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].

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 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.

*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, 18-19 July 2018, Athens, Greece

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

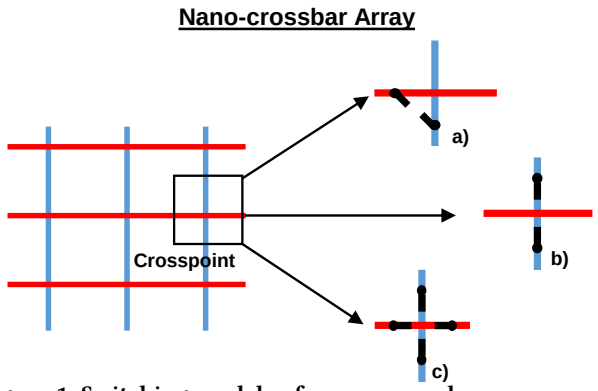

Figure 1: Switching models of a nano-crossbar array: crosspoint as a) two-terminal switch with terminals in the crossed lines, b) two-terminal switch with terminals in the same line, and c) four-terminal switch.

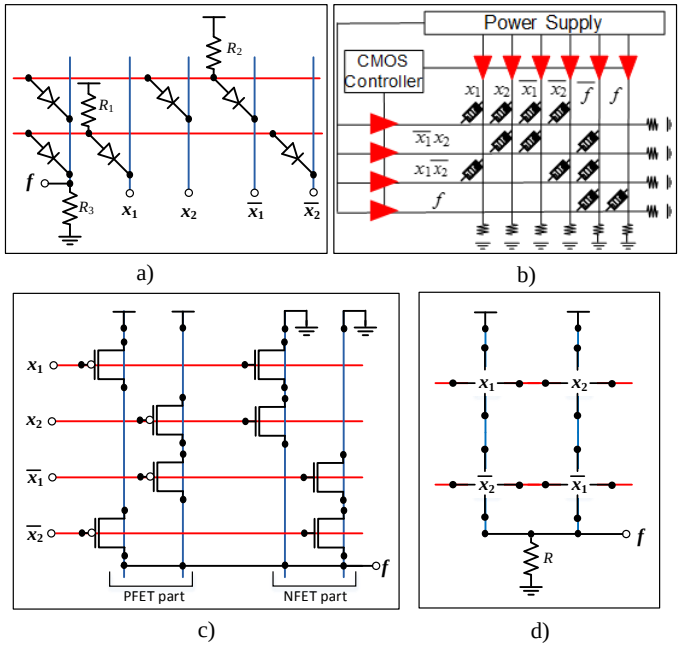

Figure 2: Implementation of f_{XOR_2} with different nano-crossbar types: crosspoint as a) diode, b) memristor, c) FET, and d) four-terminal switch.

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

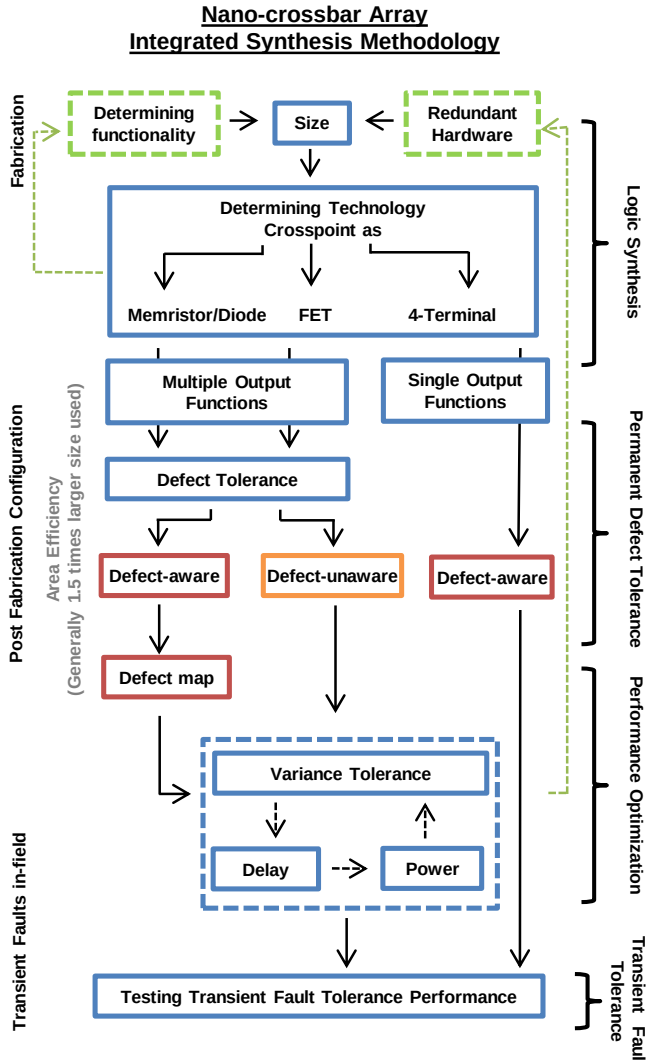

Figure 3: Integrated synthesis methodology scheme for nano-crossbar arrays.

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).

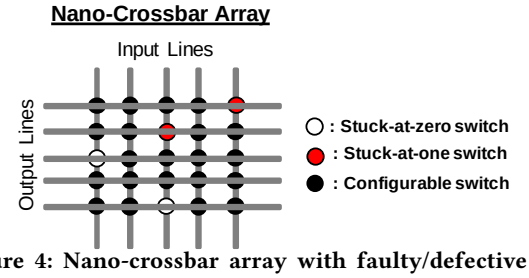

Figure 4: Nano-crossbar array with faulty/defective cross-points.

- **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

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)

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 $f = x_4\overline{x_5}x_7 + \overline{x_4}x_6\overline{x_7} + \overline{x_4}x_5\overline{x_6}x_7 + x_4\overline{x_6x_7} + x_4x_6x_7$ 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^0_C	S^0_L	$\%\frac{R^0}{r\times s}$	E^1	S^1_C	S^1_L	$\%\frac{R^1}{r\times s}$
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^0_C	S^0_L	$\%\frac{R^0}{r\times s}$	S^1_C	S^1_L	$\%\frac{R^1}{r\times s}$
[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^0_C, the Sensitivity of lattice S^0_L 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.

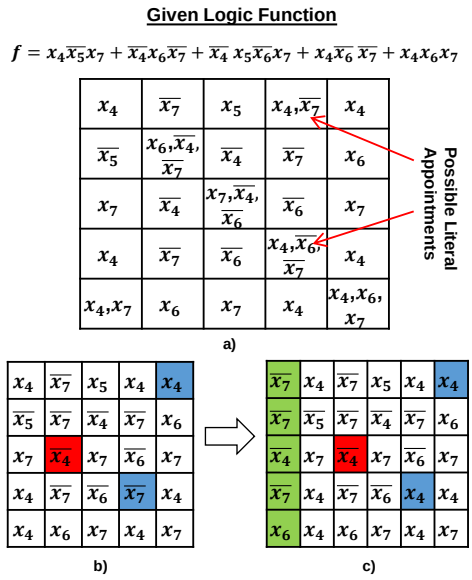

Figure 6: a) defect-free lattice; b) lattice with defects: SA0 in red and SA1 in blue; and c) lattice with the defect fixed.

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.