Fault tolerant nanoarray circuits: Automatic design and verification

Fault Tolerant Nanoarray Circuits: Automatic Design and Verification

P. Ranone, G. Turvani, F. Riente, M. Graziano, M. Ruo Roch, M. Zamboni

***Abstract*—We automatically maximize fault-tolerance in nanoarrays based on silicon nanowires and Gate-All-Around transistors optimizing their topology vs. several distributions of faults inherited by technology. We added a MonteCarlo engine in our nanoarchitecture design tool ToPoliNano and verified the effectiveness of the fault-tolerance algorithm over several circuits and faults distributions.** [1]

***Keywords*—Nanoarrays, Fault-tolerance optimization, Monte-Carlo, emerging technology, CAD for nanoelectronics.**

I. Introduction

Nanoarray circuits represent the beyond-CMOS technology nearer to scaled CMOS [1] [2] [3]. Proposed circuits topologies have in common the elementary device, i.e. the Gate-All-Around (GAA) transistor based on silicon nanowires (SiNW), and the regular organization, namely PLA-like in most of the cases (a brief overview is in section II). Both characteristics are predictive of high compactness, good trade-off between power and frequency, and remarkable predisposition for massive parallel computation [4]. Nevertheless defects will be for long time one of the main concerns from the technology side [3] [5]. Though experimental demonstrations have been already presented [6] [2], processes are still not mature. The extremely scaled feature sizes used, combined with the expected density, give a perspective where high fault rates will be for long time the default scenario (see section III). Coping with high defective rates from the design and architectural point of view is unavoidable to mask the most critical aspects limiting the integration of nanoarrays with scaled CMOS in order to exploit as much as possible their remarkable potentials.

Previous works (see section II) approached the problem by inspecting the possible fault tolerant techniques of nano-PLA [7], by using a redundant covering approach at software level [8], or by developing a high level simulator [5] for identifying the best fault masking for nanoarray circuits. None of the above comprises the main features that are emerging as mandatory by literature and technology: A) a fault-tolerant algorithm that acts at the single nanoarray design level optimizing the wire and transistor detailed topology not necessarily relying on redundancy; B) a detailed simulator able to consider transistor by transistor what happens in presence of faults before and after optimization; C) the ability to take into account faults distributions with detailed geographical localization precisely related to data from the technology.

Our contribution (see Fig. 1) goes toward this direction with a fresh approach (see section IV) with respect to the state of the art. 1) We implemented an automatic algorithm

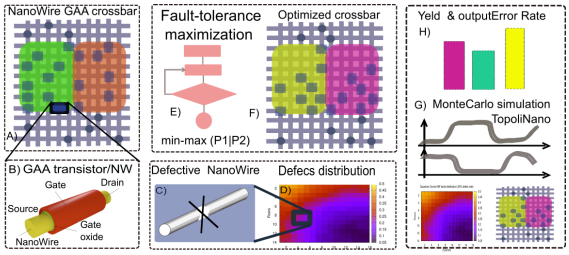

Figure 1: Contribution: For nanoarrays A) based on B) SiNWs and GAA devices, we consider faults C) on NWs and devices and D) a localization aware distribution of defects. A fault tolerance algorithm E) based on faults statistics optimizes topology F). An accurate engine considers devices, wire topology and defects and executes MonteCarlo simulation exploiting multithreading and G) evaluates yield and error rate.

(section V) to maximize fault-tolerance in nanoarray logic components based, at the moment and as case study, on NAND-NAND PLA-like dynamic nanoarrays inspired to [5] and evolved to [9] [4]. The method 1.1) considers the detailed topology of both SiNW and GAA transistors for a given function, 1.2) relates it to the faults distributions inherited by technological processes counting both faults probability and faults topological distribution, and 1.3) optimizes the nanoarray organization accordingly; this is obtained through an ad-hoc developed algorithm, FaTToR, based on the *min-max* optimization of a penalty matrix P created on the basis of a combination of logic function and faults distribution. 2) We enriched our tool for nanoarchitecture design, ToPoliNano [9] [10] [4], with a MonteCarlo engine (section IV and VI) for the analysis of faulty arrays able to: 2.1) design the circuit using as components the nanoarrays (either-tolerant or not), 2.2) execute detailed switch level simulations of exaustive input patterns, 2.3) introduce faults distibutions taking into account not only faults probabilities but also faults localization, 2.4) run MonteCarlo style simulations considing faults exploiting parallel execution (multithreading C++ efficient implementation) and 2.5) collect the final output error rates (*OER*) and Yelds (Y). 3) We tested (results in section VI) the effectiveness of the fault-tolerance method 3.1) executing exhaustive MonteCarlo simulations over several circuits, 3.2) considering different probability and topological distributions of faults, 3.3) analyzing the effectiveness of circuit reorganization.

We believe this work is a fundamental contribution to the capabilities now available to cope with the high defective rates of nanoarrays and with the necessity to deal with the design and verification of defective emerging circuits conceived for massive parallel computation.

[1]Authors are with the Electronics and Telecommunication Department, Politecnico di Torino, Corso Duca degli Abruzzi 24, Italy.
C.author: Mariagrazia Graziano, E-mail: mariagrazia.graziano@polito.it, Tel: +390110905172, Fax: +390110904117

II. Background

Nanoarray based circuits are characterized by a 2-D grid of semiconductor NWs, where active devices (GAAs) are realized at certain crosspoints. This circuit organization is derived from [5]. However, the structure we use is evolved from the original [4], as we added *buffers* in order to route signals for connecting the crossbar to other crossbars and to the I/O (e.g we designed it using input, output and connection buffer). This solution saves area without losing the benefit of the high regularity of this technology. Thus, bigger systems can be obtained cascading many tiles (elementary crossbars) connected together by connection buffers. According to [5], the dynamic two-level nanocircuit behavior is sequenced by four control signals: *hpre, heva, vpre, veva*. Microwires placed on the periphery are used to supply power to the circuit. Fig. 2 shows an implementation of the AND function with a two-level dynamic NAND-NAND logic style, where input signals are routed toward the tile by an input buffer. The two logic planes will be herein referred to as NAND1 and NAND2.

Unconventional manufacturing process based on self-assembly are not mature and produce an extremely high number of defects with respect to standard CMOS process. On the other hand, top-down lithography can reduce this number, but the limitation is inherent to lithography resolution and to the higher cost of the fabrication process. Nonetheless, extrapolation data to 9-nm node in [2] show a doubling of the performance and x67 area saving with respect the CMOS implementation of a given function. Furthermore, several techniques have been developed to integrate III-IV materials on silicon [6]. Due to the current fabrication limit, defects are difficult to be detected in the circuit. Two typical kinds of faults in nanoarray devices are depicted in Fig. 2 insets: stuck-on transistor on the top and a broken horizontal nanowire on the bottom (in section III a formal classification is given). Even if the fabrication process is currently unreliable due to technological limits, researchers are looking for architectural and system level techniques to reach an acceptable yield and output error rate. One of the outcomes of this approach is assessing the limits to which technology should set toward to assure circuits with overall satisfying reliability and thus competitive if compared to CMOS standard structures.

The AND depicted in 2 has a structure very similar to a PLA, not AND-OR but NAND-NAND based. This similarity suggests that fault models developed for CMOS PLAs, such as fault masking schemes, can be adapted also to nanodevices [7]. In literature, previous works on fault tolerance exploit the unequal fault probabilities of faulty ”0”s and ”1”s to increase yield, using biased techniques combined with some levels of structural redundancy or based on majority voting [5] [8]. However, these approaches increase the yield but suffer of an important limitation: they take into account the intrinsic logic behavior of the NAND-NAND architecture without considering the internal organization of the circuit (i.e. NWs and GAAs topology). Another problem is that the redundant technique increases a lot the total sizes of the circuit. Moreover, a part from [8], such techniques are based on the assumption that defects are present mostly in correspondence of the transistors. Since microwires manufacturing process can be considered reliable as based on standard technology, the kind of faults that can appear inside the nanoarray can be of four types: stuck-on, stuck-off, broken nanowire, bad ohmic contact. In this paper we are considering only faults related to the two NAND planes, NAND1 and NAND2, so we exclude possible faults on control and power lines. The details about these kind of faults and about the way we include them in our method are described in the section III.

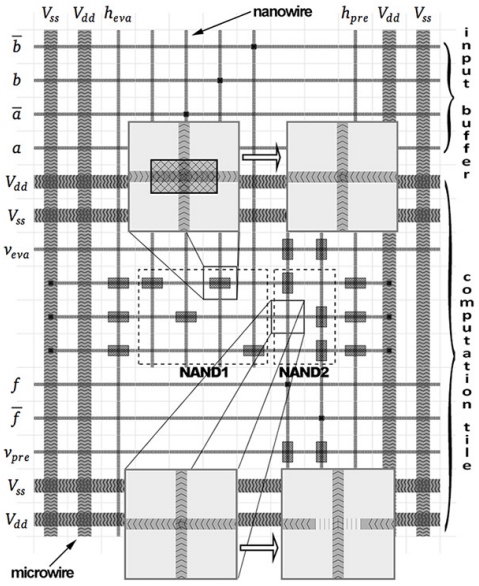

Figure 2: AND function implemented with 2-level NAND-NAND logic style. Insets show examples of stuck-on transistors (top) and broken nanowire (bottom).

Our approach is based on an algorithm, FaTToR, that takes into account the transistor disposition inside the two logic planes, in order to increase yield without increasing the circuit area. It is strictly related to the fault distribution throughout the circuit. Both these features are absent in almost all previous approaches. In [8] a partial knowledge of the circuit internal organization is provided, while faults are not associated to the circuit structure nor specific statistical faults distributions are considered out of a uniform random statistic.
Another important contribution of our method is the analysis of the effectiveness of FaTToR results through accurate switch level simulations that allow to consider the impact of defects positioned in specific points of the array, both on GAAs and on NWs, in presence of a given statistics and fault probability. No tools are available to simulate nanoarray based technologies with the required level of detail. We then enriched our automatic CAD tool ToPoliNano [9], that now allows us to perform MonteCarlo simulations on different kind of emerging technologies, and in this particular case on nanoarray based circuit (see section IV).

III. Faults classification

In this section we present the four main categories of faults that can be treated in our tool. A detailed analysis can be found in [1] [5]. The first two concern transistor (TR) defects: a stuck-on TR is considered as always on, independently from the logic level of the input, leading to a persistent short-circuit between drain and source. On the contrary, stuck-open TRs are always off, so there is no connection between drain and

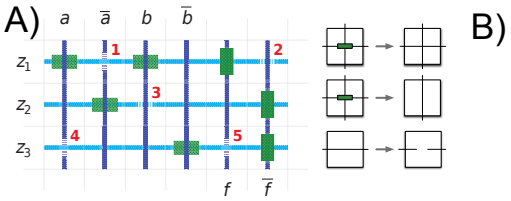

Figure 3: A) Broken NW faults on an AND gate B) Defective sub-tiles remapping

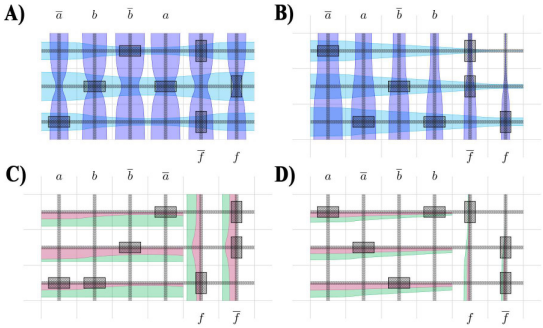

Figure 4: AND circuit with faults distribution examples. A) G1G2: G2 distribution on hor. NWs and G1 distribution on vert. NWs. B) SWSW: corner linear distribution with south-west inclination both on vert. and hor. NWs. C) G2G1 on dev.: G2 distribution on hor. GAAs and G1 distribution on vert. GAA. D) SWSW: corner linear distribution with south-west inclination on vert. and hor. devices.

source. These defects can be treated with techniques such as hardware redundancy or reconfiguration. Furthermore, NWs can be broken and when this happens logic values cannot be propagated through the circuit. The fourth fault worth mentioning is the bad ohmic contact, i.e. a bad connection between microwire and NW. The consequence is a wrong polarization of the circuit.

The stuck-on TRs are the most prevalent kind of defect [5] due to the manufacturing process. Nevertheless, broken NW faults are equivalently important. It is reasonable to assume that there is a direct relationship between NW length and the probability of having a break. Since these kind of defects cause an interruption in signal propagation, it is important to analyze how the position of the faults impacts the correct behavior of the circuit. It could happen, for example, that for a given state the output is not affected by a break. Fig. 3.A shows the circuit of an AND gate in which five types of faults are highlighted. Herein we consider these different cases independently, assuming thus that at a given time only one single fault can occur. **Fault '1'**: if we have a logic 0 nothing happens, because there is no signal to be propagated and no activation has to be done in correspondence of the TR. Otherwise, in case of a logic 1, the broken NW leads to an error. **Fault '2'**: It is never possible to totally charge the cube z_1. Then, nothing happens when z_1 has to be discharged, otherwise an error occurs. **Fault '3'**: as long as the cube z_2 is to be kept to logic 1 nothing happens, because the line, on the right part of the circuit will be always charged. Otherwise, the loss of any kind of possible connection to ground prevents the line to be totally discharged. **Faults '4'**: this kind of fault does not cause any error at all and it is totally harmless. **Faults '5'**: if the output f has to be equal to 1, the circuit works as expected. Otherwise, there is no way for the cubes to create a connection to ground, and so the logic 0 will be never produced.

Fig. 4 shows a AND gate circuit where different faults distributions are applied on both NWs and TRs. Considering the technology process it is in fact very likely that there is a non uniform distribution of faults, especially in the case of NWs. Here we considered three kind of distributions: I) Gaussian type 1 (herein identified as G1): $1 + e^{-x^2} - e^{-y^2}$ II) Gaussian type 2 (herein identified as G2): $1 - e^{-x^2} + e^{-y^2}$ III) Corner linear distribution which can have four inclinations: north-east (NE), south-east (SE), south-west (SW), north-west (NW). We finally consider also a standard uniform random distribution (herein identified as R) the only one normally adopted in previous works. Each of these distributions can be applied to NWs and devices, both horizontally and vertically. In our software circuits are described using different tiles, each of them is represented by a matrix in which each node corresponds to a particular sub-tile like NW, microwire, TR and so on. In this way it is possible to easily substitute a defective element with its corresponding model according to Fig. 3.B.

IV. Fault tolerant nanoarrays design and test: the method

Our tool involves the interaction between two different software engines. The first one, the Component Generator (CG) is tasked with creating the basic circuit components that, when added to the library of ToPoliNano, can be instantiated in the VHDL description of complex circuits. The second software is the CAD ToPoliNano which, starting from the VHDL description of the circuit and the basic blocks produced by the CG, generates the circuit layout and performs the logic simulation. As ToPoliNano itself has been described in [4] [9] not further details on the standard version are given herein.

Since defects may have different impacts on circuit behavior depending on their position within the circuit, our approach consists in finding a circuit disposition with the objective of maximizing the cases in which these faults, caused by breaking NWs or defective transistors, become as harmless as possible. In this first phase we act on the matrix representative of the simulation circuit, exchanging, according to appropriate rules, the order of the rows and columns. The algorithm we developed, FaTToR (Fault Tolerance Topology Reorganization algorithm), explained in section V, heuristically elaborates an alternative device disposition, improving Yield (Y) or output Error Rate (OER) with no cost in terms of circuit area, performance and power. The resulting optimized circuit is elaborated by the CG in order to create the optimized library component for ToPoliNano. With this second tool we perform the accurate logical simulation of the circuit at switch level (i.e. considering the signal propagation throughout the NAND1 and NAND2 planes according to inputs and control signals configuration). For this work we enriched the tool with an embedded MonteCarlo approach: we iterate the simulation n times applying different fault distributions with different probabilities. The system behavior is in Fig. 5. Finally, results are processed to calculate OER and Y and the effectiveness

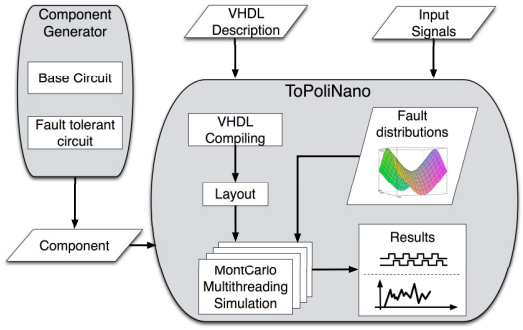

Figure 5: ToPoliNano and FaTToR flow chart

of the algorithm is tested by comparing Y and OER for the FaTToR optimized circuits w.r.t. the non optimized ones.

V. A fault tolerance topology-based optimizer: FaTToR

The permutation rules which FaTToR relies on are: r1) permuting two rows is allowed without any restriction; r2) permuting two columns is allowed provided that they belong to the same plane. Generally a matrix $A \in \mathbb{R}^{m,n}$ can be permuted by means of *permutation matrices*, i.e., a class of square matrices $q \times q$ that we define $\in \mathbb{P}^q$, whose entries are 0 or 1, and the sum of each row and each column is equal to 1. An example of with a matrix $\in \mathbb{P}^3$ is provided below:

$$\begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix}\begin{bmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\end{bmatrix}=\begin{bmatrix}5&6&7&8\\9&10&11&12\\1&2&3&4\end{bmatrix}$$

Defining X the rows permutation matrix, Y (W) the columns permutation matrix for the NAND1 (NAND2) plane, the *penalty matrix* $P \in \mathbb{R}^{r,c}$ of the circuit is in Eq. (1):

$$P(X,Y,W) = \left[P_1(X,Y) \;\middle|\; P_2(X,W)\right] \tag{1}$$

where P_1 (P_2) refers to the penalty matrix of each TR at ij, on the NAND1 (NAND2) plane (i.e. a penalty located where no TRs are present is set to zero). FaTToR minimizes penalties globally (min-avg mode), locally (min-max mode) or both, by means of an user-defined weight γ. The objective function of FaTToR, with optimal value p^*, aims to reduce the penalty by finding the best permutation matrices X, Y and W:

$$\begin{aligned} p_{\text{avg}}(X,Y,W) &= \frac{1}{\#_{\text{dev.}}}\left(\sum_{i,j} p_{i,j}(X,Y,W)\right)\\ p_{\max}(X,Y,W) &= \max_{i,j}\left(p_{i,j}(X,Y,W)\right)\\ p^* &= \min_{X,Y,W}\left(p_{\text{avg}}(X,Y,W)(1-\gamma) + p_{\max}(X,Y,W)\,\gamma\right)\end{aligned} \tag{2}$$

The way in which penalties are defined (see the following description) makes the model NP-hard (either non linear and non continuous). Then the optimization process in FaTToR is heuristic and based on a relaxed versions of $P(X,Y,W)$. Instead of solving the non-linear model, it iteratively executes a sequence of two MILP (Mixed Integer Linear Programming) problems, each one performing rows and columns optimization. For this reason, in the first step X is variable whereas Y and W are constant, and vice-versa in the second step.

Penalties structures: OPTOER and OPTY optimizations. FaTToR allows fault tolerance improvement by optimizing output error rate (herein named OPTOER version) or yield (named OPTY version). The algorithm differs only for the penalties definition. Since OPTY is a special (and simpler) case of OPTOER, in the following we illustrate the penalties matrices for OPTOER, focusing on the OPTY only at the end. In general, the matrices P_1 and P_2, which are function of the permutation matrices, contain: I) The topological description of the circuit that needs to be permuted, represented by means of *circuit matrices*; II) The matrices including fault distributions (not to be permuted), represented by the local fault probability related to a given kind of fault for each sub-tile.

In a NAND-NAND architecture we can conceive the circuit as a set of NAND gates (a row in the NAND1 plane, or to a column in the NAND2 plane). On each NAND, an input is *observable* from the output only if the other inputs belonging to the same NAND gate are set to 1. We use here *circuit matrices*, a set of matrices representing the probability a given TR placed in ij has to be observable by the output (and NAND1 outputs are NAND2 inputs). Then we define two couples of matrices $M_{\text{plane}}^{\text{logic state}}$ corresponding to the probability of having a device in $m_{\text{plane}ij}$ observable with a given logic state (H,L) on its input. In case we do not have any TR at a certain position, such probability is set to 0. For the example in Fig. 3.A with equiprobable inputs the representation is:

$$M_1^{\text{L}} = M_1^{\text{H}} = \begin{bmatrix}0.25&0&0.25&0\\0&0.5&0&0\\0&0&0&0.5\end{bmatrix} \quad M_2^{\text{L}} = \begin{bmatrix}0.25&0\\0&0.25\\0&0.25\end{bmatrix} \quad M_2^{\text{H}} = \begin{bmatrix}0.75&0\\0&0.25\\0&0.25\end{bmatrix}$$

Besides the circuit matrices, we define the following matrices each representing a given fault type: H (V) and R_{on} (R_{off}). They represent, respectively, the probability of having a horizontal (vertical) broken NW and a stuck-on (stuck-open) device at position ij. Since penalties refer to TRs, H and V are redefined in order to make fault probabilities relative to TRs positions. So, with this transformation we refer to the probability of having a fault with respect to a TR at ij, instead of dealing with local probabilities of broken NWs. Such transformation can be done due to uncorrelated local probabilities. For the horizontal NWs we define D_1^{h} and D_2^{hLeft} (D_2^{hRight}), that are, respectively, the probability of having a broken element along all the path at row i in the NAND1 plane, and the probability of having a broken element in the NAND2 plane from the left (right) edge to the position ij. D_1^{v}, represents the probability of having a vertical broken NW in the NAND1 plane from the input to the location ij, and D_2^{v}, which represents the probability of having a vertical NW broken along all the path at column j in the NAND2 plane.

Starting from these matrices, defect rates $K_{\text{plane}}^{\text{logic state}}$ are computed. They represent the cases in which an error is propagated towards the output for a given logic state in a given plane (where $\circ$ is the Hadamard product), and where $R_{\text{plane}}^{\text{ok}} = 1 - R_{\text{plane}}^{\text{on}} - R_{\text{plane}}^{\text{off}}$ represents the probability of having TRs with no faults.

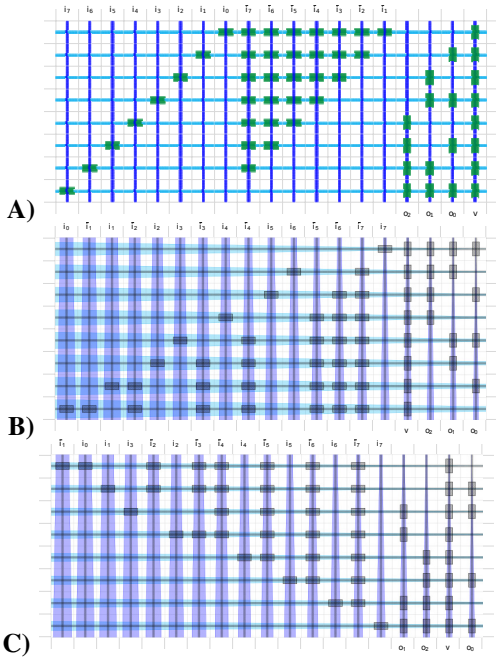

Figure 6: 8-bit priority encoder: A) standard version B) optimized with OPT-OER C) optimized with OPT-Y algorithm

$$
\begin{aligned}
K_1^{\mathrm{L}} &= R_1^{\mathrm{on}} \circ \left(1 - D_1^{\mathrm{h}}\right) \\
K_1^{\mathrm{H}} &= R_1^{\mathrm{on}} \circ D_1^{\mathrm{h}} + R_1^{\mathrm{off}} + R_1^{\mathrm{ok}} \circ \left(D_1^{\mathrm{v}} \cup D_1^{\mathrm{h}}\right) \\
K_2^{\mathrm{L}} &= \left(R_2^{\mathrm{on}} + R_2^{\mathrm{ok}} \circ D_2^{\mathrm{hLeft}} \circ \left(1 - D_2^{\mathrm{hRight}}\right)\right) \circ \left(1 - D_2^{\mathrm{v}}\right) \\
K_2^{\mathrm{H}} &= R_2^{\mathrm{on}} \circ D_2^{\mathrm{v}} + R_2^{\mathrm{off}} + R_2^{\mathrm{ok}} \circ \left(D_2^{\mathrm{v}} \cup D_2^{\mathrm{hRight}}\right)
\end{aligned}
\tag{3}
$$

By looking at the matrices K they are in general computed considering three cases, i.e., when the TR at ij is stuck-on, stuck-open or working, and for each of these cases we analyze the probability of having error propagation. Also, since we do not know if a TR is placed at a given position ij or not (because it depends on the optimization result), matrices K are calculated by assuming that in every entry ij a TR is placed. For instance, in NAND1 the probability of having an error with a logic zero in correspondence of a possible presence of a TR in ij is given by the probability of having either an error-free horizontal line $1 - D_1^{\mathrm{h}}$ and the TR in stuck-on. The complete representation of penalty matrices, including the circuit matrices M and the defect rates K, is in the following:

$$
\begin{aligned}
P_1^{\mathrm{L}} &= \left(XM_1^{\mathrm{L}}Y\right) \circ K_1^{\mathrm{L}} \quad P_1^{\mathrm{H}} = \left(XM_1^{\mathrm{H}}Y\right) \circ K_1^{\mathrm{H}} \\
P_2^{\mathrm{L}} &= \left(XM_2^{\mathrm{L}}W\right) \circ K_2^{\mathrm{L}} \quad P_2^{\mathrm{H}} = \left(XM_2^{\mathrm{H}}W\right) \circ K_2^{\mathrm{H}}
\end{aligned}
\tag{4}
$$

where $P_1 = P_1^{\mathrm{L}} + P_1^{\mathrm{H}}$, $P_2 = P_2^{\mathrm{L}} + P_2^{\mathrm{H}}$.

For what concerns OPTY, the algorithm does not need to consider observability, and so it does not make any distinction among the TRs fault masking capability. The reason behind this is that targeting Y implies that every input combination is correctly propagated, independently from the probability of TRs to be maskable for a certain set of input patterns. For this reason, the circuit matrices are intentionally rounded up to 1 for each entry greater than 0. Another difference w.r.t. OPTOER is that, in absence of redundancy, there is no sense to consider fault distribution on horizontal NWs (in both planes)

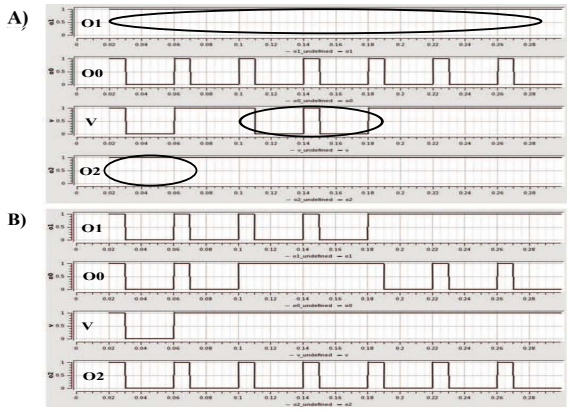

Figure 7: Output waveforms under faults injection of a 8-bit priority encoder with: A) no optimizations B) optimazions

and on vertical NWs in the NAND2 plane, because they are in any case always filled by TRs, which all contribute to the yield achievement. So, in order for the algorithm to ignore the relevance of such fault distributions, the matrices D_1^{h}, D_2^{hLeft}, D_2^{hRight} and D_2^{v} are dropped to 0.

VI. Results

As described in the previous paragraph, the algorithm, according to a given fault distribution modifies the columns and rows disposition in order to place TR in a low failure rate area. For example we applied FaTToR on a 8-bit priority encoder circuit, (Fig. 6.A) using a corner linear distribution to the base circuit. The TR disposition optimized for minimizing OER (OPTOER) is shown in Fig. 6.B, where the faults distribution is in evidence with thicker lines where the probability of faults is bigger. Fig. 6.C, shows the case optimized for Yield (OPTY). The algorithm clearly moves away the devices from the more defective corner (south-west in this example) in order to assure to have the maximum number of circuits completely working. In case of OPTOER the optimization mimimizes the probability to have an error at the output, even in case of defects that have some effects on the circuit behavior in some cases.

Exploiting ToPoliNano, we perform the MonteCarlo multithreading simulation in both optimized and non-optimized circuits injecting faults according to different distributions. Fig. 7.A shows a portion of the resulting simulation waveforms obtained with the non optimized case where some errors due to faults are highlighted; in Fig. 7.B the simulation of optimized version is error free. In this case the output error rate dropped from 3.49 % to 1.84 %.

We report here results obtained for a 4-bit and for a 8-bit priority encoder (PE). Fig. 8 shows results as percentage difference between the circuit optimized by FaTToR w.r.t. the standard non optimized version. Pictures A and B refer to the 4-bit circuit, while the 8-bit one are in pictures C and D. A and C show the improvement in terms of Y while B and D the OER change. We tested the method by injecting faults according to different distribution on NWs and devices (in the x-axis). Among all the set we tested, for sake of brevity, we show here a selection to show the importance of this variable in

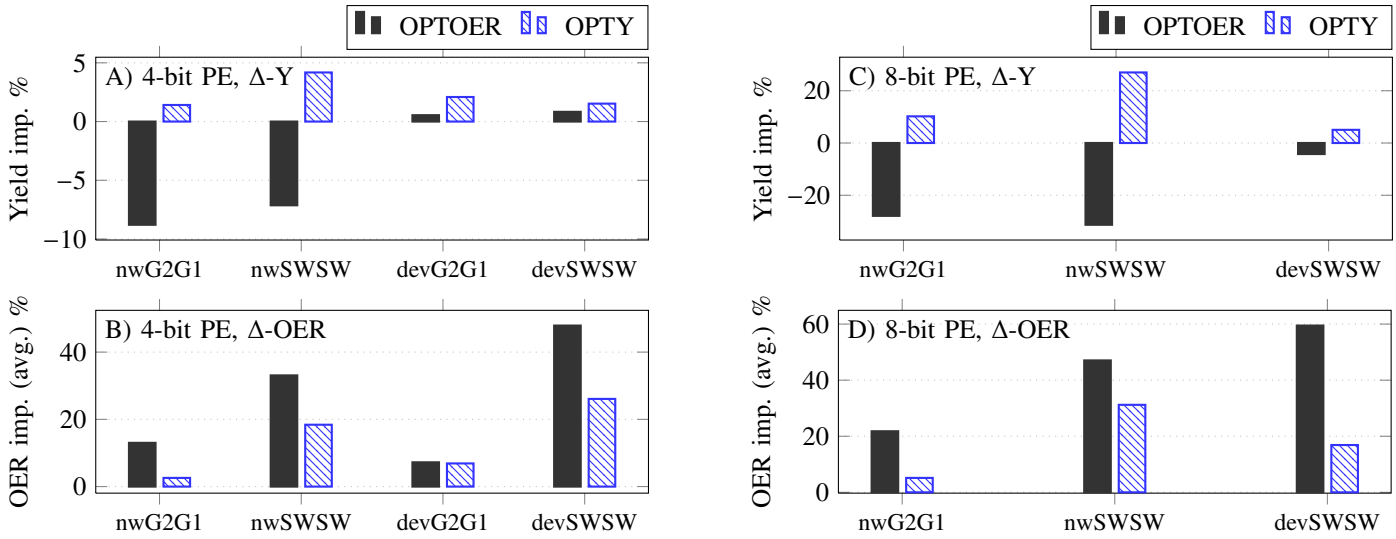

Figure 8: Percentage improvement of the circuit optimized by FaTToR w.r.t. the non optimized one with different fault distributions. For each case both the effects of OPTOER and OPTY algorithms are reported. A) Y for 4-bit priority encoder circuit (PE). B) OER for 4-bit PE. C) Y for 8-bit PE. D) OER for 8-bit PE. Distributions: nw=faults on NWs; dev=faults on devices; G2G1=G2 distribution on H NWs(dev) and G1 distribution on V NWs(dev); SWSW=south west corner linear distribution.

the final outcome. The maximum injected fault probability in these selected cases is 2%. We used for these results $\gamma = 0.8$. Each case is tested both for OPTOER (OER minimization) and OPTY (Yield maximization) algorithms. As expected the OPTY algorithm (patterned histograms) gives a gain in terms of Y, more evident for the 8-bit circuit (because bigger and with more degrees of freedom at the optimization phase) and more clear in the corner case faults distribution. This suggests that in presence of NWs or devices faults more concentrated in one corner (due to NWs manipulation for example) the process could be directed toward privileging the SWSW corner as more defective. If OPTOER is the optimization type chosen, then Y is penalized, while the improvement in terms of OER rises to very high percentages, especially in the case of faults on devices. If on the one hand, one could summarize that optimizing for Yield gives always a good result both in terms of Y and in terms of OER, still, depending on how the circuit is used and which other techniques can be exploited to reduce the impact of defects (e.g. redundancy), optimizing for OER could give very interesting outcomes. Finally, for Random distributions on both NWs and devices (reported as in previous works is the only distribution considered), we obtain a Y improvement of 15.6% with OPTY and of 3.5% with OPTOER; the OER improvement is 13.3% and 21.11% in the two cases, respectively.

VII. Conclusion

Our first contribution, the fault tolerance optimization algorithm for nanoarray based circuits, shows excellent results both in terms of Yield and Output Error Rate. Our second contribution, the tool for nanoarray circuits design and accurate simulation enriched with a multithreading MonteCarlo fault injection engine, allows to test both optimized and non optimized circuits against faults obtained from technological processes and assuming any desired distribution with topological information. We are not aware of previus contributions able to include both in the optimization engine and in the MonteCarlo based simulator 1) precise localization awareness of faults with any kind of distribution, 2) double yield and output error rate fault tolerance objectives, 3) detailed simulation at switch level, 4) multithreading simulation engine potentially ready for huge circuits analysis.

References

[1] A. DeHon. Array-based architecture for fet-based, nanoscale electronics. *Nanotechnology, IEEE Transactions on*, 2(1):23–32, 2003.

[2] P. E. Gaillardon, M. Haykel Ben-Jamaa, F. Clermidy1, and I. OConnor. Evaluation of a crossbar multiplexer in a lithography-based nanowire technology. *Circuits and Systems (ISCAS), IEEE International Symposium*, pages 2930–2933, 2011.

[3] S. Frache, D. Chiabrando, M. Graziano, E. Enrico, L. Boarino, and M. Zamboni. Silicon nanoarray circuits design, modeling, simulation and fabrication. In *Nanotechnology (IEEE-NANO), 2012 12th IEEE Conference on*, pages 1–5, 2012.

[4] S. Frache, D. Chiabrando, M. Graziano, M.Vacca, L.Boarino, and M. Zamboni. Enabling design and simulation of massive parallel nanoarchitectures. *J. of Par. and Distr. Computing*, In press, 2013.

[5] Md Muwyid U. Khan, P. Narayanan, P. Joshi, P. Panchapakeshan, and C. A. Moritz. Fasttrack: Toward nanoscale fault masking with high performance. *Proc. of the IEEE*, 11(4), 2012.

[6] P. Das Kanungo, H. Schmid, M. T Björk, L. M Gignac, C. Breslin, J. Bruley, C. D Bessire, and H. Riel. Selective area growth of iiiv nanowires and their heterostructures on silicon in a nanotube template: towards monolithic integration of nano-devices. *Nanotechnology*, 2013.

[7] W. Rao, A. Orailoglu, and R. Karri. Fault tolerant approaches to nanoelectronic programmable logic arrays. *Dependable Systems and Networks*, pages 216–224, 2007.

[8] F. Angiolini, M. Haykel Ben Jamaa, D. Atienza, L. Benini, and G. De Micheli. Improving the fault tolerance of nanometric pla designs. pages 570–575, 2007.

[9] S. Frache, D. Chiabrando, M. Graziano, F. Riente, G. Turvani, and M. Zamboni. Topolinano: Nanoarchitectures design made real. *IEEE/ACM Int. Symp. on Nanoscale Arch.*, pages 160–167, 2012.

[10] S. Frache, M. Graziano, and M. Zamboni. A flexible simulation methodology and tool for nanoarray-based architectures. *International Conference on Computer Design*, pages 60–67, 2010.