \cdot s)) = 21/(2^4 \cdot (5 \cdot 5)) = 21/400$.

3) Benchmarks and Simulations: The defect simulations have been run on a machine with two Intel Xeon E5-2683 for a total of 64 CPUs and 756 GByte of main memory, running Linux CentOS 7. The benchmarks functions are expressed in PLA form and are taken from a subset of LGSynth93 [47]. Each output of a function is implemented as a separate Boolean function for total of about 1000 functions.

The software used for simulations is written in C++. We used ESPRESSO [23] to implement the method described in [6], 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 [17].

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 [6] and [17]. The benchmarks marked with a $\star$ in Table III, were stopped after ten minutes of SAT execution and, for this reason, they are missing from the benchmarks synthesized using quantified Boolean logic.

More precisely, in both methods, the first column reports the name and the number of the considered outputs 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 III shows that the synthesis method proposed by [6] produces lattices with a lower number of errors and a higher number of robust cells. However, the area of these lattices is greater than the area of the corresponding lattices synthesized using [17]. For example, consider the benchmark circuit alu3(2): the area of the lattice minimized with [6] (resp., [17]) method is 10×11 (resp., 6×4), while the percentage of robust cells in the lattice synthesized using [6] (resp., [17]) method is 98% (resp., 33%) in the SA0 case. For the other fault models the percentage's comparison is similar.

Table IV describes the overall results for the considered benchmarks. More precisely in the columns from one to three is reported the synthesis method, the average lattice area, and the average number of input. Columns from four to twenty-one show the average values of S_C, S_L, and $\%R/r \times s$ for each fault injection methodology. The columns from twenty-two to twenty-four contain the average value of S_C, S_L, and $\%R/r \times s$ obtained considering all the fault models.

Table IV shows that the percentage of cells that are considered robust according to our metrics is higher in the first approach [6]. For example if we consider the benchmark *prom2(0)* with SA1 faults, using the approach [6] we have 100% of resilient cells, instead using the approach [17] there are no resilient cells. This is due to the more constrained structure of the lattices produced by the first synthesis method. Indeed, the method proposed in [6] computes a lattice for f and its dual, which it is in general less compact than the lattice given by [17]. However, we can note that the sensitivity of the lattices is quite low for both methods. In fact, the experiments show that, in general, non-robust cells compute a failed output for a very limited number of inputs. In particular, lattices present lower sensitivity to Adjacent Cellular faults, with respect to the Stuck-At faults. This is due to the fact that when two adjacent cells contain the same value, this situation will not show an impact on the output. In both synthesis approaches, we observe that the lattice is more sensitive to SA0 faults, while in the case of SA1 faults, it requires higher percentage of redundant cells.

It is worth noting that we test lattices with up to 12 variables, this makes interesting the use of this technology to implement multipliers, or other complex arithmetic functions.

4) 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 in terms of area it is (closer to the optimal solution), the higher the defect sensitivity of the output to a cell defect. It is thus mandatory to include the mapping algorithm defect-avoidance heuristics, but

TABLE III
A SAMPLE OF BENCHMARK FUNCTIONS SYNTHESIZED WITH [6] AND [17] 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^R	S_C^R	S_L^R	$\% \frac{R^R}{r \times s}$	E^L	S_C^L	S_L^L	$\% \frac{R^L}{r \times s}$	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 [6]																										
alu3(2)	10×11	8	16	0.008	0	96	4	0.008	0	98	12	0.003	0	87	12	0.003	0	87	4	0.001	0	89	4	0.001	0	89
b12(6)	9×6	9	160	0.028	0.006	80	2072	0.225	0.075	68	80	0.013	0.003	78	72	0.013	0.003	80	2072	0.150	0.075	50	344	0.037	0.012	67
br2(0)*	5×14	12	227	0.002	0.001	50	546	0.003	0.002	41	100	0.001	0	56	125	0.001	0	54	545	0.003	0.002	36	381	0.002	0.001	41
clpl(3)	6×6	11	4085	0.074	0.055	25	1382	0.042	0.019	57	1010	0.019	0.014	28	3342	0.063	0.045	28	1381	0.032	0.019	42	274	0.008	0.004	53
dist(2)*	33×30	8	301	0.005	0.001	78	207	0.007	0.001	89	182	0.004	0.001	84	198	0.004	0.001	82	187	0.005	0.001	86	175	0.005	0.001	85
jbp(44)	2×10	11	576	0.031	0.014	55	1024	0.071	0.025	65	192	0.007	0.005	35	64	0.003	0.002	45	640	0.045	0.016	65	1024	0.056	0.025	55
luc(13)	9×10	6	70	0.024	0.012	50	36	0.019	0.006	68	36	0.017	0.006	62	38	0.017	0.007	61	29	0.014	0.005	64	36	0.014	0.006	57
m4(7)*	32×29	8	514	0.006	0.002	63	278	0.010	0.001	89	354	0.005	0.001	71	341	0.005	0.001	71	271	0.008	0.001	85	276	0.007	0.001	85
max128(9)	10×9	7	188	0.029	0.016	43	51	0.025	0.004	82	135	0.022	0.012	46	134	0.024	0.012	52	43	0.015	0.004	76	51	0.015	0.004	71
prom2(0)	8×6	8	122	0.063	0.014	78	184	0	0	100	92	0.043	0.009	80	92	0.039	0.007	83	168	0	0	94	136	0	0	94
radd(3)*	36×36	8	471	0.012	0.001	88	388	0.017	0.001	93	314	0.009	0.001	89	284	0.008	0.001	88	384	0.013	0.001	91	334	0.011	0.001	91
rd73(1)*	42×42	7	299	0.008	0.001	83	178	0.008	0.001	90	159	0.006	0.001	89	171	0.006	0.001	88	178	0.006	0.001	88	171	0.006	0.001	88
Synthesis with Quantified Boolean Logic [17]																										
alu3(2)	6×4	8	398	0.097	0.065	33	168	0.041	0.027	33	262	0.060	0.043	29	208	0.048	0.034	29	122	0.034	0.020	42	198	0.039	0.032	17
b12(6)	4×5	9	688	0.079	0.067	15	896	0.219	0.088	60	424	0.064	0.041	35	504	0.082	0.049	40	544	0.097	0.053	45	904	0.126	0.088	30
clpl(3)	6×3	11	1218	0.040	0.033	17	1579	0.043	0.043	0	783	0.027	0.021	22	946	0.033	0.026	22	618	0.017	0.017	0	1584	0.043	0.043	0
jbp(44)	2×7	11	2112	0.079	0.074	7	3328	0.116	0.116	0	2048	0.071	0.071	0	2048	0.077	0.071	7	3456	0.121	0.121	0	3648	0.127	0.127	0
luc(13)	6×4	6	86	0.058	0.056	4	115	0.078	0.075	4	82	0.058	0.053	8	85	0.058	0.055	4	94	0.064	0.061	4	105	0.071	0.068	4
max128(9)	6×4	7	269	0.091	0.088	4	210	0.071	0.068	4	216	0.073	0.070	4	260	0.092	0.085	8	225	0.076	0.073	4	234	0.079	0.076	4
prom2(0)	6×4	8	382	0.062	0.062	0	550	0.090	0.090	0	308	0.050	0.050	0	300	0.049	0.049	0	470	0.076	0.076	0	306	0.050	0.050	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	$\% \frac{R^B}{r \times s}$	Average values		
																					S_C	S_L	$\% \frac{R}{r \times s}$
[6]	63	6	0.13	0.11	39%	0.08	0.04	56%	0.02	0.01	35%	0.02	0.01	35%	0.05	0.03	43%	0.03	0.02	45%	0.06	0.04	42%
[17]	15	7	0.1	0.08	20%	0.09	0.08	25%	0.06	0.06	17%	0.06	0.06	17%	0.07	0.07	19%	0.07	0.07	19%	0.08	0.07	20%

hardware-level defect tolerant scheme may also be necessary, especially in the case of very high defect densities and transient faults rates. Hardware redundancy schemes can be used at the lattice column level, or at the block level [27], as the basic computation unit of a memristor array is not the memristive cell, as in classical CMOS based memories, but the entire column. Therefore, several redundant columns or blocks can be added to the initial design to replace the memristive RAM affected columns but also to overcome the situation when potential SAFs affect the redundant parts as well.

Besides Redundancy Repair Techniques, other defect tolerance techniques can be used. For example, Error Correcting Codes (ECC) especially when transient faults are the main issue, but also fault masking techniques to typically repair permanent faults, or combinations of both. Various advanced error correction techniques have been proposed for emerging technologies structures, which provide improvements over the conventional ECC or Redundancy Repair techniques [19]. But it is worth noting that all of them are generating quite significant area overhead due to the encoding/decoding circuitry, or to the extra spare rows and columns, or a more complex addressing and accessing schemes, but they also impact the write/read margins of the lattice, and increase the access latency, and power consumption. Therefore, a combination of efficient defect avoidance mapping algorithm combined with Redundancy based Repair techniques and/or ECC codes will lead to fully functional lattices even for high defect and transient fault rates. 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 can be mapped on highly critical cells, or not, so that the output is not compromised.

Mitigation for Stuck at Faults:

In order to mitigate the sensitivity of a lattice to SAF, we propose the following possible strategy applied to the synthesis method proposed in [6] which has been proven a less sensitive to SAF impact on the output functions: (1) For a given mapped function, if a potential SA0, SA1 faults affects a robust cell identified by the defect injection campaign, the lattice still computes the correct output, thus we do not need any more 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 increase the robustness, we change the literal with the new one. (3) Otherwise, if the injected SA0 fault is proven as being critical for the output value, the column that contains that defective cell has to be replaced by a spare one. 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 7(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 [6]. The example shows one case of mitigation of 3 independent SAD affecting the lattice implementing the function, yielding an approximate 10% defects. In Figure 7, critical SA1 cells are marked in blue and SA0 cells a remarked in red.

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

1) Identify robust cells for a given function mapping. Example: the defect in the first row, fifth column is non-

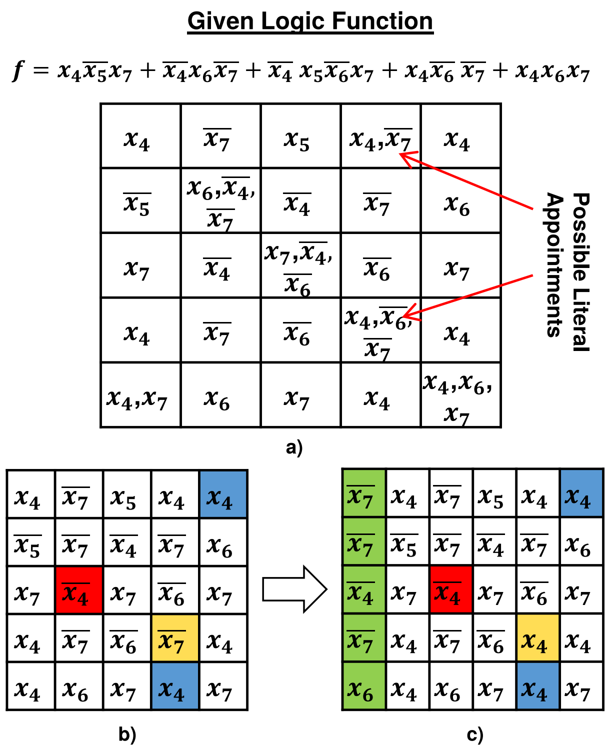

Given Logic Function
$f = x_4\overline{x_5}x_7 + \overline{x_4}x_6\overline{x_7} + \overline{x_4}\,x_5\overline{x_6}x_7 + x_4\overline{x_6}\,\overline{x_7} + x_4x_6x_7$
x_4 $\overline{x_7}$ x_5 $x_4,\overline{x_7}$ x_4
$\overline{x_5}$ $x_6,\overline{x_4},\overline{x_7}$ $\overline{x_4}$ $\overline{x_7}$ x_6
x_7 $\overline{x_4}$ $x_7,\overline{x_4},\overline{x_6}$ $\overline{x_6}$ x_7
x_4 $\overline{x_7}$ $\overline{x_6}$ $x_4,\overline{x_6},\overline{x_7}$ x_4
x_4,x_7 x_6 x_7 x_4 x_4,x_6,x_7
Possible Literal Appointments
a)
x_4 $\overline{x_7}$ x_5 x_4 x_4
$\overline{x_5}$ $\overline{x_7}$ $\overline{x_4}$ $\overline{x_7}$ x_6
x_7 $\overline{x_4}$ x_7 $\overline{x_6}$ x_7
x_4 $\overline{x_7}$ $\overline{x_6}$ $\overline{x_7}$ x_4
x_4 x_6 x_7 x_4 x_7
b)
$\overline{x_7}$ x_4 $\overline{x_7}$ x_5 x_4 x_4
$\overline{x_7}$ $\overline{x_5}$ $\overline{x_7}$ $\overline{x_4}$ $\overline{x_7}$ x_6
$\overline{x_4}$ x_7 $\overline{x_4}$ x_7 $\overline{x_6}$ x_7
$\overline{x_7}$ x_4 $\overline{x_7}$ $\overline{x_6}$ x_4 x_4
x_6 x_4 x_6 x_7 x_4 x_7
c)

Fig. 7. a) defect-free lattice; b) lattice with defects: SA0 in red and SA1 in blue; and c) lattice with the defect fixed.

significant on the value of the output (robust cell). The sensitivity map obtained through defect injection campaign, shows that this cell (x_4) is not sensitive to SA1 for the mapped function.

2) Identify the swapping of literals during synthesis process on a column of a high sensitive cell. Example: the defect in fifth row, fourth column (i.e., blue cell) is sensitive to SA1. We can note that the yellow cell (the fourth row, fourth column cell) contains the literals $\overline{x_7}$. By Figure 7(a) we know that we can choose x_4 instead of $\overline{x_7}$. With this new literal (x_4), we have an equivalent lattice where if the blue cell is affected by a SA1 at fabrication time, this 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, with critical cells, 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:

In the case of adjacent cellular fault model, we can note that adjacent cells containing the same controlling literal are robust to faults. For example, consider the lattice depicted in Figure 6(a), and the two variables x_4 at the end of the first row. The lattice is obviously robust to a L-ACF in the last x_4 (and to a R-ACF in the forth cell, containing x_4). For this reason, in sensitivity analysis to ACF model, it is of huge importance to maximize the number of adjacent cells with the same controlling literal.

Recall that the properties of the lattice synthesized with [6] guarantee that row and column permutations for a given lattice do not change the Boolean function computed by the switching lattice. Because of the good level of flexibility, we consider the lattice obtained by this synthesis method as a starting point for increasing the number of robust cells without changing the function represented.

Thus, given a lattice synthesized with [6], a possible strategy is to perform column and row permutations in order to maximize the number of adjacent cells containing the same literal. For example, consider the R-ACF, the number of cells with the same literal in the right-adjacent cell in the lattice in Figure 6(a) is 1. Suppose to simply swap the second and the third column, the number of cells with the same literal in the right-adjacent cell becomes 4 (i.e., the number of robust cells is incremented by 3). Note also that while the number of robust cells to R-ACF and L-ACF can be increased by column permutations, the number of robust cells to T-ACF and B-ACF can be increased by row permutations. Finally, observe that this method increases the number of robust cells without increasing the dimension of the lattice.

C. Transient Fault Tolerance

Before moving on to transient faults, one distinction should be emphasized such that contrary to defects, transient faults are not known in advance and occur in-field. For this reason, transient fault tolerance works in predictive manner. In addition, since nano-crossbars are in the early stage of development, there is data deficiency regarding the occurrence rates and characteristics of transient faults.

First, we follow the same naming conventions of defects (permanent faults) for transient faults in modeling. Accordingly, the effects of transient faults can be categorized with 4 different scenarios:

1) **Stuck-at-zero** on unused crosspoint produces no effect
2) **Stuck-at-zero** on used crosspoint removes corresponding input
3) **Stuck-at-one** on unused crosspoint adds corresponding input
4) **Stuck-at-one** on used crosspoint produces no effect.

To mitigate faults, first approach is to introduce extra circuit elements to compensate erroneous results. Inserting redundant components can be constructed with adding extra rows and/or columns as shown in [29] [7]. A detailed study of redundancy techniques can be found in [41].

The second approach requires the knowledge of prevalent fault types beforehand so that it is possible to modify the target logic function accordingly. If stuck-at-zero faults are more common, then using the logic function form with least number of inputs implies many unused crosspoints and high tolerance (scenario 1). If stuck-at-one faults are more common, then using the logic function form with the most number

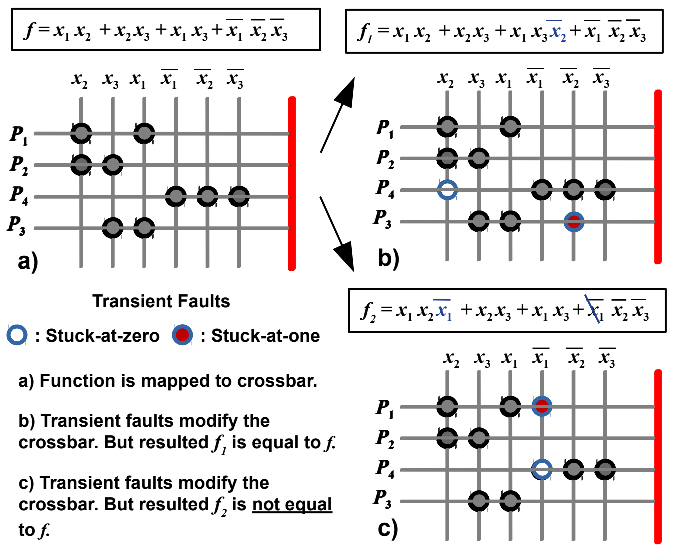

$f = x_1x_2 + x_2x_3 + x_1x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$
$f_1 = x_1x_2 + x_2x_3 + x_1x_3\overline{x_2} + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$
$f_2 = x_1x_2\overline{x_1} + x_2x_3 + x_1x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$
P_1 P_2 P_4 P_3
a) b) c)
Transient Faults
: Stuck-at-zero : Stuck-at-one
a) Function is mapped to crossbar.
b) Transient faults modify the crossbar. But resulted f_1 is equal to f.
c) Transient faults modify the crossbar. But resulted f_2 is not equal to f.

Fig. 8. Transient faults and how inherent tolerance works.

of inputs (e.g. using the minterm form) implies many used crosspoints and high tolerance (scenario 4). Nevertheless, trade-off between area size and transient fault tolerance is open to further research.

Finally, logic functions possess an inherent tolerance independent of methodology and benefits from the equivalence of different logic functions. This is an intrinsic feature of the given logic function having many forms. Figure 8 shows how different faults modify the crossbar and resulted logic function is equivalent to the original in Figure 8(b) and not in Figure 8(c). These tolerated faults are limited to certain switches of crossbar and can be found with using algebraic calculations as show in [40]. As for the example in Figure 8 $f = x_1x_2 + x_2x_3 + x_1x_3 + \overline{x_1}\ \overline{x_2}\ \overline{x_3}$, every following function is equivalent to the original one:

- $f_1 = x_1x_2 + x_2x_3\overline{x_1} + x_1x_3 + \overline{x_1}\ \overline{x_2}\ \overline{x_3}$
- $f_2 = x_1x_2\overline{x_3} + x_2x_3 + x_1x_3 + \overline{x_1}\ \overline{x_2}\ \overline{x_3}$
- $f3 = x_1x_2 + x_2x_3 + x_1x_3\overline{x_2} + \overline{x_1}\ \overline{x_2}\ \overline{x_3}$

For more detailed explanations of determining tolerable switches, calculating the fault tolerance and effects of multi-output functions reader can refer to [40].

Eventually, a more resource hungry approach would be fault sensitivity analysis by performing extensive Monte Carlo simulation and determining every possible case of fault occurrence, yet it is rather costly. As a final word, transient fault tolerance, at the moment, is at its infancy and additional research is needed for best practices.

V. Performance Optimization

We focus on area-oriented optimization for our methodology. As stated in Section I, main motivation to use nano-crossbars is that they are dense. Thus, logic functions can be realized with smaller area. We first find the minimum sized array, and perform defect tolerance analysis. If it is needed, we increase the area size based on the defect tolerance results. Later, we perform delay and power performance analysis with minor tuning. However, as a downside we loose the benefit of further delay and power minimization [13] [5].

Section III investigates the area performance. Here, we will analyze the delay and power depending on number of products and/or literals. Main purpose is to construct a fair metric for performance comparison. In our previous study [43], we have conducted this analysis for memristive crossbars. All supply voltages of technologies are assumed to be the same. Therefore, comparing their performance behaviors (for a given function) is considered to be sufficient method for integrated synthesis methodology.

1) Delay Analysis: We have used an approach of multiplying resistive and capacitive loads (RC-delay) 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 $7 \times t_{cycle}$ (independent from the target function; i.e. latency). Considering FET, we need to calculate resistive loads and capacitive loads from the longest path for the worst-case scenario. Lastly, four-terminal's delay is directly proportional to the longest path on the lattice, same as FET. The longest path on the lattice can be calculated as explained in [6]:

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

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

Finally, **Delay** performances are proportional to:

- **Diode**: $\propto (load\_resistor) \times (\#\ of\ literals\ in\ f)$
- **Memristor**: $\propto 7 \times t_{cycle}$ (latency) [43]
- **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**: $\propto L_{long\_path}$

2) Power Analysis: Power consumption is the 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 a period at maximum frequency). Because we assumed that circuit will be used in maximum frequency. For delay estimation, we will going to use the formulation in above.

The power of diode based crossbar is dominated by static power. So, it is inversely proportional only with a load resistor. Load resistor can be considered as 10 times of diode's inner resistance. Total energy consumption of memristor is proportional to total count, because it is assumed that all memristors are changing their states one time only [43]. Note that, in this memristor based crossbar, sneak path current is minimized [20], so the power consumption caused by it can be neglected. Total energy consumption of FET crossbars is directly proportional to the capacitance of the output node and nodes related to the output node. For four-terminal, we assume all crosspoints consume energy at the worst-case scenario. Finally, we divide energy consumption by the delay to find the complexity of power consumption.

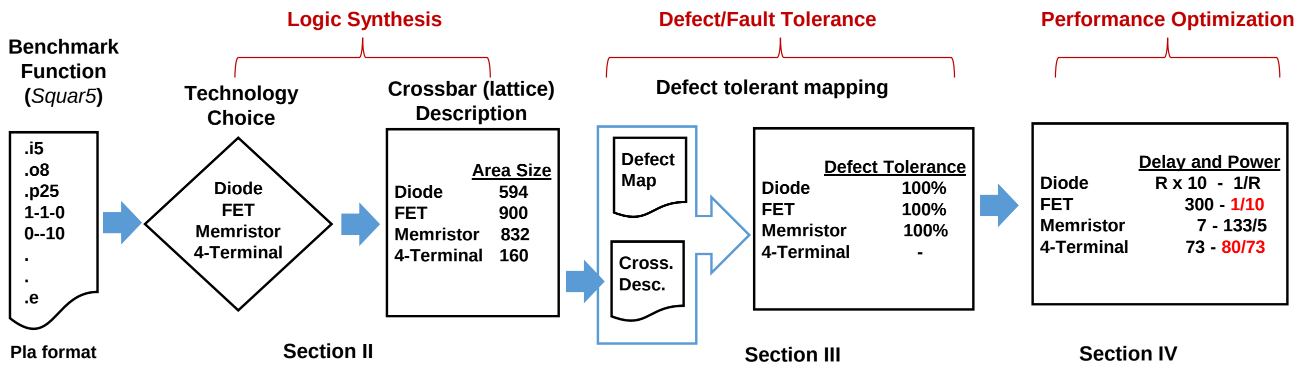


Fig. 9. Whole integrated synthesis pipeline of benchmark function *squar5*. R denotes the $load\_resistor$.

However, effect of the estimated switching activity of an input to the power consumption is not the same for all technologies. For memristor, even though change in inputs does not cause any change for the state of the output, there will be dynamic power consumption. Because every cycle (stage) will be recalculated. Also, since the power of diode is dominated by static power, we didn't include the effect of switching activity to the calculation of power consumption. For only FET and four-terminal, we include the estimation of switching activity by multiplying it with 0.5. (Notice that, supply voltages are assumed as the same.)

Finally, **Power** performances are proportional to:

- **Diode**:$\propto 1\ /\ (load\_resistor)$
- **Memristor**: $\propto (total\ memristor\ count\ in\ crossbar)/7$ [43] [39]
- **FET**: $\propto 0.5\ /\ (degree\ of\ the\ largest\ product\ in\ f\ and\ f^D)$ (@ maximum frequency)
- **Four-terminal**: $\propto$ $(0.5\ (\#\ of\ products\ in\ rows) \times (\#\ of\ products\ in\ columns))\ /\ L_{long\_path}$

VI. 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 9.

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

four-terminal: $(degree\ of\ the\ largest\ product\ in\ f^D_{oi}) \times (\#\ of\ products\ of\ all\ f_{oi} + n - 1)$ = 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 IV, 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% [41] [43]; 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 (latency) is constant (independent from the function) and $7t_{cycle}$. 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 1/10. Including the next four-terminal, FET has the largest delay.

Four-terminal: The longest path is 73, so is the delay. Power is 80/73 $\approx$ 1.

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 the least value, yet it is static power also depends source voltage. On the other hand, FET crossbars has only dynamic power consumption with complementary architecture.

VII. Conclusion and Discussion

In conclusion, we have presented an integrated synthesis methodology regarding every design steps of a crossbar circuit including novel optimization and defect tolerance algorithms for each step. Since the main motivation of crossbar technologies is that they are able produce denser circuit structures comparing to the conventional technologies, the synthesis methodology takes area optimization as base. In addition to that, defect tolerance analysis gets the size of the crossbar as the input. However, devising area-oriented optimization causes only having course delay and power analysis/optimization. An extension of the selection of delay- or power-oriented optimizations will increase the performance optimization capability of the methodology.

In logic synthesis section, comparison of four crossbar types with multi-output logic functions is firstly presented. Also, a new technique of logic synthesis for memristor based crossbars is presented. Since developing new efficient algorithms for logic synthesis is in progress and also the area sizes are depending on different parameters of a logic, there has not been a verdict on this race.

In defect tolerance section, we focus on four-terminal architectures and design a novel sensitivity analysis and fault tolerance approach. First step of the method locates the problematic cells and guides the second step through introducing preventive measures such as reassignment of cell (if possible) or adding redundant columns/rows to the lattice. Currently, method is limited in term of input size, so an important improvement to sensitivity analysis would be to increase the number of variables the method can manage.

In performance optimization section, effects of the crossbar size and function parameters to the delay and power are presented. Even though room for optimizations of delay and power are limited due to the area-oriented optimization, the paper presents a clear projections to the designer. However the real values in the crossbar technologies (for different types) is in process of evolving. Certain studies are in proof of concept level. For instance, pitch sizes were given as micrometers in the early 2000s [18] [34], yet they are reduced to tens of manometers [20]. Also, a recent study [13] realizes four-terminal switch based crossbar (lattice) with 65nm CMOS-process for a proof of concept. As expected, the situation is also same for the comparison on the delay and power analysis. This is the inherent reason that this study is independent from technology nodes, thus it can be implemented to the every technology. 3D printing electronics could be a promising candidate for comparing the crossbar types with actual values.

As a future work, a fully automated software tool for the synthesis methodology is planned to be developed. It will take 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 produce the mapping using technology specification data. The tool is expected to be updated based on established experimental results and novel algorithms. Another direction could be the study of transient faults on lattices, so that it can generate complete defect tolerance analysis. The future study should evaluate how the redundancies in lattices can limit the area overhead.

Acknowledgement: We thank Dr. Levent Aksoy for his contribution to synthesis process of multi-output functions with four-terminal based switching arrays.

References

[1] L. Aksoy and M. Altun, "Novel methods for efficient realization of logic functions using switching lattices," *IEEE Transactions on Computers*, vol. 69, no. 3, pp. 427–440, 2019.

[2] L. Aksoy and M. Altun, "A satisfiability-based approximate algorithm for logic synthesis using switching lattices," in *Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2019, pp. 1637–1642.

[3] L. Aksoy and M. Altun, "A novel method for the realization of complex logic functions using switching lattices," in *Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2020.

[4] 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 *Euromicro Conference on Digital System Design (DSD).* IEEE, 2016, pp. 334–341.

[5] M. Altun, I. Cevik, A. Erten, O. Eksik, M. Stan, and C. A. Moritz, "Nano-crossbar based computing: Lessons learned and future directions," in *Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2020.

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

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

[8] R. Ben-Hur, R. Ronen, A. Haj-Ali, D. Bhattacharjee, A. Eliahu, N. Peled, and S. Kvatinsky, "SIMPLER MAGIC: synthesis and mapping of in-memory logic executed in a single row to improve throughput," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[9] A. Bernasconi, "Composition of switching lattices and autosymmetric boolean function synthesis," in *2016 Euromicro Conference on Digital System Design (DSD).* IEEE, 2017.

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

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

[12] J. Borghetti, Z. Li, J. Straznicky, X. Li, D. A. Ohlberg, W. Wu, D. R. Stewart, and R. S. Williams, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proceedings of the National Academy of Sciences*, vol. 106, no. 6, pp. 1699–1703, 2009.

[13] I. Cevik, L. Aksoy, and M. Altun, "Cmos implementation of switching lattices," in *Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2020, pp. 274–277.

[14] 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.

[15] 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, pp. 462–468, 2003.

[16] A. D. Friedman, "Easily testable iterative systems," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1061–1064, 1973.

[17] 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, pp. 1–14, 2014.

[18] 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.

[19] W. Kang, W. Zhao, Z. Wang, Y. Zhang, J.-O. Klein, Y. Zhang, C. Chappert, and D. Ravelosona, "A low-cost built-in error correction circuit design for STT-MRAM reliability improvement," *Microelectronics Reliability*, vol. 53, no. 9-11, pp. 1224–1229, 2013.

[20] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications," *Nano letters*, vol. 12, no. 1, pp. 389–395, 2012.

[21] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[22] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature communications*, vol. 9, no. 1, pp. 1–8, 2018.

[23] P. C. McGeer, J. V. Sanghavi, R. K. Brayton, and A. Sangiovanni-Vicentelli, "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 4, pp. 432–440, 1993.

[24] M. C. Morgul, O. Tunali, M. Altun, L. Frontini, V. Ciriani, E. I. Vatajelu, L. Anghel, C. A. Moritz, M. R. Stan, and D. Alexandrescu, "Integrated synthesis methodology for crossbar arrays," in *14th IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2018, pp. 91–97.

[25] 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.

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

[27] 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.

[28] 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. 4, no. 4, pp. 522–532, 2018.

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

[30] S. Safaltin, O. Gencer, M. C. Morgul, L. Aksoy, S. Gurmen, C. A. Moritz, and M. Altun, "Realization of four-terminal switching lattices: Technology development and circuit modeling," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 504–509.

[31] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Logic synthesis for RRAM-based in-memory computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1422–1435, 2017.

[32] A. M. S. Shrestha, S. Tayu, and S. Ueno, "Orthogonal ray graphs and nano-PLA design," in *2009 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2009, pp. 2930–2933.

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

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

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

[36] D. B. Strukov and K. K. Likharev, "CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, no. 6, p. 888, 2005.

[37] M. B. Tahoori, "A mapping algorithm for defect-tolerance of reconfigurable nano-architectures," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2005, pp. 668–672.

[38] P. L. Thangkhiew, A. Zulehner, R. Wille, K. Datta, and I. Sengupta, "An efficient memristor crossbar architecture for mapping boolean functions using binary decision diagrams (bdd)," *Integration*, vol. 71, pp. 125–133, 2020.

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

[40] 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, 2016.

[41] O. Tunali and M. Altun, "A survey of fault-tolerance algorithms for reconfigurable nano-crossbar arrays," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–35, 2017.

[42] O. Tunali and M. Altun, "Logic synthesis and defect tolerance for memristive crossbar arrays," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 425–430.

[43] 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.

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

[45] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "A mapping methodology of boolean logic circuits on memristor crossbar," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 311–323, 2017.

[46] 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.

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

[48] J. Yao, H. Yan, S. Das, J. F. Klemic, J. C. Ellenbogen, and C. M. Lieber, "Nanowire nanocomputer as a finite-state machine," *Proceedings of the National Academy of Sciences*, vol. 111, no. 7, pp. 2431–2435, 2014.