Logic Synthesis and Testing Techniques for Switching Nano-Crossbar Arrays

Dan Alexandrescu[a], Mustafa Altun[b], Lorena Anghel[c], Anna Bernasconi[d], Valentina Ciriani[e], Luca Frontini[e], Mehdi Tahoori[f]

[a]*IROC Technologies Grenoble, France, dan.alexandrescu@iroctech.com*
[b]*Dept. of Electronics and Communication Engineering, Istanbul Technical University, Turkey, altunmus@itu.edu.tr*
[c]*TIMA laboratory, Grenoble-Alpes University, France, lorena.anghel@imag.fr*
[d]*Dipartimento di Informatica, Università di Pisa, Italy, anna.bernasconi@unipi.it*
[e]*Dipartimento di Informatica, Università degli Studi di Milano, Italy, {valentina.ciriani, luca.frontini}@unimi.it*
[f]*Karlsruhe Institute of Technology, Karlsruhe, Germany, tahoori@ira.uka.de*

Abstract

Beyond CMOS, new technologies are emerging to extend electronic systems with features unavailable to silicon-based devices. Emerging technologies provide new logic and interconnection structures for computation, storage and communication that may require new design paradigms, and therefore trigger the development of a new generation of design automation tools. In the last decade, several emerging technologies have been proposed and the time has come for studying new ad-hoc techniques and tools for logic synthesis, physical design and testing. The main goal of this project is developing a complete synthesis and optimization methodology for switching nano-crossbar arrays that leads to the design and construction of an emerging nanocomputer. New models for diode, FET, and four-terminal switch based nanoarrays are developed. The proposed methodology implements logic, arithmetic, and memory elements by considering performance parameters such as area, delay, power dissipation, and reliability. With combination of logic, arithmetic, and memory elements a synchronous state machine (SSM), representation of a computer, is realized. The proposed methodology targets variety of emerging technologies including nanowire/nanotube crossbar arrays, magnetic switch-based structures, and crossbar memories. The results of this project will be a foundation of nano-crossbar based circuit design techniques and greatly contribute to the construction of emerging computers beyond CMOS. The topic of this project can be considered under the research area of "Emerging Computing Mod-

Preprint submitted to Embedded Hardware Design (Microprocessors and Microsystems)August 23, 2017

els" or "Computational Nanoelectronics", more specifically the design, modeling, and simulation of new nanoscale switches beyond CMOS.

Keywords: Nano-Crossbar, Emerging Computing Model, Computational Nanoelectronics

1. Introduction

CMOS transistor dimensions have been shrinking for decades in an almost regular manner. Nowadays this trend has reached a critical point and it is widely accepted that the trend will end in a decade [1]. Even Gordon Moore, who made the most influential prediction in 1965 about CMOS size shrinking (Moore Law), accepted that his prediction will lose it validity in near future [2]. At this point, research is shifting to novel forms of nanotechnologies including molecular-scale selfassembled systems [3, 4]. Unlike conventional CMOS that can be patterned in complex ways with lithography, self-assembled nanoscale systems generally consist of regular structures. Logical functions and memory elements are achieved with arrays of crossbar-type switches. This project targets this type of switching crossbars by using models based on diodes, FETs, and four-terminal switches [5, 6, 7], as illustrated in Figure 1. In particular, Figure 2 shows three implementations of a specific Boolean function with these models. Among these models a model based on four-terminal switches deserves a special mention.

A four-terminal switch is specifically developed for crosspoints of nanoarrays; note that each crosspoint has four neigbour crosspoints. The four terminals of the switch are all either mutually connected (ON) or disconnected (OFF). If a controlling literal takes the value of 1, the switch is ON; otherwise, it is OFF. On the other hand, diode and FET are conventional two-terminal switch based devices, i.e., their two terminals are either connected (ON) or disconnected (OFF).

In this paper we describe the Marie Skłodowska-Curie grant agreement No 691178 (European Union's Horizon 2020 research and innovation programme). Section 2 provides a general overview of the project. The following sections summarize the obtained results and the main objectives identified in the first part of the project. In particular, Section 3 investigates logic synthesis and area optimization techniques for diode, FET, and four-terminal switch based nanoarrays by comparing array sizes needed to implement given Boolean functions. Section 4 shows a new decomposition method for the four-terminal switch based model. Section 5 briefly discusses reconfiguration and redundancy approaches needed to face temporal and spatial variations of component electrical properties and hard faults.

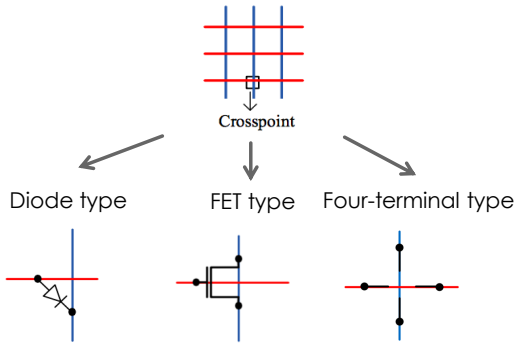


Figure 1: Different models of switching crossbars.

Section 6 introduces capacitor-resistor models for diode, FET, and four-terminal switch based nano-crossbar arrays that are to be used for power and delay analysis. Section 7 concludes the paper.

This article is an extended version of the conference report in [8].

2. Overview of the Project

2.1. Research Objectives

The main objective of this project is developing a complete synthesis methodology for nanoscale switching crossbars that leads to the design and construction of an emerging computer. To achieve this objective, we follow a roadmap, illustrated in Figure 3, with sub-objectives listed below.

1. Finding optimal crossbar sizes, modeling, and optimization: Fundamentally, all building parts of a computer use Boolean functions for their operations. Therefore, implementing Boolean functions with optimal sizes significantly advances us toward achieving our main goal. Besides size, other parameters such as power consumption, delay, and reliability have been considered by developing related models. This part of the project has been successfully concluded developing a systematic approach for synthesis and performance optimization (see Sections 3 and 4).
2. Implementing the elements by considering reliability, area, delay, and power dissipation of the crossbars: We implement arithmetic elements such as adders and multipliers, and memory elements such as flip-flops and registers as building blocks. We also perform optimization for circuit performance

parameters using the specifics of applicable technologies. Our methodology will considers all circuit performance parameters. This can allow us to compare our results with those of CMOS circuits in a realistic and comprehensive manner. First, the trade-offs between parameters will be investigated. Then, the specifics of applicable technologies for the performance parameters will be determined. Finally, a comprehensive optimization software package for the concurrent physical and logical design of applicable technologies will be revealed.

3. Realizing a synchronous state machine (SSM): We will implement a SSM with a programmable multi-array (tile) architecture such that each crosspoint in arrays corresponds to a programmable four-terminal switch. We intend not to use any individual transistor and switch that causes interconnection problems and significantly worsens the density. Another potential problem is signal quality degradation that could upper-bound the number of separate arrays/crossbars in the architecture. Conventionally, this problem is solved by adding simple restoration circuits at the outputs of each circuit block (in our case a crossbar). Unfortunately, this solution would be quite costly for nano-crossbars since they are compact and hard-to-manipulate structures. Another solution for the signal degradation problem especially for memories is using accurate sense amplifiers that should be built-in a crossbar memory using the same technology. We have presented our preliminary results using memristor memory arrays [9]. Additionally, as a contingency plan, we will design a single clocked programmable array that synchronously restores the signals.

2.2. State of the Art

Researchers have been interested in models of regular arrays since the seminal paper of Akers in 1971 [10]. In recent years, this interest sees a dramatic spike with the rise of emerging technologies based on regular arrays of switches [11]. Such technologies have apparent advantages over conventional CMOS technologies, such as high density (small area and delay) and easy manufacturability due to self-assembly [4]. Our models target these emerging technologies including nanowire/nanotube crossbar arrays, magnetic switch-based structures, and crossbar memories [12, 13, 14, 15]. Using diode, FET, and four-terminal switch based models we aim to achieve a synchronous state machine at the end of this project that would be the first in the literature.

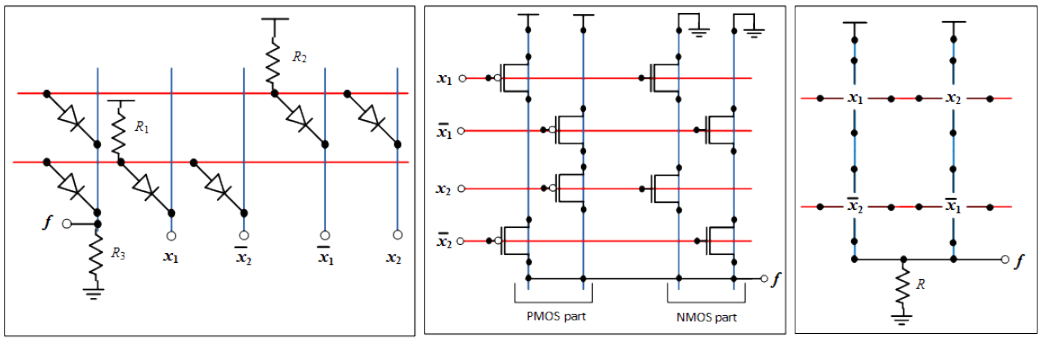


Figure 2: Implementation of a Boolean function $f = x_1\bar{x}_2 + \bar{x}_1 x_2$ with a) diode, b) FET, and c) 4-terminal models.

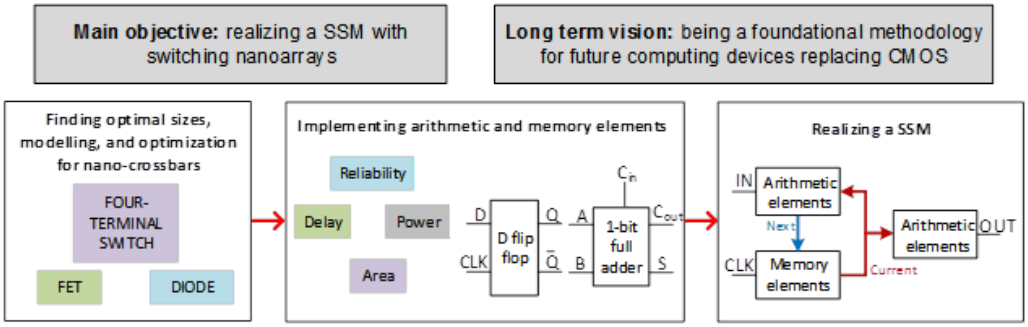


Figure 3: Project overview.

Type	Array Size Formulas (Optimal)
Diode	(number of products in f) x ("number of literals in f" + 1)
FET	(number of literals in f) x ("number of products in f " + "number of products in f^D")

Figure 4: Array size formulas for diode and FET based implementations.

Type	Array Size Formula (Non-optimal)
Four-terminal	(number of products in f) x (number of products in f^D)

Figure 5: Array size formula for four-terminal switch based implementation.

3. Logic synthesis with area optimization for diode, FET, and four-terminal switch based nanoarrays

In this section, we survey logic synthesis techniques for diode, FET, and four-terminal switch based nanoarrays [21, 22, 23, 24]. We present experimental results on standard benchmark circuits to compare array sizes needed to implement given Boolean functions. For diode and FET based nanoarrays, Boolean functions are implemented by using conventional techniques that are diode-resistor logic and CMOS logic [21]. This implies an important constraint regarding nanoarray structures. Boolean functions should be implemented in their sum-of-products (SOP) forms; other forms such as factored or BDD (Binary Decision Diagram) cannot be used since they require manipulation/wiring of switches that is not applicable for self-assembled nanoarrays.

Array sizes for diode and FET based nanoarrays: Given a target Boolean function f, we derive formulas of the array sizes. This is shown in Figure 4. For diode based implementations, each product of f requires a row (horizontal line), and each literal of f requires a column (vertical line) in an array. Additionally, one extra column is needed to obtain the output. For FET based implementations, each product of f and its dual, f^D, requires a column, and each literal of f requires a row in an array. As an example, consider a target function $f = x_1\overline{x_2}+\overline{x_1}x_2$ having 4 literals and 2 products; $f^D = x_1x_2 + \overline{x_1}\ \overline{x_2}$ has 2 products. This results in array sizes of 2×5 and 4×4 for diode and FET based implementations, respectively. Note that both formulas, for diode and FET, always result in optimal array sizes; no further reduction is possible.

Benchmark	CMOS	Diode	4-Terminal	Optimal 4-Terminal
Alu 0	30	18	6	**6**
Alu 1	30	18	6	**6**
Alu 2	30	18	6	**6**
Alu 3	30	18	6	**6**
B12 0	80	32	24	**12**
B12 1	120	70	35	**16**
B12 3	30	20	8	**8**
B12 4	42	28	8	**8**
B12 6	132	77	35	**18**
B12 7	110	66	24	**18**
B12 8	90	70	14	**14**
C17 0	36	18	9	**6**
C17 1	30	20	8	**8**
Clpl 0	64	32	16	**12**
Clpl 1	36	18	9	**9**
Clpl 2	16	8	4	**4**
Clpl 3	144	72	36	**18**
Clpl 4	100	50	25	**15**
Dc1 1	25	10	6	**6**

Benchmark	CMOS	Diode	4-Terminal	Optimal 4-Terminal
Dc1 2	72	36	16	**12**
Dc1 5	35	15	12	**6**
Dc1 6	36	18	9	**6**
Ex5 31	156	104	32	**24**
Ex5 33	110	77	21	**21**
Ex5 46	81	54	18	**18**
Ex5 49	72	54	12	**12**
Ex5 50	81	63	14	**14**
Ex5 61	64	48	12	**12**
Ex5 62	49	35	10	**10**
Misex1 1	48	16	8	**8**
Misex1 2	132	55	35	**15**
Misex1 3	156	60	40	**24**
Misex1 4	121	44	28	**16**
Misex1 5	90	45	25	**15**
Misex1 6	143	66	42	**18**
Misex1 7	81	36	20	**15**
Mp2d 4	345	75	90	**24**
Newtag	108	72	32	**18**

Figure 6: Array sizes of three different nano-crossbar based logic families [21, 23].

Four-terminal switch based implementation considers each crosspoint of an array as a four-terminal switch [7]. Four terminals of the switch are all either mutually connected (ON-logic 1 applied) or disconnected (OFF-logic 0 applied). Boolean functions are implemented with top-to-bottom paths in an array by taking the sum (OR) of the product (AND) of literals along each path. This makes Boolean functions implemented in their sum-of-products (SOP) forms.

Array size for four-terminal switch based nanoarrays: Given a target Boolean function f, the array size formula can be derived by considering that each product of f and its dual, f^D, require an array column and an array row, respectively. This is shown in Figure 5. As an example, consider a target function $f = x_1\overline{x_2} + \overline{x_1}x_2$ and $f^D = x_1x_2 + \overline{x_1}\ \overline{x_2}$ both having 2 products. This results in an array size of 2×2.

Examining the array size formulas in Figure 4 and Figure 5, we see that while the formulas in Figure 4 always result in optimal sizes, the sizes derived from the formula in Figure 5, that is, for four-terminal switch based arrays, are not necessarily optimal [7]. For example, consider a target function $f = x_1x_2x_3 + x_4x_5x_6$. It has only two products, but its dual has $3^2 = 9$ products. With using the formula in Figure 5, an array with 9 rows and 2 columns would be required. This is not an optimal solution. By placing 0's between two columns implementing the two products, one can implement the function with an array having 3 rows and 3

columns.

In the following part we present an algorithm that finds an optimal size implementation of any given target Boolean function. Finding whether a certain array with assigned literals to its switches implements a target function is the main problem in finding optimal sizes. This problem requires to check if each assignment of 0's and 1's to the switches, corresponding to a row of the target function's truth table, results in logic 1 (a top-to-bottom path of 1's exists). To check this we have to enumerate all top-to-bottom paths; the size of this task grows exponentially with the array size. This is a general statement that holds also for our algorithm described below. The presented algorithm is a preliminary result of an ongoing study. Although its performance for runtime and relatively for time complexity is not quite satisfactory, it gives us an important inference: four-terminal switch based arrays overwhelm those based on two-terminal switches regarding the array size.

Our simple brute force algorithm finds optimal array sizes to implement given target Boolean functions with arrays of four-terminal switches in three steps:

1. Obtain irredundant sum-of-products (ISOP) expressions of a given-target function f_T and its dual ${f_T}^D$. Determine the upper bound (UB) on the array size using the formula in Figure 5. Obtain the lower bound (LB) values directly from the lower bound table proposed in [7].
2. List arrays with dimensions (R×C) between UB and LB and sort them regarding array sizes in ascending order. While ordering, if the array sizes are the same – for example 3×5 and 5×3 – then first take the array having smaller number of rows. Suppose that there are a total of N different arrays $(R\times C)_1$...$(R\times C)_i$...$(R\times C)_N$ in the list. For Step 3, start with $i = 1$ ($1 \leq i \leq N$).
3. If the array can implement f_T then the corresponding size $(R\times C)_i$ is the optimal size; otherwise increase i by one and repeat this step again.

Complexity analysis of our algorithm is as follows. Suppose that a target function f_T and its dual ${f_T}^D$ have n and m number of products, respectively. Also suppose that f_T has k variables. In Step 2, the total number of arrays in the list is upper bounded by $n \times m$ by using the formula in Figure 5. In step 3, the worst-case time complexity of checking if an array implements f_T is $O(2^k \times n \times m)$. As a result, the worst-case complexity of the algorithm is $O(2^k \times n^2 \times m^2)$. Of course, this expression is for the worst-case; for most of the cases a tiny percentage of 2^k truth table rows is used. In practical terms, each of the results in Figure 4 is obtained under a limit of 2 hours. Another perspective is hardware sharing.

Especially if target functions with same variable sets (same truth table inputs) are given, hardware sharing is perfectly applicable in Step 3.

Simulation results: In Figure 6, we report synthesis results for standard benchmark circuits. We treat each output of a benchmark circuit as a separate target function. The number of products for each target function f_T and its dual ${f_T}^D$ are obtained through sum-of-products minimization using the program Espresso. The array size values for "Diode", "CMOS", and "4-terminal" are calculated by using the formulas in Figure 4 and Figure 5. The array size values for "Optimal 4-terminal" are obtained using the presented optimization algorithm. Examining the numbers in Figure 6, we always see the same sequence from the worst to the best result as "CMOS", "Diode", "4-terminal", and "Optimal 4-terminal". This proves that models based on four-terminal switches overwhelm those based on two-terminal switches regarding the array size. Further, the numbers obtained by our optimal synthesis method [21, 23] compare very favorably to the numbers obtained by the previous method [7].

4. Lattices and decomposition methods

An important issue in logic synthesis is to produce efficient implementations of single or multi-output Boolean functions. The standard CMOS synthesis is typically performed with Sum of Products (or SOP) minimization procedures, leading to two-level circuits. Two-level logic minimization has been one of the most studied problems in logic synthesis. Big efforts have been done to obtain efficient Sum of Products minimization procedures and their related CAD tools. Whereas research and synthesis tools for SOP minimization are quite mature and consolidated, the study of networks for new technologies is still at early stages. In particular, four-terminal switching lattice synthesis has been just studied in some recent works [7, 21, 23, 22, 24, 25].

In this section we briefly present some new results obtained within this project, that show how the cost of implementing a four-terminal switching lattice could be mitigated by exploiting Boolean function decomposition techniques. The basic idea of this approach is to first decompose a function into some subfunctions, according to a given functional decomposition scheme, and then to implement the decomposed blocks with separate lattices, or physically separated regions in a single lattice. Since the decomposed blocks usually correspond to functions depending on fewer variables and/or with a smaller on-set, their synthesis should be more feasible and should produce lattice implementations of smaller size. In the framework of switching lattice synthesis, where the available minimization

tools are not yet as developed and mature as those available for CMOS technology, reducing the synthesis of a target Boolean function to the synthesis of smaller functions could represent a very beneficial approach [26, 27]. As a first work for this project, we have focused on the particular decomposition method that gives rise to the bounded-level logic networks called *P-circuits* [27, 28, 29]. P-circuits are extended forms of Shannon cofactoring, where the expansion is with respect to an orthogonal basis $\overline{x}_i \oplus p$ (i.e., $x_i = p$), and $x_i \oplus p$ (i.e., $x_i \neq p$), where p is a function defined over all variables except for a critical variable x_i (e.g., the variable with more switching activity or with higher delay that should be projected away from the rest of the circuit). They can be defined as follows:

$$\text{P-circuit}(f) = (\overline{x}_i \oplus p)\, f^{=} + (x_i \oplus p)\, f^{\neq} + f^{I}$$

where I is the intersection of the projections of f onto the two sets $x_i = p$ and $x_i \neq p$, and

1. $(f|_{x_i=p} \setminus I) \subseteq f^{=} \subseteq f|_{x_i=p}$
2. $(f|_{x_i\neq p} \setminus I) \subseteq f^{\neq} \subseteq f|_{x_i\neq p}$
3. $\emptyset \subseteq f^{I} \subseteq I$.

This definition can be easily generalized to incompletely specified Boolean functions. Thus, the synthesis idea of P-circuits is to construct a network for f by appropriately choosing the sets $f^{=}$, $f^{\neq}$, and f^{I} as building blocks.

The same idea can be exploited in the switching lattice framework: the subfunctions $f^{=}$, $f^{\neq}$, and f^{I} depend on $n-1$ variables instead of n, they have a smaller on-set than f, and their lattice synthesis should produce lattices of reduced area. Therefore, the overall lattice for f derived composing minimal lattices for $f^{=}$, $f^{\neq}$, and f^{I}, could be smaller than the one derived for f without exploiting its P-circuits decomposition. This expectation has been confirmed by a set of experimental results, (see [30]), where the utility of the decomposition-based approach has been evaluated applying the two synthesis methods presented in [7] and in [25]. These results demonstrate that lattice synthesis benefits from this type of Boolean decomposition, yielding smaller circuits with an affordable computation time (even less in some cases). Indeed, in 30% of the analyzed cases the synthesis of switching lattices based on the P-circuit decomposition of the logic function allows to obtain a more compact area in the final resulting lattice, with an average gain of at least 20%.

We can address more complex types of decompositions, both within the class of P-circuits (with more expressive projection functions p) and beyond. In particular, we also consider a decomposition scheme that can be applied to lattice

synthesis of a special class of regular Boolean functions called *D-reducible* functions. D-reducible functions [31] are functions whose points are completely contained in an affine space strictly smaller than the whole Boolean cube $\{0,1\}^n$. A D-reducible function f can be written as $f = \chi_A \cdot f_A$, where A is the smallest affine space that contains the on-set of f, χ_A is the characteristic function of A, and f_A is the projection of f onto A. Notice that f and f_A have the same number of points, but these are now compacted in a smaller space.

The D-reducibility of a function f can be exploited in the lattice synthesis process: the idea is to independently find lattice implementations for the projection f_A and for the characteristic function χ_A of A, and then to compose them in order to construct the lattice for f. Since the two functions f_A and χ_A depend on fewer variables than the original function f, their synthesis is more feasible and can produce lattice implementations of smaller area.

The area of the overall lattice can be further reduced exploiting the peculiar structure of the characteristic function χ_A, that represents the minterms of an affine subspace of $\{0,1\}^n$. To this aim, we can synthesize compact lattices of affine spaces whose characteristic function is represented by the product of single literals and XOR factors of two literals. The experimental results validate the approach, demonstrating that the lattice synthesis based on the D-reducibility property allows to obtain a more compact area in 15% of the considered cases, with an average gain of about 24%. Moreover, the experimental results show that we can reduce the synthesis time of the lattices of about 50%, with respect to the time needed for the synthesis of plain lattices [32].

5. Built-in Variation, Defect, and Fault Tolerance

One of the main focuses of this project is development of defect, variation and fault tolerant techniques in the presence of high defect densities and extreme parametric variations, particularly for crossbar array nano-architectures. To tolerate high defect rates and variations, our revolutionary approach is to integrate *defect tolerance* to improve the manufacturing yield (for fabrication defects), *fault tolerance* to ensure the lifetime reliability (for errors during normal operation), and *variation tolerance* to ensure the predictability and performance (for parametric variations), in the design methodologies for future nanotechnologies. *Adaptive* and *built-in* defect, variation and fault tolerant design flows, fundamentally different from conventional approaches, are proposed in which the objective is to ensure high manufacturing yield and runtime reliability of the circuit at extremely low costs. We plan to exploit the opportunities created by this nanotechnology

such as reprogrammability and abundance of programmable resources to provide defect, variation and fault tolerance.

5.1. Built-in Self-testing (BIST) and Self-diagnosis (BISD)

New nanomanufacturing process techniques and steps for nano-scale devices that are conceptually different from conventional lithography-based process techniques may result in new failure mechanisms not completely understood today. Therefore, test techniques for such systems should not be restricted to a particular fault model, but be very comprehensive to cover all logical fault models.

The main novelties of our proposed BIST are 100% exhasutive coverage of all logic-level faults (including stuck-at, bridging, open, and functional faults) and minimality of test vector and configurations set. Our proposed approach is based on implementing *single-term* functions in all crossbars during the test mode which allows propagation of all sensitized faults to the outputs, and hence, detection [33, 34]. The truth table of a single-term function consists of only one minterm or one maxterm. The input pattern corresponding to that minterm (maxterm) is called *activating input* (AI). A single-term function can be viewed as an AND (OR) function with possible inversions at the inputs and/or output, e.g. $F = A + B' + C' + D$ with $AI_F : ABCD = 0110$. If the primary inputs to a logic network of single-term functions are assigned such that the inputs to each single-term function is its activating input, then all (subsets of multiple) sensitized faults in the network will be detected. Figure 7 shows an example of a network of single-term functions with test vector 100011. This test vector results in activating inputs at the inputs of all single-term functions in this logic network. All sensitized faults (including all multiple subsets), i.e. stuck-at-v for all the nets with value v' and bridging faults for all pairs with opposite values, are detected.

By implementing different sets of single-term functions in crossbars and applying appropriate test patterns to sensitize all faults, all crossbars can exhaustively be tested. In each test configuration, a subset of faults are sensitized and then the corresponding single-term functions are implemented in the logic blocks (crossbars). A set of test configurations is then required to cover all faults and achieve 100% coverage. Test vector and configuration generation process can be fully automated. The number of test configurations is logarithmic to the size of array [35]. For instance, a design with one billion (10^9) nets can exhaustively be tested for all possible 10^{18} logic-level faults in only 30 test configurations.

Diagnosis is achieved by selecting the subset of sensitized fault in each test configuration in such a way that the pass/fail outcomes of test configurations uniquely encodes the faulty resources. The number of diagnosis configurations

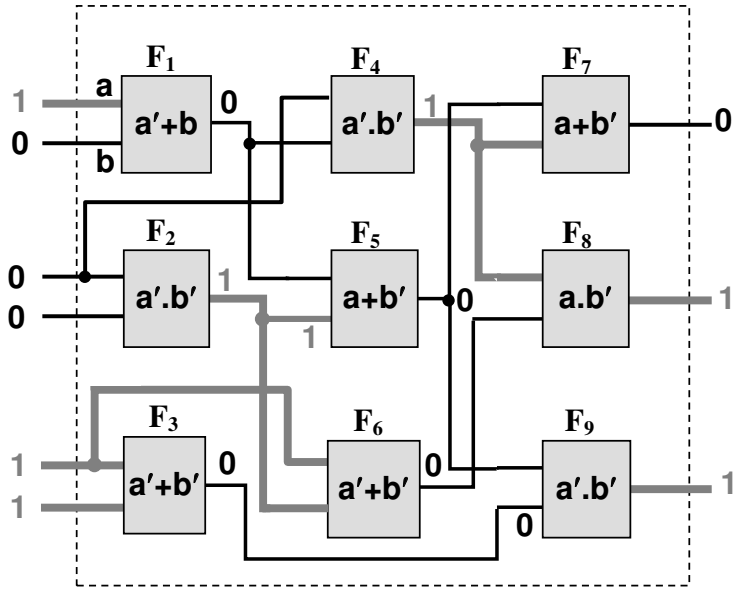


Figure 7: A logic network of single-term functions

is also logarithmic to the number of faults. The subsets of sensitized faults in diagnosis configurations can be modeled by block codes. Each diagnosis configuration represents a code word and sensitized faults in each diagnosis configuration are bit positions in the corresponding code word. In order to achieve multiple fault diagnosis, block codes with desirable distance d_{min} ($t = \lfloor (d_{min} - 1)/2 \rfloor$ error correcting capability), such as BCH codes [36, 37, 38], need to be considered and the corresponding diagnosis configurations (single-term functions and test vectors) can be generated. In this case, any required t multiple fault diagnosis within each crossbar can precisely be diagnosed. Note that since all n crossbars are tested and diagnosed simultaneously in the proposed BIST/BISD scheme, $n.t$ multiple faults in the crossbar array can be diagnosed with only $O(t + \log n)$ test/diagnosis configurations [39].

5.2. Built-in Self-mapping (BISM)

Since it is expected that all manufactured nano-chips contain a considerable percentage of defects even in a mature fabrication process, defect tolerance is inevitable. The goal of defect tolerance is to bypass defective resources using test and diagnosis information. Since defects are device specific, this part of the design flow, mapping the application and bypassing defective resources, has to

be device specific as well. However, the information required for such mapping, which is obtained only after test and diagnosis, is not available at the design time. Therefore, some parts of the application mapping phase have to be postponed from *design time* to the *test time*. Nevertheless, we propose a novel design flow to minimize such per-chip customized design efforts in Sec. 5.3.

As parts of the design flow are shifted to the test time, we propose a built-in self-mapping (BISM) approach to minimize per-chip customized mapping efforts. BISM allows the crossbar array to be configured by the on-chip interface circuitry and bypass defective resources. It also reduces physical design efforts in which detailed placement and routing will be performed on-the-fly. In other words, only global placement and routing has to be completed at the design time and detailed configuration of individual crossbars (for logic mapping or signal routing) will be determined at the configuration time by BISM.

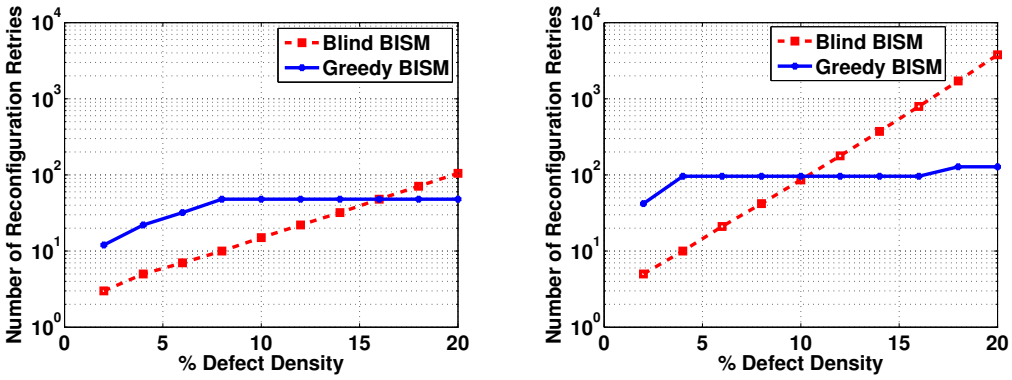
We propose the following BISM schemes depending on the defect density level.

Blind BISM. This is the simple and fast implementation of BISM. In this scheme, a random configuration for the crossbar is generated on-the-fly and then application-dependent BIST is used to check whether this configuration is defect-free. The above process is repeated if BIST detects any fault in the generated configuration. Blind BISM is suitable for low defect densities in which it is expected that a randomly generated configuration is defect-free with a high probability such that few configuration retries are performed. Since no application-independent test is performed and no diagnosis is involved (neither application-independent nor application-dependent), blind BISM is very fast and effective for low defect-densities. The self-reconfiguration circuitry is also very simple and small.

Greedy BISM. When defect density is high, blind BISM approach becomes ineffective due to too many configuration retries. In this case, greedy BISM is performed as follows. It starts with a random configuration followed by BIST. If the configuration is failed, application-dependent BISD is performed to identify the defective resources utilized in the most recent configuration. The self-reconfiguration uses the diagnosis information to only bypass (reconfigure) the defective parts of that configuration. This process is repeated until the last configuration is defect-free. Note that each retry phase in greedy BISM takes longer than blind BISM since application-dependent BISD is used and self-reconfiguration is more complex. However, the number of retries is smaller compared to blind BISM for higher defect densities, resulting in shorter mapping time in overall.

Hybrid BISM. This BISM procedure is the combination of the above procedures and works for all defect densities and also various defect density distributions

across different crossbars in a nano-chip, i.e. ideal for both global and local defect density variations. In hybrid BISM, the BISM procedure initially starts with blind BISM. If it is not successful after a pre-defined number of retries, it automatically switches to greedy BISM. The hybrid approach can quickly configure the crossbars with smaller defect densities and also performs well on crossbars with higher defect densities.



(a) 16×16 (b) 32×32

Figure 8: The number of configuration retries (a) 16×16 crossbar (b) 32×32 crossbar

Figure 8 shows the number reconfiguration retries for defect-free matching (used for logic mapping and signal routing) in 16×16 and 32×32 crossbars for various defect densities using blind and greedy BISM. The number of BISM retries as well as the number of test configurations (blind BISM) and diagnosis configurations (greedy BISM) have been considered here.

5.3. Application-Independent Defect Tolerant Flow

In the conventional defect tolerant flow, the existence and the location of defective elements are identified using test and diagnosis steps and stored in *defect map*. Defect tolerance is achieved by avoiding defective resources in the physical design flow using the defect map. Particularly, placement and routing phases of the physical design use the defect map in order to map the design to the crossbar array (the link from physical design back to the crossbar array) by using only defect-free resources. This flow is shown in Figure 9(a). We call this flow *defect-aware* since design phases use the defect map information. However, due to prohibitively large size of defect map and per chip customization of entire

design flow, this traditional approach cannot be used for high-volume production of nano-chips.

Most drawbacks of the traditional *defect-aware* flow are due to the fact that this method is *application dependent*, i.e. defects are handled in a per-application basis. In contrast to the defect-aware design flow, we propose a *defect-unaware* design flow to tolerate defects in crossbar arrays. This design flow is shown in Figure 9(b). In this flow, defect tolerance is performed once and the same recovered (defect-free) set of resources are used for all applications. In the proposed flow, almost all design steps remain unaware of the existence and the location of defects within the nano-chip. The key idea in the proposed defect-unaware flow is to identify *universal* defect-free subsets of resources within the original partially-defective nano-chip. All design steps work with this universal defect-free subset of the chip called the *design view*. The size of these "universal" subsets is identical for all nano-chips fabricated in the same process environment (similar defect densities). Also, these universal defect-free subsets remain unchanged for different applications mapped into the same nano-chip, making this approach *application-independent*. There is a *final mapping* phase, with very low complexity, at the end of physical design flow that makes the connection between the defect-free design view and the actual *physical view* of the nano-chip which contains actual defects. This is the only defect-aware step which has to be performed per chip. This final mapping phase will be implemented as a part of the proposed BISM approach.

The main idea of this mapping flow is to make some of the defect tolerant flow generic and independent of individual fabricated crossbars and chips, and then some final mapping has to be done in a per-chip basis. This is in contrast with the traditional flow that the entire physical design and mapping steps are done in per-chip basis. The "generic" part of the flow has to work with a generic model of crossbars and should only work with expected defect distribution. This means that for a fabricated NxN crossbar, there will be a $k \times k$ defect-free subset assumed ($k < N$), which cannot be adjusted in a crossbar basis, otherwise it would defeat the purpose of "generality" and "device independence". As the reviewer also mentioned, in reality, the same value of k cannot be guaranteed for all crossbars. Therefore, at a higher level, a notion of crossbar yield comes to the picture, which means whether the actual defect-free subset of a particular crossbar is at least $k \times k$. Otherwise, that crossbar is marked as defective and unusable. In other words, the actual crossbar is either usable (has a defect-free subset of at least $k \times k$) or unusable. This information can be used in global mapping phase and high-level defect and fault tolerant scheme, which is in our future work.

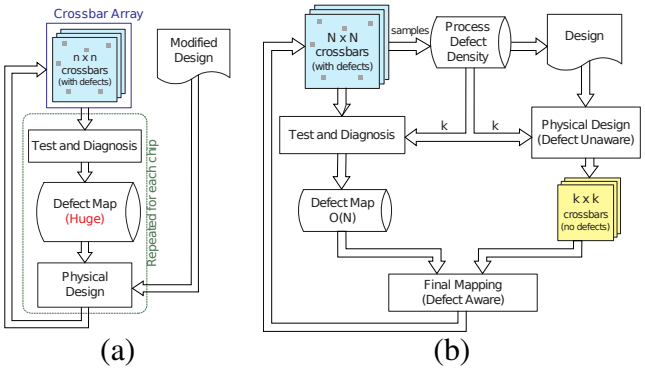


Figure 9: (a) Traditional defect-aware design flow (b) The proposed defect-unaware design flow

5.4. Built in Self Reconfiguration and Adaptation

As it has been already proved, future complex devices based on nanowire crossbars provide better density over conventional CMOS devices due to the new methods for growing and assembling. They are also a very good candidate for future high density interconnects, combinational circuits and storage parts. Some of the potential solutions are of regular types [40], others based on memristive networks are irregular structures [41].

These technologies are highly sensitive to design variations, defects and intermittent faults, or susceptible to environmental factors, such as thermal stress, radiation, and so on. It may result in crossbars structures with high number of defect rates related to manufacturing or environmental constraints, much more than what are considered today in conventional CMOS technologies. These threats can be seen as major obstacles to adopting and using such architectures and technologies to build future application specific circuits or even processors.

Once the crossbar array size is determined and the defect map is provided with BISM methodology, built in self reconfiguration and adaptation architectural schemes can be also designed to cope with higher defect level. In most of the cases they are using additional spare units. As a matter of fact, new fault tolerant design paradigm is needed, since with such high defect densities both the regular resources and the redundant ones will be affected by the defects, disabling the basic principle of traditional fault tolerance (use of a fault-free redundant unit to perform the job of a faulty regular unit). When the defect level exceeds the capability of the BSIM tool, this possibility allows to provide enough spares in

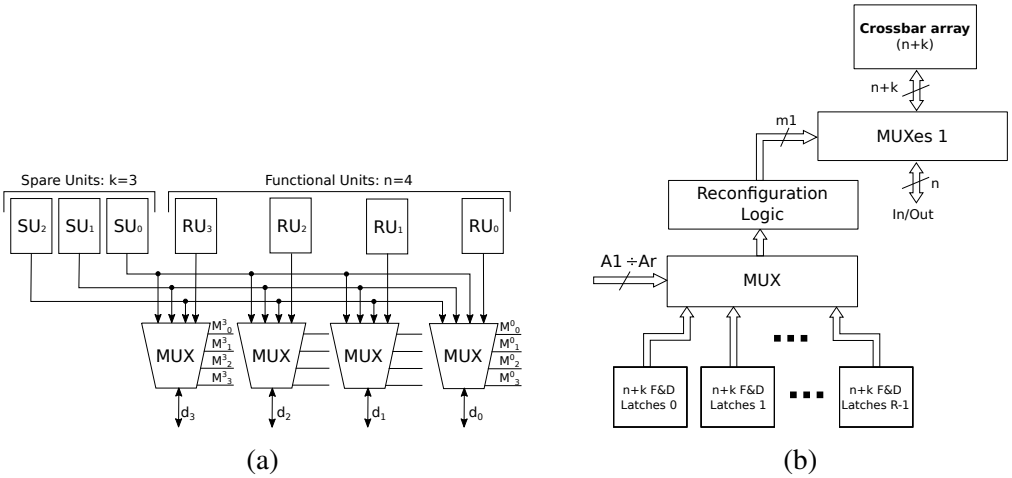


Figure 10: Reconfiguration technique for a crossbar array implemented with spare single function units. (a) shows the global architecture (for $k = 3$ and $n = 4$). (b) shows the connection with BIST unit.

the form of k additional column-like units. This is solution is easier to implement and design as the complexity of the reconfiguration functions are easier to master. Specific min-terms functions in the spare independent columns can be implemented and used by the reconfiguration functions. These functions could repair multiple faults affecting both the regular and spare elements and perform the repair by means of a single test/repair pass. The scheme also minimizes the hardware cost for implementing the repair control and for storing the reconfiguration information. It optimizes the BISR cost and the repair efficiency. These are important attributes when we have to consider high defect densities. The general architecture of this approach is presented in Figure 10.

We dispose of $(n, n + k)$ crossbar, where $n \times n$ is the size of the initial functional units in the crossbar and k the number of spares available in a column like organization ($k \geq 0$). The results of crossbar testing performed by means of the BIST/BISD comparison operations are stored in the $n + k$ F&D registers, and point out exactly the location of the defect, selected by the value of the address bits $A1, A2, ..., Ar$ at MUX (Figure 10(b)). The reconfiguration logic drives the MUXes1 (which are also shown in Figure 10(a)) performing the repair. These MUXes connect the right-most functional units to the k fault-free spares. This reconfiguration can be done at static level and the granularity n can be chosen base on the crossbar size and the defect level. Evaluations on different sizes, defect levels and functions are currently under study, but preliminary results allow suc-

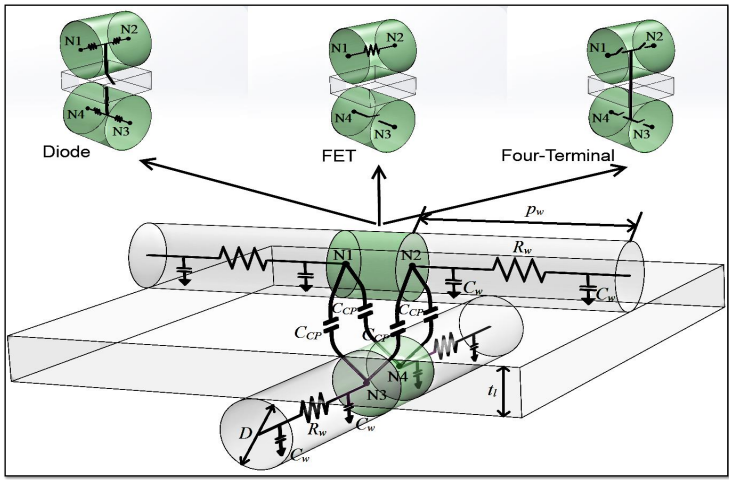


Figure 11: A nano-crossbar cell and its different forms for diode, FET, and 4-terminal switch based crosspoints.

cessfully combining faulty regular and with fault-free spare units, but also faulty functional with faulty spare units (in case of a high defect density) to create a fault-free unit. The combination works as far as the two units are affected by faults involving the same polarities or errors.

6. Capacitor-Resistor Modeling

Previous studies on performance modeling and analysis of nanoarrays lack of accuracy and comprehensiveness. Capacitor-resistor models and their parameter values are determined with weak assumptions without in depth analysis of nanoarray technologies [42]. Additionally, some studies exploit current or predictive technology models for nanoscale CMOS which certainly has major differences from nanoarray based technologies, both in design and manufacturing levels [43, 44]. In this part, we aim to overcome these shortcomings by introducing accurate modeling and performance analysis techniques for nano-crossbar arrays [45].

Nano-crossbar arrays are regular structures consisting of identical crosspoint cells. Figure 11 illustrates a cell with the proposed capacitor-resistor placements. It consists of two crossed lines/wires with intersecting parts shown in green and nonintersecting parts shown in grey. The intersecting part is expected to behave as an electronic component such as a diode, a FET, or a switch. We model this part with wire resistors and four identical crosspoint capacitors C_{CP}'s. The reason

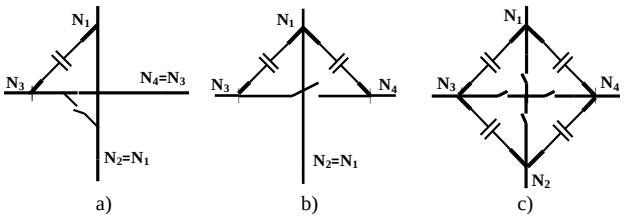


Figure 12: Capacitor and switch placements for crosspoints based on a) diode, b) FET, and c) four-terminal switch (N_1-N_2: upper, N_3-N_4: lower).

of using four capacitors instead of one is the necessity of considering resistances between N_1-N_2 and N_3-N_4 nodes. Using a single crosspoint capacitor is only applicable if these resistances are negligibly small. For the nonintersecting parts, we use a wire resistor R_w and a wire capacitor C_w that is composed of parasitic wire, parallel wire, wire-layer, and wire-bulk capacitors. Other parameters defined are wire diameter D, layer thickness t_l (between wires), and pitch size p_w (distance between parallel wires).

We explicitly show our model's applicability for three different technologies of nanowire crossbar arrays where each crosspoint behaves as a diode, a FET, and a four-terminal switch as shown in the upper part of Figure 11. Here, along with wire resistors we use switches having series ON and OFF parasitic resistances. For the diode-based crosspoint, it is assumed that the upper and the lower wires are p-type and n-type nanowires, respectively. The crosspoint is modeled with a switch, representation of a pn-diode four capacitors, and four wire resistors. For the FET based crosspoint, the layer between two wires acts as an insulator, so no current flows between the wires. The upper wire is modeled with a resistor and the lower wire is modeled as a switch controlled by the upper wire's voltage. Since the upper wire does not conduct current, N_1 and N_2 nodes are shorted that results in two crosspoint capacitors, each having a value of $2C_{CP}$. For the four-terminal switch based crosspoint, the upper and the lower wires are identically modeled using total of four capacitors and four switches. Here, current can flow in multiple directions. Comparison of these three models with neglected wire resistors is visualized in Figure 12 (N_1-N_2: upper, N_3-N_4: lower).

Simplified power-delay model

We simplify our capacitor-resistor models with an aim of effectively using them for power and delay analysis [45]. We transform in-between node capacitors C_{CP}'s, shown in Figure 12, into grounded equivalent node capacitors C_{CP-eqv}'s

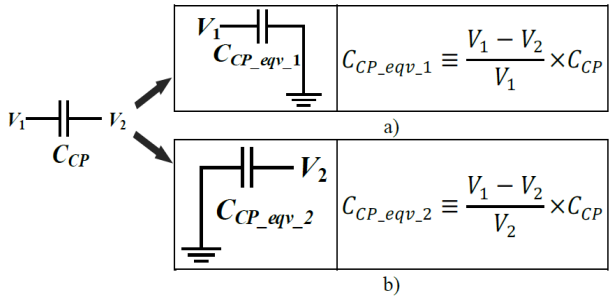


Figure 13: C_{CP} and its equivalent capacitors a) $C_{CP-eqv-1}$ on V_1 and b) $C_{CP-eqv-2}$ on V_2.

that is to be compatible with the Elmore delay model. Miller theorem is used for this purpose. Equivalent grounded capacitors for C_{CP}'s are obtained with the formulas given in Figure 13. Formulas are derived by exploiting the conservation of the capacitor charge Q_C; recall that $I_C = C \times \frac{dV_C}{dt}$ and $\Delta Q_C \cong C \times \Delta V_C$.

Our crosspoint model and its equivalent with grounded capacitors are shown in Figure 14a) and Figure 14b), respectively. There are two criteria for comparing these two models: 1) effectiveness of using them in power-delay analysis, and 2) accuracy and easiness of calculating related capacitor values. For the first criteria, the model in Figure 14b) overwhelms the other; grounded capacitors are highly desired both in circuit simulations and hand calculations. However, things are reversed for the second criteria. Since we define C_{CP}'s with physical reasoning, we can calculate their values using technology parameters such as distances, concentrations, and physics constants. On the other hand, accurately calculating the values of C_{CP-eqv}'s necessitates to know node voltage values and this might not be practical regarding that node voltages are dynamically changing between a supply voltage and a ground, namely V_{DD} and GND = 0V. This problem can be solved using an assumption of $C_{CP-eqv} = 3C_{CP}$ to calculate C_{CP-eqv} at the cost of lower accuracy [45]. This assumption along with other probable assumptions are thoroughly justified in the referred paper.

In simulations with Spice (circuit simulation tool), we see that even for small crossbars having less than 10 crosspoints, the model in Figure 14a) requires a large computational load that results in impractical runtimes needed to have delay and power values. On the other hand, using the model in Figure 14b) with an assumption of $C_{CP-eqv} = 3C_{CP}$ is very efficient in terms of the computing time. Also it is reported that the delay values obtained with this model deviated only

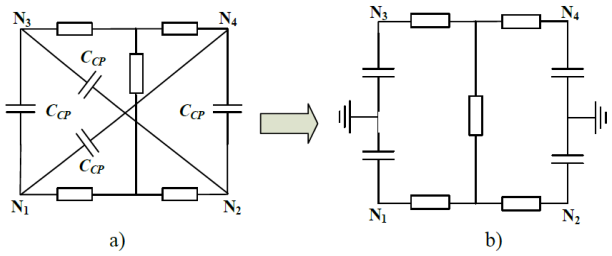


Figure 14: Our crosspoint models using a) real crosspoint capacitors and b) their equivalent grounded capacitors.

3% in average from the values using the model in Figure 14a) [45].

7. Conclusion

In this paper we have summarized the results obtained in the first part of the project. In particular, we have investigated logic synthesis and area optimization techniques for diode, FET, and four-terminal switch based nanoarrays. We have proposed new decomposition methods for the four-terminal switch based model. We have also studied defect, variation and fault tolerant techniques in the presence of high defect densities and extreme parametric variations, particularly for crossbar array nano-architectures. Finally, we have introduced capacitor-resistor models for power and delay analysis.

Integrating a new technology into a mature industry such as the semiconductor industry is a long road in which device performances and manufacturability have to be developed jointly through a blend of advanced research, technology development and industry-compliant implementation. One of the major promises of emerging nanotechnologies for on-chip applications is ultimate integration density, manufacturing and integration cost reduction, and the reduction of power consumption. However, there is a big gap in 1) extending the existing electronic design automation flow for emerging technologies in order to introduce them in the architecture and system design in a systematic-way, and 2) novel computer architecture systems based on emerging technologies to provide high performance and minimize power consumption at the same time. Therefore, future work on this project includes the introduction of hybrid EDA flow as well as emerging computer architectures by gathering well respected experts working in these broad fields. In particular, the remaining part of the project will focus on implementation of logic, arithmetic, and memory elements, in order to realize a nano-crossbar

based synchronous state machine.

8. Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 691178.

References

[1] Overall Technology Roadmap Characteristics, Tech. rep., International Technology Roadmap for Semiconductors (2010, retrieved 2013).

[2] M. Dubash, Maaoore -'- s law is dead, says gordon moore, Techworld. com (13).

[3] K. Ariga, M. V. Lee, T. Mori, X.-Y. Yu, J. P. Hill, Two-dimensional nanoarchitectonics based on self-assembly, Advances in Colloid and Interface Science 154 (1-2) (2010) 20 – 29.

[4] G. M. Whitesides, B. Grzybowski, Self-Assembly at All Scales, Science 295 (5564) (2002) 2418–2421.

[5] W. Lu, C. Lieber, Nanoelectronics from the Bottom Up, Nat Mater 6 (11) (2007) 841–850.

[6] P. Avouris, Molecular Electronics with Carbon Nanotubes, Acc. Chem. Res. 35 (12) (2002) 1026–1034.

[7] M. Altun, M. D. Riedel, Logic Synthesis for Switching Lattices, IEEE Transactions on Computers 61 (11) (2012) 1588–1600.

[8] D. Alexandrescu, M. Altun, L. Anghel, A. Bernasconi, V. Ciriani, L. Frontini, M. Tahoori, Synthesis and Performance Optimization of a Switching Nano-Crossbar Computer, in: 2016 Euromicro Conference on Digital System Design (DSD), 2016, pp. 334–341.

[9] M. Ayasoyu, M. Altun, S. Ozoguz, K. Roy, Spintronic memristor based offset cancellation technique for sense amplifiers, in: The International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2017.

[10] S. B. Akers, A Rectangular Logic Array, in: Switching and Automata Theory, 1971., 12th Annual Symposium on, 1971, pp. 79–90.

[11] A. Y. Zomaya, Handbook of Nature-Inspired and Innovative Computing, 2006.

[12] Y. C. Chen, S. Eachempati, C. Y. Wang, S. Datta, Y. Xie, V. Narayanan, Automated Mapping for Reconfigurable Single-electron Transistor Arrays, in: Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE, 2011, pp. 878–883.

[13] A. Dehon, Nanowire-based Programmable Architectures, J. Emerg. Technol. Comput. Syst. 1 (2) (2005) 109–162.

[14] A. Khitun, M. Bao, K. L. Wang, Spin Wave Magnetic NanoFabric: A New Approach to Spin-Based Logic Circuitry, IEEE Transactions on Magnetics 44 (9) (2008) 2141–2152.

[15] Y. Levy, J. Bruck, Y. Cassuto, E. G. Friedman, A. Kolodny, E. Yaakobi, S. Kvatinsky, Logic Operations in Memory Using a Memristive Akers Aarray, Microelectronics Journal 45 (11) (2014) 1429 – 1437.

[16] Z. Chen, J. Appenzeller, Y.-M. Lin, J. Sippel-Oakley, A. G. Rinzler, J. Tang, S. J. Wind, P. M. Solomon, P. Avouris, An Integrated Logic Circuit Assembled on a Single Carbon Nanotube, Science 311 (5768) (2006) 1735–1735.

[17] H. Yan, H. S. Choe, S. Nam, Y. Hu, S. Das, J. F. Klemic, J. C. Ellenbogen, C. M. Lieber, Programmable Nanowire Circuits for Nanoprocessors, Nature 470 (7333) (2011) 240–244.

[18] A. H. Shaltoot, A. H. Madian, Memristor Based Carry Lookahead Adder Architectures, in: Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on, 2012, pp. 298–301.

[19] G. Snider, P. Kuekes, R. S. Williams, CMOS-like Logic in Defective, Nanoscale Crossbars, Nanotechnology 15 (8) (2004) 881.

[20] G. S. Snider, P. J. Kuekes, D. R. Stewart, Nanoscale Latch-array Processing Engines (06 2007).

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

[22] O. Tunali, M. Altun, Defect tolerance in diode, fet, and four-terminal switch based nano-crossbar arrays, in: Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on, IEEE, 2015, pp. 82–87.

[23] M. Altun, Computing with emerging nanotechnologies, in: Low-Dimensional and Nanostructured Materials and Devices, Springer, 2016, pp. 635–660.

[24] O. Tunali, M. Altun, Permanent and transient fault tolerance for reconfigurable nano-crossbar arrays, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, DOI: 10.1109/TCAD.2016.2602804doi:10.1109/TCAD.2016.2602804.

[25] G. Gange, H. Søndergaard, P. J. Stuckey, Synthesizing Optimal Switching Lattices, ACM Trans. Design Autom. Electr. Syst. 20 (1) (2014) 6:1–6:14.

[26] A. Bernasconi, V. Ciriani, R. Drechsler, T. Villa, Logic Minimization and Testability of 2-SPP Networks, IEEE Trans. on CAD of Integrated Circuits and Systems 27 (7) (2008) 1190–1202.

[27] A. Bernasconi, V. Ciriani, G. Trucco, T. Villa, Using flexibility in p-circuits by boolean relations, IEEE Trans. Computers 64 (12) (2015) 3605–3618.

[28] A. Bernasconi, V. Ciriani, G. Trucco, T. Villa, On Decomposing Boolean Functions via Extended Cofactoring, in: Design Automation and Test in Europe (DATE), 2009.

[29] A. Bernasconi, V. Ciriani, V. Liberali, G. Trucco, T. Villa, Synthesis of P-Circuits for Logic Restructuring, Integration 45 (3) (2012) 282–293.

[30] A. Bernasconi, V. Ciriani, L. Frontini, V. Liberali, G. Trucco, T. Villa, Logic synthesis for switching lattices by decomposition with p-circuits, in: Euromicro Conference on Digital Systems Design (DSD), 2016.

[31] A. Bernasconi, V. Ciriani, Dimension-reducible boolean functions based on affine spaces, ACM Trans. Design Autom. Electr. Syst. 16 (2) (2011) 13.

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

[33] M. B. Tahoori, Application-Dependent Testing of FPGAs, in: IEEE Transaction on Very Large Scale Integrated Circuits , 2006.

[34] M. B. Tahoori, Application-Dependent Diagnosis of FPGAs, in: Proc. International Test Conference, 2004, pp. 645–654.

[35] M. B. Tahoori, Application-dependent testing of fpgas, IEEE Trans. VLSI Syst. 14 (9) (2006) 1024–1033. doi:10.1109/TVLSI.2006.884053.
URL https://doi.org/10.1109/TVLSI.2006.884053

[36] A. Hocquenghem, Codes corecteurs d'erreurs, in: Chiffres, Vol. 2, 1959, pp. 147–156.

[37] R. C. Bose, D. K. Ray-Chaudhuri, On a Class of Error Correcting Binary Group Codes, in: Intl Control, Vol. 3, 1960, pp. 68–79.

[38] W. W. Peterson, Encoding and Error Correction Procedures for the Bose-Chaudhuri Codes, in: IRE Trans. Inf. Theory, Vol. IT-6, 1960, pp. 459–470.

[39] M. B. Tahoori, High resolution application specific fault diagnosis of fpgas, IEEE Trans. VLSI Syst. 19 (10) (2011) 1775–1786. doi:10.1109/TVLSI.2010.2056941. URL https://doi.org/10.1109/TVLSI.2010.2056941

[40] G. Snider, Computing with hysteretic resistor crossbars, Appl. Phys. A 80 (2005) 1165 – 1172.

[41] A. Z. Stieg, A. V. Avizienis, H. O. Sillin, M. A. C. Martin-Olmos, J. K. Gimzewski, Emergent Criticality in Complex Turing B-type Atomic Switch Networks, Advanced Materials 24 (2) (2012) 286 – 293.

[42] M. M. Ziegler, C. A. Picconatto, J. C. Ellenbogen, A. Dehon, D. Wang, Z. Zhong, C. M. Lieber, Scalability simulations for nanomemory systems integrated on the molecular scale, Annals of the New York Academy of Sciences 1006 (1) (2003) 312–330.

[43] M. Gholipour, N. Masoumi, A comparative study of nanowire crossbar and mosfet logic implementations, in: EUROCON-International Conference on Computer as a Tool (EUROCON), 2011 IEEE, IEEE, 2011, pp. 1–4.

[44] W. Lu, P. Xie, C. M. Lieber, Nanowire transistor performance limits and applications, IEEE transactions on Electron Devices 55 (11) (2008) 2859–2876.

[45] M. C. Morgul, F. Peker, M. Altun, Power-delay-area performance modeling and analysis for nano-crossbar arrays, in: VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on, IEEE, 2016, pp. 437–442.