Spring 2008

Defect-tolerance and testing for configurable nano-crossbars

Mandar V. Joshi

Department:

DEFECT-TOLERANCE AND TESTING FOR CONFIGURABLE NANO-CROSSBARS

by

MANDAR VIJAY JOSHI

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE
IN COMPUTER ENGINEERING

2008

Approved by

Waleed K. Al-Assadi, Advisor
Scott C. Smith
Minsu Choi
Theodore McCracken

PUBLICATION THESIS OPTION

This thesis consists of the following two articles that have been submitted for publication as follows:

Pages 1-20 are accepted at the IEEE INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT-TOLERANCE IN VLSI SYSTEMS 2007

Pages 21-40 are submitted to IEEE TRANSACTIONS ON VLSI SYSTEMS, 2008

ABSTRACT

Moore's Law speculated a trend in computation technology in terms of number of transistors per unit area that would double roughly every two years. Even after 40 years of this prediction, current technologies have been following it successfully. There are however, certain physical limitations of current CMOS that would result in fundamental obstructions to continuation of Moore's Law. Although there is a debate amongst experts on how much time it would take for this to happen, it is certain that some entirely new paradigms for semiconductor electronics would be needed to replace CMOS and to delay the end of Moore's Law. Silicon nanowires (SiNW) and Carbon nanotubes (CNT) possess significant promise to replace current CMOS. Digital circuits can be synthesized using programmable junctions (crosspoints) in 2-D arrays of CNTs of pitches of the order of a few nanometers. Programmable Logic Arrays and memories using this technology have also been proposed. This technology, however exhibits significantly high defect rates, creating failures in configurations. Some researchers have proposed methods that can detect defective crosspoints, and others have also proposed techniques that avoid configuration at the defective crosspoints with a very high success rate. These techniques, however, have certain limitations that may produce poor yield from the configuration; i.e. programming of some defective or non-programmable crosspoints. Also, these techniques need exhaustive defect mapping before the configuration algorithms are applied. This adds to the time complexity for overall configuration and routing process. Paper I deals with redundancy methods to minimize the defects in configuration and in turn decrease the time complexity for configuration. Paper II proposes a Built-in Self Test (BIST) technique to generate the defect map in a 2-D nano-array.

ACKNOWLEDGMENTS

I am highly thankful to my thesis advisor, Dr. Waleed Al-Assadi for his immense support, guidance and encouragement during my course-work, research and projects I was involved in. I am also grateful to my thesis committee members, Dr. Minsu Choi, Dr. Ted McCracken and Dr. Scott Smith for their cooperation. I would also like to thank my family members, room-mate Poorna, lab-mates Sagar, Sasikiran, Sindhu and Vipin for all their constant help and support.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

PAPER I

Figure Page

PAPER II

PAPER I

NANOFABRIC PLA ARCHITECTURE WITH REDUNDANCY ENHANCEMENT

Mandar V. Joshi, Waleed K. Al-Assadi
Electrical and Computer Engineering
Missouri University of Science and Technology, Missouri, U.S.A 65409
Email: mvjvx8@mst.edu, waleed@mst.edu

Abstract

Fundamental electronic structures such as Diodes and FETs have been shown to be constructed using selectively doped semiconducting Carbon Nanotubes or Silicon Nanowires (CNTs, SiNWs) at nanometer scale. Memory and Logic cores have been proposed, that use the configurable junctions in 2-D crossbars of CNTs. These Memory and Logic arrays at this scale exhibit a significant amount of defects that account for poor a yield. Configuration of these devices in the presence of defects demands an overhead in terms of area and programming time. This work introduces a PLA configuration that makes use of fixed and adaptive redundancy in terms of the number of nanowires. This is done in order to simplify the process of programming the PLA, increase the yield, reduce the time complexity, and in turn, reduce the cost of the system.

1. Introduction

Recent advances in Photolithographic techniques have made possible the miniaturization of electronic circuits. According to Moore's Law, the number of transistors per unit area will continue to double approximately every two years. However, the applicability of Moore's law will cease to continue as the pitch sizes approach molecular dimensions. It therefore becomes

necessary to explore devices and technologies that can match these trends of an increase in transistors per area [1]. Programmable Logic Array (PLA) architectures using Carbon Nanotubes that make use of their semiconducting properties have been previously suggested and are briefly dealt with. This paper proposes a technique to tolerate defects in those PLA architectures at Nanometer scale using Carbon Nanotubes and Si Nanowires.

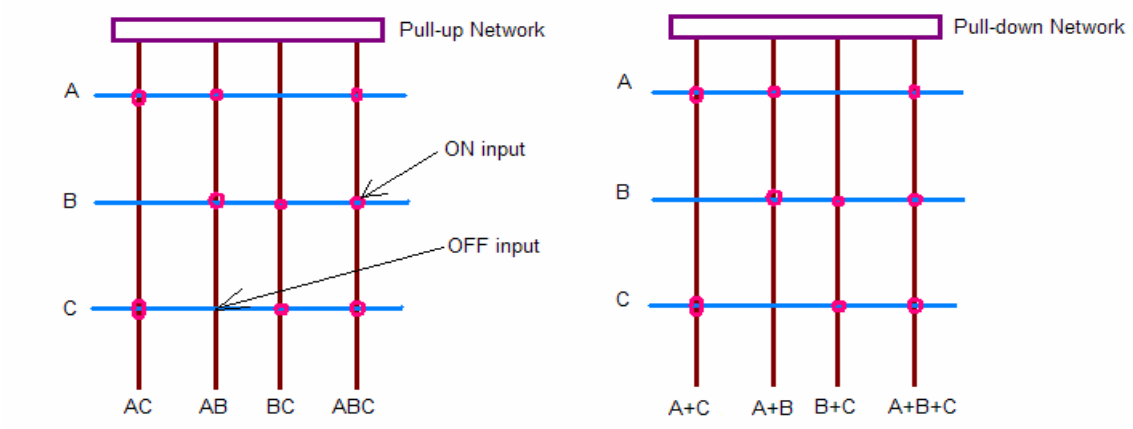
Semiconducting Carbon-Nanotubes and SiNWs exhibit electronic properties similar to those of conventional lithographic-scale CMOS devices in terms of electron and hole mobilities. Chemical passivation of SiO_x shell surrounding single crystal SiNW cores has been shown to significantly enhance conductance-gate voltage behavior making these wires highly suitable to be used as Field Effect Transistors [2], and in turn as building blocks for digital circuits. The electronic applications of NWs are based on diode and FET-like properties of NW junctions or *"Crosspoints"* in 2-D arrays, called as *Nanofabrics* or *Crossbars.* Crosspoints can be grouped to form a memory or logic device. Cha et al. [3] have shown electro-mechanical switching devices using suspended nanotubes. The Crosspoints at the junctions are programmed using this "Bistable" property that they exhibit. Their ON-state behavior is similar to that of a diode. When the two wires forming a junction are in close contact, the junction resistance is very small; when the wires are far away, their resistance increases by a great extent (~33MΩ in one state and ~10kΩ in the other) [3]. A Crosspoint can be programmed ON or OFF by applying a voltage differential of ~3.6V. The Crosspoint takes part in the evaluation of Boolean expressions in the state in which they show diode-like properties.

Figure 1 illustrates the setups for a Nano-crossbar as an AND array and OR array, respectively. The working is based on diode-like properties of Crosspoints and the presence of a pull-up/pull-down network. The programmable Crosspoints allow this network to implement logical sum/product terms on it. The inputs that take part in the evaluation of the sum/product terms are called ON inputs, and the ones that do not are called OFF inputs. Due to the presence of defects, these Crosspoints may lose their programmability. Thus, if an ON input corresponds to a

defective Crosspoint, then it results in a faulty output. The defect mechanisms will be discussed in section 2.

The Synthesis of Boolean expressions can be made possible on PLAs based on crossbars. A row in a crossbar can be made to act as a Boolean product/sum term by programming ON only the junctions or Crosspoints that correspond to the variables that take part in the term, as shown in Figure 1. Inputs A and C in first row of Figure 1(a) are the ON inputs, as they take part in the evaluation of the product term. The Rest of the inputs are called the OFF inputs. The programmability of a Crosspoint is statistical in nature [8], and therefore such a configuration of PLAs gives a poor number of successfully configured Crosspoints even for a small number of junctions to be programmed on a NW. This work proposes redundancy schemes to tolerate the occurrence of Crosspoint defects to obtain an acceptable yield for PLA configuration.

The paper is organized as follows; Sections 2 and 3 describe the defect model and mapping. Section 4 gives previous work of PLA architectures and configuration algorithm. Section 5 introduces the two Variable Redundancy approaches for defect tolerance. Finally Section 6 details the simulation results with respect to both the approaches.



(a) **(b)**

Figure 1. (a) AND-array (b) OR-array using nanowires

2. Crosspoint defect model

The bistable property of Crosspoints can be used to implement *logic blocks* in a PLA [4] or *memory cells* [5] [9]. The Crosspoints can lose their programmability because of the mechanisms discussed below. For the simulation in this work, a random distribution of such defects is assumed throughout the crossbar.

Breaks in Nanowires: It has been observed that the probability of having breaks in a nanowire increases with the increase in its length. Some breaks may occur during the fabrication of nanowires on account of the limitation of the fabrication techniques and axial stress. Therefore, their lengths should, nominally, not exceed 10 –30 microns. It is reasonable to assume that as high as 5% of the Nanowires exhibit breaks and therefore are unusable [6].

Non-Programmable Crosspoints: These defects are characterized by the inability of a Crosspoint to be programmed as "closed" or "open." The latter is observed to be extremely unlikely and therefore is not considered in the present discussion. The occurrence of defective Crosspoints is a function of the fabrication technique, size of the array, and the random distribution of molecules at the junction area. With reasonable assumptions of operating conditions, it can be proved that the occurrence of a "permanently open" defect is largely due to the absence of sufficient electrons at the junction area [8].

3. PLA configuration and mapping

Consider a domain of four digital variables, A, B, C, and D. Consider a function F_1 given by F_1=ABC. This expression can be rewritten by F_1=1110, taking all the variables in the sequence A-B-C-D. A "1" is placed if the variable takes part in a logical evaluation, and a "0" is placed if it does not. Therefore A, B and, C are considered ON inputs and D is an OFF input of the variable F_1. It is possible to have n number of such output variables from F_1 to F_n. "F" therefore becomes a matrix with n elements, which provides all the ON and OFF inputs of the logical functions to be evaluated. It is therefore necessary that the locations in the PLA that correspond to ON inputs should be defect free. Another Matrix P provides all the locations of defective Crosspoints of a PLA, where a defect is denoted by "0" and a defect-free Crosspoint is denoted by "1." In order to have a successful match between F and P, the following condition must be satisfied: $P_{i,j} \geq F_{i,j}$

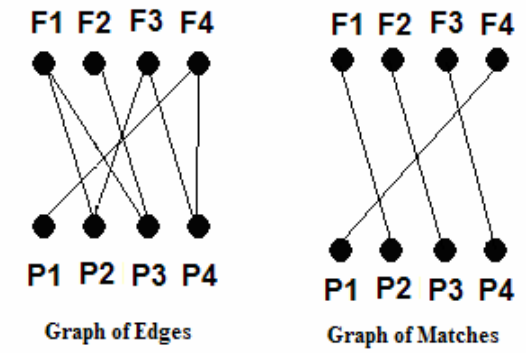


Figure 2. Edges and Matches between F and P

If every column of F and P are to be denoted as a single point as shown in Figure 2, then two rows result one for the Matrix P and the other for Matrix F. In order to get a full yield, every element of F needs to find a match with at least one element of P. Therefore, the problem becomes a classical example of a "Bipartite Graph." In a bipartite graph, two possible relations exist between two points from different matrices.

Let the number of elements in Matrix F be "n." Two points are said to share an "EDGE" if the row corresponding to point in matrix F can be successfully mapped to the row corresponding to the point in Matrix P. Let the number of edges between a pair of matrices be E. Two points are said to have a "MATCH" if and only if they have an edge, and the maximum number of points in Matrix F also has edges with UNIQUE points in Matrix P. Figure 2 illustrates the difference between the graph of edges and the graph of matches.

If the number of matches between a pair of matrices is M and if M cannot be greater than n, the solution lies in maximizing the value of M. Therefore, designer's objective is to be able to program the Nano-PLA in the presence of these defects that amount to approximately 20% or more.

4. Previous Work

Nanofabric Molecular Logic Array (MLA) proposed by Goldstein et al. [7] requires the introduction of both real and inverted inputs from the west and north sides of the array, respectively. This architecture is based on configurable junctions that act as diodes, and consequently, it is incapable of complementing the inputs. The entire Nanofabric is composed of MLAs that either act as "NanoBlocks" (programming elements) or "SwitchBlocks" (switching elements). The present work is targeted primarily for MLA architecture. DFT strategies in Nanowire based PLAs are discussed in [10].

Algorithms that use "detect and avoid" strategies for Crosspoint defects have previously been suggested by DeHon et al. in [8]. For those, it is necessary to have a defect map of the PLA under consideration in order to program it. A defect-tolerant methodology that is proposed in [8] uses a Greedy Heuristic Algorithm to find a solution to the mapping problem discussed above. This algorithm sorts in descending order the functions of the number of ON inputs contained in them,

which enhances the probability of a successful match. This algorithm is intended to be used with Nano-PLA architecture, and the results of this study will be compared with those obtained using the Greedy Heuristic algorithm. It is shown that the time complexity for sorting is an exponential function of the defect rate and number of Crosspoints to be programmed. Therefore, it becomes infeasible to use this algorithm for Nano-PLA with high defect rates and a number of ON inputs greater than 10%.

The proposed redundancy technique in this work can be used in conjunction with NanoPLA in [4] or *Nanofabric Molecular Logic Array* (MLA) Proposed by Goldstein et al. in [7].

5. Nanowire Redundancy

This work targets a NanoPLA with higher defect rates than 20%. It can be noted that the Greedy Algorithm in [8] consumes a very high time complexity for higher defect rates in crossbars. To minimize this time complexity, redundancy was introduced in terms of the number of nanowires. A redundancy scheme has been discussed in [11] that proposes a dynamic reconfiguration algorithm. But it needs the knowledge of presence of defects before reconfiguration, unlike methods suggested in this paper. Two redundancy schemes are proposed here, viz. Double Variable Redundancy and Adaptive Variable Redundancy. The yield rates and area overhead were observed. In Double Variable Redundancy (or DVR) scheme, two horizontal and two vertical NWs are dedicated per input variable. If ‘n’ number of NWs per input variable were dedicated, where the value of ‘n’ is governed by the number of times the variable is used in the function set. It follows that a set of “n” Crosspoints, any of which are programmable, will make the PLA work. It follows that in DVR, “n” always equals two. In Adaptive Variable Redundancy (or AVR) scheme, the number of Crosspoints allocated for a particular sum or product term depends on a number of factors such as the defect rate, size of the PLA in use, and

number of variables taking part in the evaluation of a product term. It is expected that the value of 'n' will increase with the number of Crosspoints to be programmed in a row. This value will also increase with an increase in the defect rate and size of the array.

5.1. Double Variable Redundancy

We propose to obtain greater yield rates for the configuration of a NanoPLA without having to compromise for Time Complexity, seen in [8]. We introduce redundancy in terms of number of nanowires [12]. We allocate two vertical Nanowires per Product (or Sum) term in order to achieve a better yield in presence of crosspoint defects. This is illustrated in Figure 3. If any of the two Vertical Nanowires have a programmable junction with any of the two Horizontal Nanowires in consideration, the condition is equivalent to having a programmable resource. We term this method as Double Variable Redundancy (DVR).

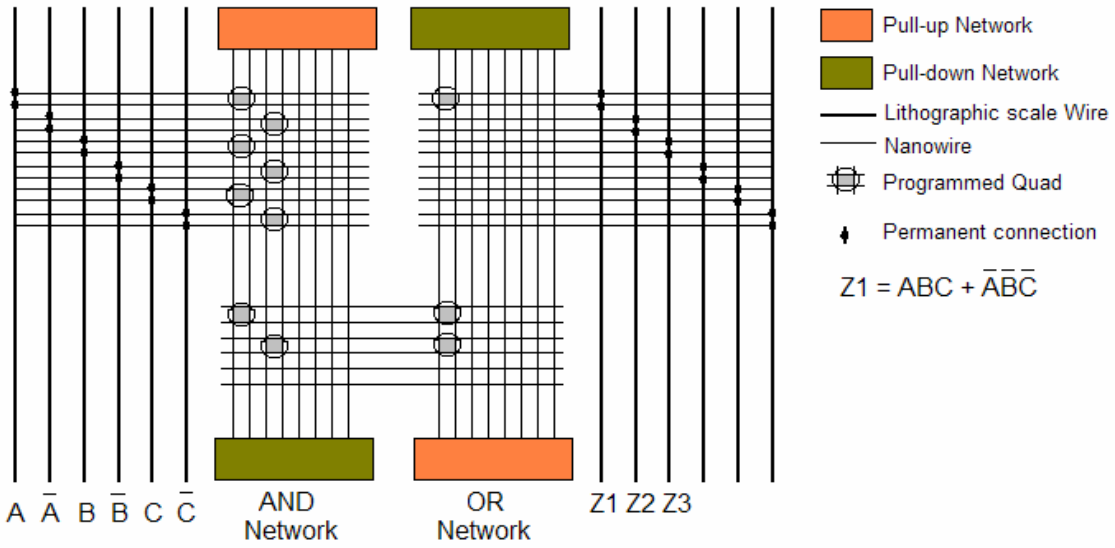


Figure 3. The DVR based proposed PLA Architecture with an illustration of pre-configured Boolean function

It can be mathematically proved that the probability of getting a pair of NWs unsuitable for a given minterm is significantly low as compared to the defect rate in the crossbar. This fact is the result of the three redundant crosspoints present. On account of this, the need of sorting the rows/columns to configure the PLA for given logical functions can be eliminated. The address decoding scheme can be same as that in [3]. The location of Pull-up and Pull-down networks determines the working of the array as AND array or OR array, as seen in Figure 3. In this work, we only make use of the "Diode-like" working of a crosspoint during the evaluation of Boolean functions. Therefore the NanoBlock does not have an ability to invert the inputs, and hence we introduce both, the real and inverted inputs externally. This also eliminates the need of "inverting block" in [3]. The DVR based PLA architecture is illustrated in Figure 3. It shows implementation of product-sum terms using DVR-based PLA.

5.1.1 Calculation of fault probability in DVR based PLA configuration: The following is the list of parameters governing the defect probability.

P_{cp}= 0.05 to 0.2 (Probability that a single crosspoint is non programmable)

c= number of columns in PLA

r= number of rows in PLA (After introducing redundancy, total crosspoints become n= c*r*4)

P_{on}= Probability that a given crosspoint-quad is to be programmed

Each product/sum term to be programmed has to follow a certain "path" through each section of the PLA. The failure of a product/sum term would mean that there is AT LEAST ONE quad that did not get programmed.

A quad being non-programmable has a probability equal to fourth power of the probability, that two consecutive crosspoints are defective. If we have "n" crosspoints, and the probability of occurrence of defective crosspoint is "P_{cp}", the expected number of defective crosspoints is given by (1),

$$d = \text{integer}\ (n * P_{cp}) \qquad (1)$$

The probability of a given Quad of crosspoints being defective is:

$$P\ (\text{defective quad}) = (n-4)\ C\ (d-4) / nCd$$

Where ${}^{n}C_{d}$ is the number of combinations of "d" elements among "n" elements, and is given by

$$nCd = n! / ((n-d)!\ d!)$$

Now, the probability that a given programmable quad is to be programmed and it is defective is given by,

$$P_{on} * (n-4)\ C\ (d-4) / nCd$$

This applies to all the quads in the same column. We know that the total number of quads in a column is same as the number of rows in the device. Therefore, the probability of finding ONE defect on a given NW column is given by expression (2)

$$P\ (1) = r * P_{on} * (n-4)\ C\ (d-4) / nCd \qquad (2)$$

The NW column is unusable if there is AT LEAST one defective quad at a location that needs to be programmed. Therefore, the probability of getting a non-programmable NW column is given by,

$$P_{dcrosspoint} = \sum_{i=1}^{r} P(i)$$

$$= \sum_{i=1}^{r} \{r * (P_{on})^{i} * (n-4i)\, C\, (d-4i)/\, nCd\} \qquad (3)$$

Expression (3) gives us the defect probability exclusively due to crosspoint-defects. As an example, for a 50x50 NanoPLA having a crosspoint defect rate of 15%, this probability is of the order of 2%, i.e. a yield of 98%. Average yield in CMOS based FPGA's is however, of the order of 83% to 89.5% [13], much lesser than that achieved at Nanoscale using DVR. The calculations given in [8] suggest that the defect density in nano-crossbars would be of the order of 15% to 20%. For DVR analysis, therefore, we analyze the yield for defect density that ranges from 10% to 20%.

5.2 AVR allocation algorithm

Adaptive Variable Redundancy is assigned to each variable using the algorithm discussed below. It can be seen in Figure 4, that the number of redundant NWs assigned depends on the number of times the variable is used. E.g. "A" in Figure 4 is used eight times and four NWs are allocated to it. On the other hand, "B" is used only three times, so only two NWs are assigned.

The following pseudo code describes the allocation algorithm.

Initialize the size of the Function Matrix F

Initialize the defect Density d=P_{def}

Initialize the probability of ON input occurrence, P (ON)

Initialize threshold

FOR *i= 1 to (size of F)*

Flex[i] =1; // Stores the redundant wires needed

LOOP: *d=d ^ flex[i];*

a[i]= P(defect for all possible defect orientations)

//assuming 'i' crosspoints are to be programmed

b[i]= P(i crosspoints to be programmed) = $^{n}C_{i}\ d^{i}\ (1-d)^{(n-i)}$

// found using Binomial Distribution

*c[i]= a[i] * b[i];*

IF (*c[i]>threshold)*

Flex[i] =Flex[i] +1;

GOTO ***LOOP****;*

END IF

END LOOP

END FOR

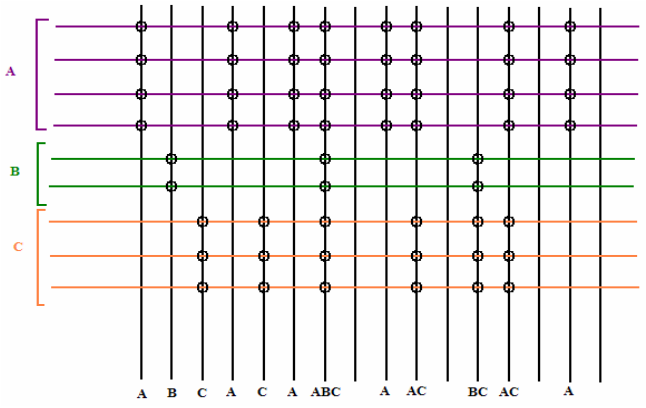


Figure 4. Illustration of AVR

6. Simulation results

6.1 Yield and time complexity in DVR

The MATLAB simulations based on the configuration sequence gives us the following results for different array sizes and defect rates. Yield results can be seen in Figure 5. The simulations are carried out by varying the following parameters:

Defect rate: 10% to 20%

PLA size: 50x50 to 500x500 and P (on) =50% to 90%

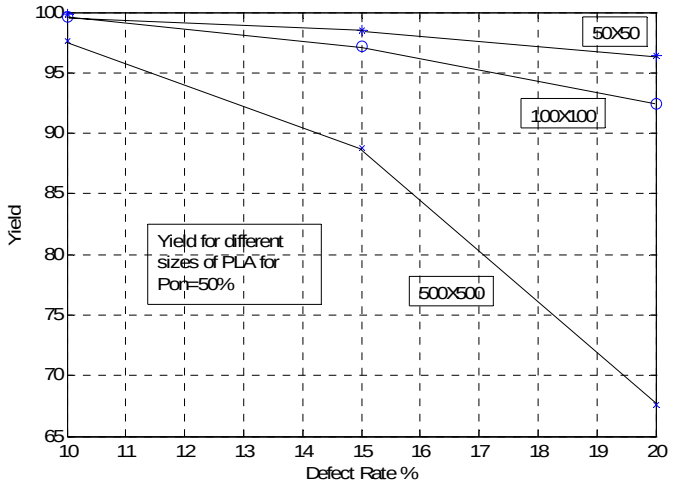
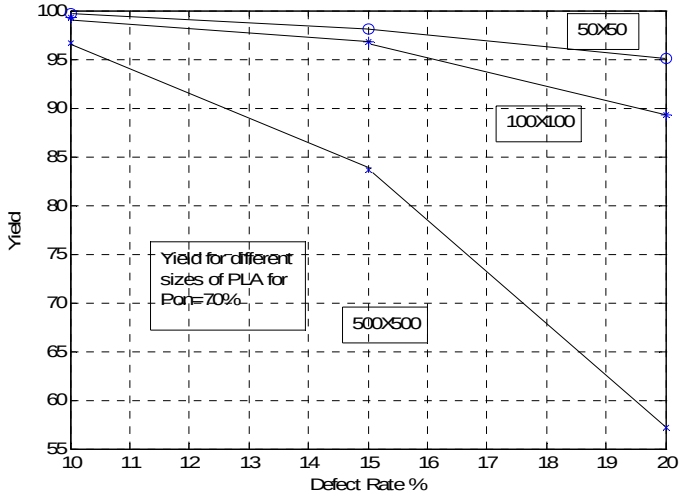


Figure 5. (a) P_{on}=50%

Yield Vs Defect Rates for different PLA sizes. It also establishes the relation between Yield and the Probability Pon.



(b) P_{on}=70%

Figure 5. contd.

Summarizing the results in [8], we have the time complexity for Greedy Heuristic Algorithm as,

$T_{c(GA)}$= O (|F| log (|F|)) + O (|F| · P_J^{-cm} · cm) (5)

Where,

$T_{c(GA)}$= Time complexity for Greedy Heuristic Algorithm

|F| = size of the array of Boolean functions

cm= maximum number of crosspoints to be programmed in a minterm

P_J= Probability that the given junction is Programmable.

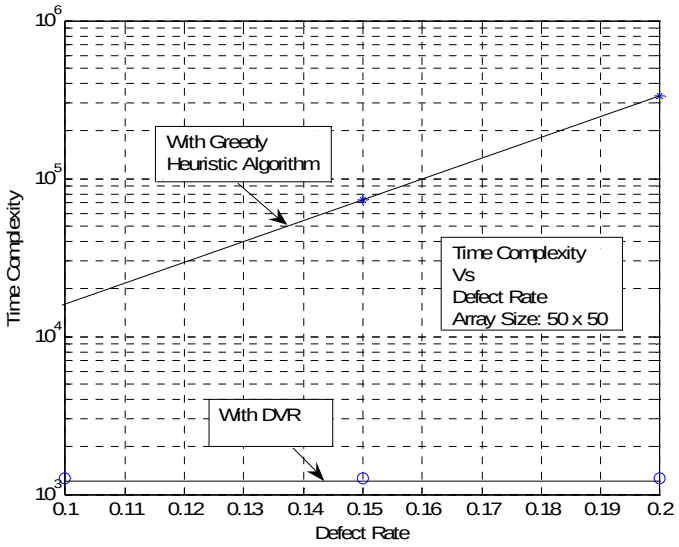
The time complexity involved in the configuration of a DVR based PLA is given as:

T_c= Total number of crosspoints to be Programmed.

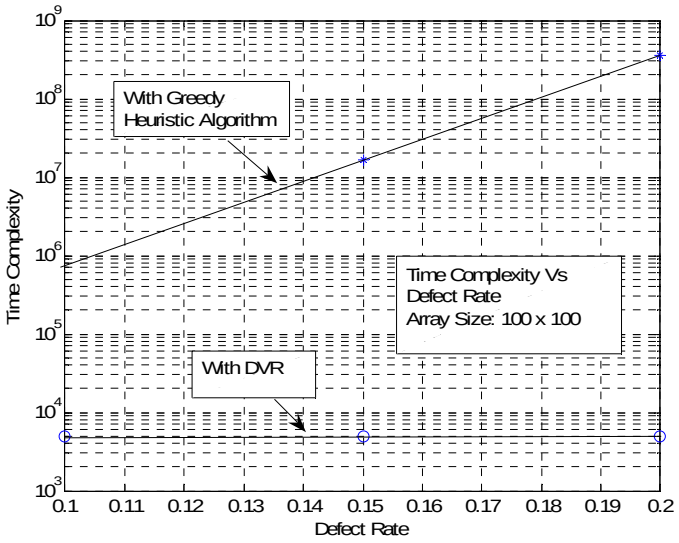
Therefore the Time complexity in DVR is given by,

$T_{c(DVR)}$ = O(r*c*P_{on}) (6)

Double Variable Redundancy is therefore observed to be distinctly advantageous in terms of yield and time complexity over Greedy Heuristic Algorithm, as seen in Figure 5 and 6.



(a)



(b)

Figure 6. Comparison between the time complexities in the PLA configuring mechanisms: Greedy Heuristic Algorithm and DVR for different Array sizes. (a) Array size 50 x 50 (b) Array size 100 x 100

6.2 Yield and area overhead in AVR

The AVR Allocation Algorithm has been developed in such a way that the redundancy allocation for a given defect rate and ON input density depends on *how many Crosspoints* in a row need to be configured. The algorithm first calculates the probability of having a defective configuration for a given number of Crosspoints to be programmed in a row. It then compares this value with a certain pre-defined threshold to iterate the amount of redundancy required. It follows that the more likely a certain combination is, the more redundancy that is allocated to it, as shown in Figure 7. More redundant resources ensure an acceptably high yield. For a typical case where P(on)= 0.4 and 20% defect density, a yield higher than 95% is shown at a cost of an area overhead approximately equal to 4.8. It can be observed in Figure 8 and Figure 9 that a lesser threshold value gives the system a greater yield at an expense of area overhead. It is noted that at higher defect rates than 40%, the yield becomes unacceptably low (seen in Figure 8) even at the expense of higher resource allocation (seen in Figure 9).

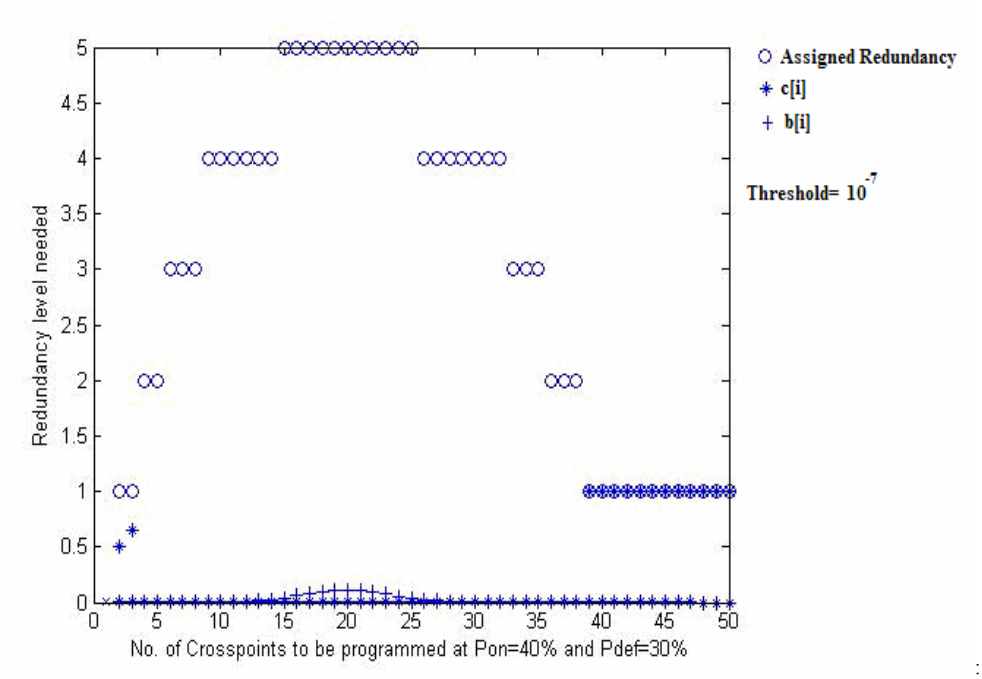
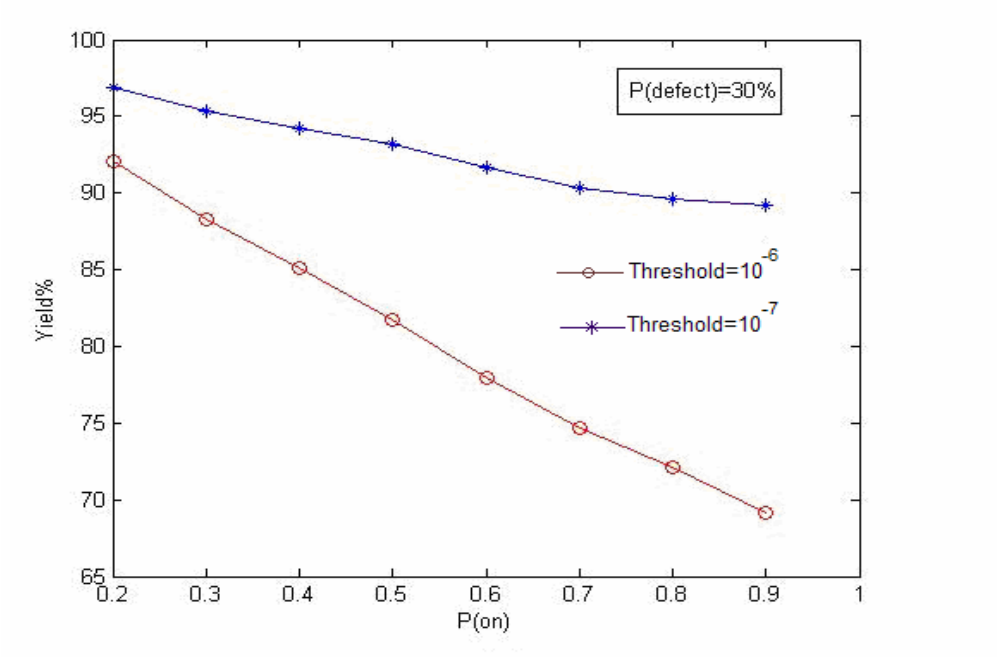
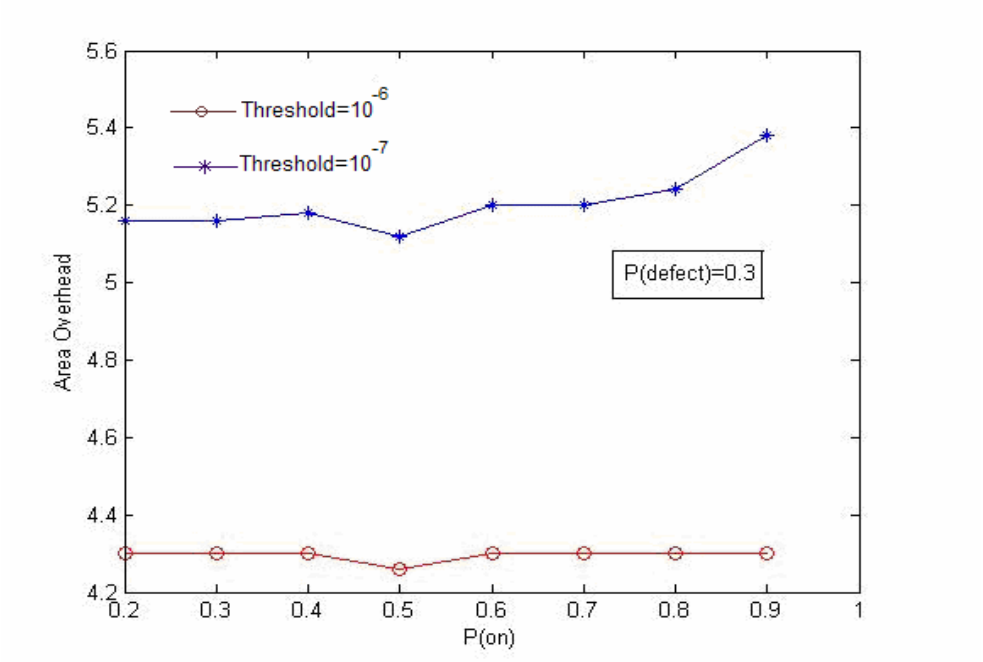


Figure 7. Redundancy levels vs. number of ON inputs

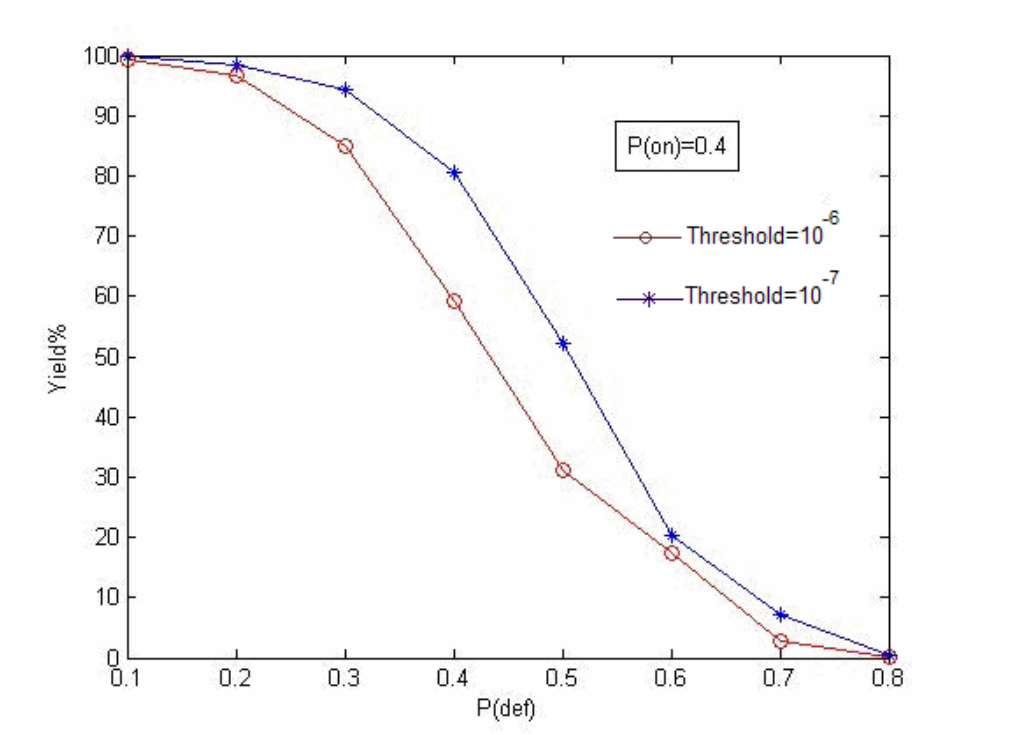


(a)

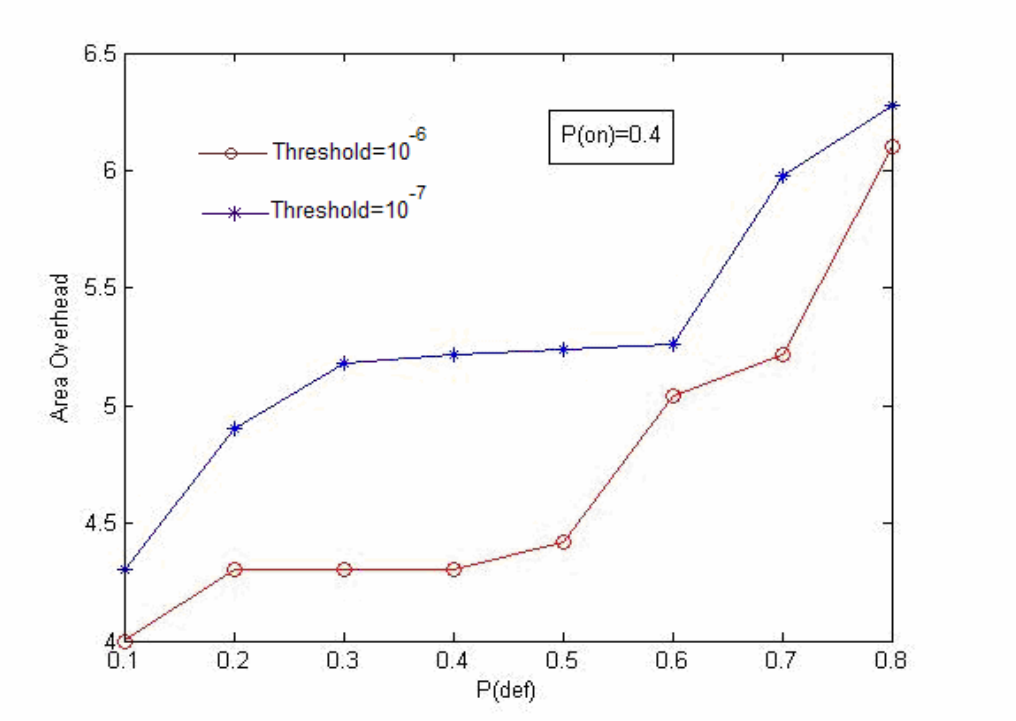


(b)

Figure 8. Yield and Area overhead vs. P(on) at constant Defect rate



(a)



(b)

Figure 9. Yield and Area overhead vs. Defect Rate at constant P(on)

7. Conclusion

Variable Redundancy greatly increases the yield in a NanoPLA configuration even with defect rates well above 20%. Double Variable Redundancy improves Yield and lowers the Time Complexity of configuration of the PLA by a great extent. Adaptive Variable Redundancy algorithm allocates redundancy based on the factors that affect the yield directly, and therefore it shows better yield results than DVR. Because AVR and DVR both use sequential configuration, no sorting algorithm is needed before configuration. The need to obtain the defect map is completely eliminated using Variable Redundancy, also eliminating the need to sort the functions. It is seen that the sorting of functions in descending order of number of ON inputs in case of the Greedy Heuristic algorithm increases its time complexity. The time complexity for the configuration for both AVR and DVR is significantly lower, at the expense of increased area overhead. It is important to note that NW PLA-based FPGA would still require much less area compared to a CMOS FPGA, even after the substantially increased area overhead of the proposed NW-PLA.

References

[1] M. Mishra and S. Goldstein, “Defect Tolerance: at the end of the Roadmap,” International Test Conference (ITC) 2003, pp. 1201-1210.

[2] Y. Cui, Z. Zhong, D. Wan, W. Wang and C. Lieber, “High Performance Silicon Nanowire Field Effect Transistors,” Nano Letters 2003, vol. 3, no. 2, pp. 149-215.

[3] S. Cha, J. Jang, Y. Choi, G. Amaratunga, D. Kang, D. Hasko, J. Jung and N. Kim, “Fabrication of a Nanoelectromechanical switch using a Suspended Carbon Nanotube,” Applied Physics Letters 2005, vol. 86, no. 8, id. 083105.

[4] A. DeHon and M. J. Wilson, “Nanowire-Based Sublithographic Programmable Logic Arrays,” International Symposium on Field Programmable Gate Arrays (FPGA’04), pp. 123-132.

[5] A. DeHon, S. Goldstein and P. Kuekes, “Nonphotolithographic, Nanoscale Memory Density Prospects,” IEEE Transactions on Nanotechnology, 2005, vol. 4, no. 2, pp. 215-221.

[6] T. Hogg and G. Snider, “Defect-tolerant logic with Nanoscale Crossbar Circuits,” HP Labs, 2004 URL:http://www.hpl.hp.com/research/idl/papers/molecularAdder/circuits.pdf

[7] S. C. Goldstein and M. Budiu, “NanoFabrics: Spatial Computing Using Molecular Electronics,” International Symposium on Computer Architecture (ISCA) 2001, pp. 178-191.

[8] H. Naeimi, “A Greedy Algorithm for Tolerating Defective Cross points in Nano PLA Design,” MS Dissertation 2005, California Inst. of Technology, Pasadena, California.

[9] C Jeffery and R. J. O. Figueiredo, “Hierarchical Fault Tolerance for Nanoscale Memories,” IEEE Transactions on Nanotechnology, July 2006, vol. 5, no. 4, pp. 407-414.

[10] M. Tehranipoor, “Defect Tolerance for Molecular Electronics-Based NanoFabrics using Built-In Self-Test Procedure,” Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT) 2005, pp. 305- 313.

[11] W Rao, A. Orailoglu and R Karri, “Nanofabric Topologies and Reconfiguration Algorithms to Support Dynamically Adaptive Fault Tolerance,” Proceedings of the 24th IEEE VLSI Test Symposium (VTS) 2006, pp. 214-221.

[12] M. V. Joshi and W. K. Al-Assadi, “Nanofabric PLA architecture to minimize Configuration Time Complexity,” Proceedings of IEEE Region 5 Technical Conference, 2007, Fayetteville, Arkansas.

[13] P. Maidee, and K. Barzagan, "Defect-tolerant FPGA architecture exploration," International conference on Field Programmable Logic and applications (FPL), 2006.

PAPER II

A BIST APPROACH FOR CONFIGURABLE NANOFABRIC ARRAYS

Mandar V. Joshi, Waleed K. Al-Assadi
Electrical and Computer Engineering
Missouri University of Science and Technology, Missouri, U.S.A 65409
Email: mvjvx8@mst.edu, waleed@mst.edu

***Abstract* — The sustainability of Moore's Law has faced challenges as the physical limits of transistor miniaturization have begun to appear. CMOS scaling with current technology setup introduces new difficulties that include device parameter variation and increased leakage current. New technologies are therefore being evaluated for their feasibility as replacement for present CMOS. Ultra-miniaturized Diodes and Field–Effect Transistors with pitches of well below 30 nm have demonstrated effectiveness. These structures are synthesized in 2-D arrays of Silicon nanowires (SiNWs) or Carbon Nanotubes (CNTs). Memory and Logic cores using these technologies that use the configurable junctions in two-dimensional crossbars of CNTs have been proposed. These devices, however, exhibit a significantly higher number of transistor defects and, in turn, faults, than present technology. Configuring these devices in the presence of defects demands an overhead in terms of area and programming time. It also imposes the challenge of obtaining acceptably high yield by tolerating these defects. This work proposes a Built-in Self Test (BIST) approach to test crossbars for a defined set of faults. The BIST can classify the different programmable elements in the crossbars as non-defective or defective with a certain fault type. The logic synthesis can then configure the crossbar by avoiding these defective elements.**

***Index Terms*— Crossbar, Nanofabrics, Nanowires, BIST, Recovery**

I. INTRODUCTION

Bottom-up techniques that enable us to fabricate circuits of molecular dimensions, exploiting mechanical and electronic properties of CNTs and SiNWs have been suggested for digital systems [1] [2] [3]. A junction of two SiNWs or CNTs is termed a "crosspoint." A complete NanoPLA architecture uses the "stochastic addressing" developed by De Hon and takes advantage of programmable crosspoints [4]. All such architectures assume a certain assembly of NWs or CNTs, but crossbar (also referred to as "nanofabric") architectures are the most common of all. The key idea of configurability is that each NW can be uniquely addressed with a very high probability by introducing redundancy in terms of the number of wires. Redundancy ensures that even in the presence of a very high number of defects (nominally 13% to 20%); the desired digital circuits can be synthesized.

Our previous work pertaining to PLA architectures introduced the new concept of introducing fixed [5] or variable nanowire (NW) redundancy [6] to obtain higher yields than most of the other proposed logic blocks in a PLA. In this work, however, we invoke a higher level of abstraction in which we divide our crossbar into a number of Programmable Blocks (PBs) equal in size to each other and equidistant, as shown in Figure 1. To build a Built-in Self Test for such a crossbar, researchers have developed self-testing algorithms. In the BIST procedure, they configure the nanofabric array in a defined sequence of macros (logic circuits) and observe the outputs of neighboring logic blocks to find and analyze defects. The performance of such BIST procedures is governed by the types of configurations they need and the number of configurations in which the entire nanofabric array is checked for defects. Each PB can be thought of as a PLA block, which has a rich interconnect. Each PB is either defective or defect free for a given configuration. If a PB is found to be defective, BIST techniques will tag it and the synthesizer will not be allowed to use it for the corresponding configuration. The entire nanofabric array gets divided into sets of blocks that act either as the Blocks Under Test (BUTs) or Checker Blocks that test the

validity of the outputs of the BUTs. A NanoBlock is a configurable block and a SwitchBlock is used as interconnect between different NanoBlocks.

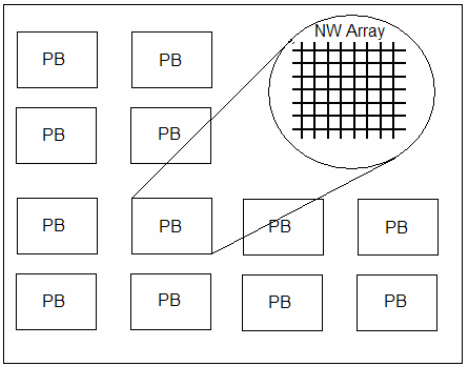


Fig.1 nanofabric PLA as an Array of Programmable Blocks

II. OVERVIEW OF PREVIOUS BIST TECHNIQUES PROPOSED FOR NANOFABRICS

M. Tehranipoor proposed a Built-in Self Test procedure for nanofabrics [7]. In this procedure, the nanofabric is split into NanoBlocks and SwitchBlocks that perform logical and routing operations, respectively. In the self-test, each NanoBlock is configured either as a *Pattern Generator (PG)* or a *Response Generator (RG)*. A *Test Group* is created using a set of PGs, RGs and SwitchBlock(s) between the two. Test Groups of the same kind form a *Test Architecture,* or *TA*, as shown in Figure 2. TAs are generated based upon the direction of fault in each NanoBlock and SwitchBlock. During the test sequence, every NanoBlock is configured as both a PG and an RG in different Test Architectures. The NanoBlock configured as a PG tests itself and generates the test pattern for RG. An external device is needed to program the NanoBlocks and read the RGs' responses. $3n^2/4$ devices are configured. In the test configurations, stuck-at, stuck-open, forward biased and reverse biased diode and AND & OR bridging faults are targeted. A specific configuration of PGs and RGs is used for every type of fault. If the size of the RG is K x K, it is estimated that 8K + 5 configurations are needed to provide 100% fault coverage. The main

disadvantage of this scheme is that it requires an external tester. Moreover, faults in the SwitchBlocks are not considered.

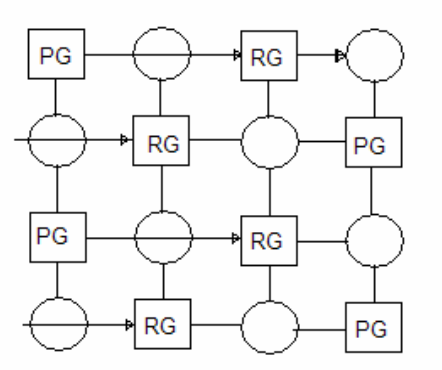
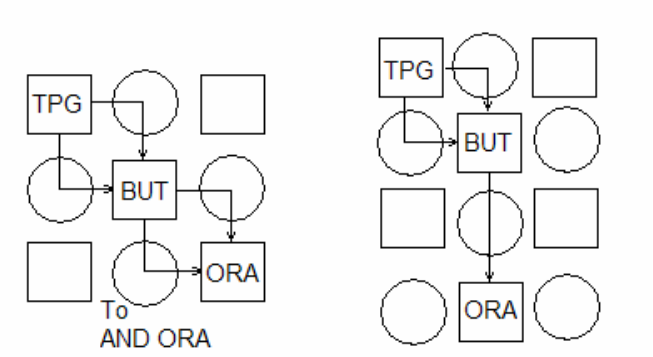


Fig. 2: Test Architecture for BIST

Z. Wang proposed a BIST approach that is similar in many ways to the approach discussed above [8]. In this BIST procedure, NanoBlocks can be configured as Test Pattern Generators (TPGs), Block Under Test (BUTs) or Output Response Analyzers (ORAs) as shown in Figure 3. These blocks, along with the corresponding SwitchBlocks, comprise a TG (Test Group) similar to one discussed in [7]. In a TG, the TPG generates the testing patterns for a BUT and ORAs examine the BUT output response. A TG and a set of Fault Detecting Configurations (FDCs) are used where different BUT faults can be tested. They provide 100% fault coverage for stuck-at, stuck-open, bridging and connection faults. The metric defined for the quality of the test is called "recovery," which is defined as the ratio of non-defective blocks identified to the actual number of non-defective blocks. A BUT is declared defect-free only if it operates correctly under all FDCs. The separate test procedure for each type of TG needed to achieve full fault coverage results in three partial defect maps. The types of FDCs used in the test sequence are identical to test configurations used in [7]. A NanoBlock is defect free when it bypasses all three partial defect maps. It is assumed in the test sequence that ORAs can be read out using the mechanism that configured the fabric. The test results show that a 10x10 nanofabric with a 10% defect density yields a recovery of 76.9%.



(a) AND-OR ORA (b) OR ORA

Fig. 3: Different Test Groups Implementing FDCs

Another BIST approach called CAEN-BIST was proposed by J. Brown [9]. CAEN stands for Chemically Assembled Electronic Nanotechnology and in this case, refers to programmable nano-arrays. Their research involved a behavioral modeling of SwitchBlock and NanoBlock. A nanofabric would consist of numerous NanoBlocks of size k x k each. A k x k array can have (2k-1) inputs and one output or (2k-2) inputs and two outputs, and so on, if the I/Os are introduced at the block's northwest boundary. To inject defects, a random sequence of bits applied at the block's horizontal and vertical inputs simulate stuck-line and connection faults. Bridging faults can also be introduced by arbitrarily implementing AND/OR logic functions between the wires. Once obtained, this defect map is used to further estimate the test's accuracy. One more defect map is obtained after applying the test, and a comparison between the two maps is used to find the recovery. The CAEN-BIST algorithm not only enables the nanofabric to test itself, but also stores the results of the test internally. Because the defect density of nanofabrics is very high, BIST algorithms cannot be used internally and an external tester is required to generate the test patterns and check responses. A walking sequence of 1s and 0s is applied to the BUT during a test. The response is stored in the neighboring NanoBlock. This technique eliminates the possibility of a defective block marking itself as non-defective. The BIST works in a wave-like

manner, creating tests for diagonal elements. There are square root of "n" diagonals to be tested, where "n" represents the total number of blocks and k^2 patterns are applied for each block.

III. NEW APPROACH FOR NANOFABRIC BIST

A. Test Configuration

In the new approach, we model the nanofabric as a set of NanoBlocks similar to those in [7]. The types of blocks that can be targeted are single stuck-at and bridging faults. A test architecture consists of three blocks: two BUTs and one Comparator (denoted as "C"), as shown in Figure 4. Therefore, all the NanoBlocks take part in each test, and the test for a particular set of faults is completed in two configuration sequences. The BIST configures the blocks externally using the device's I/O interface.

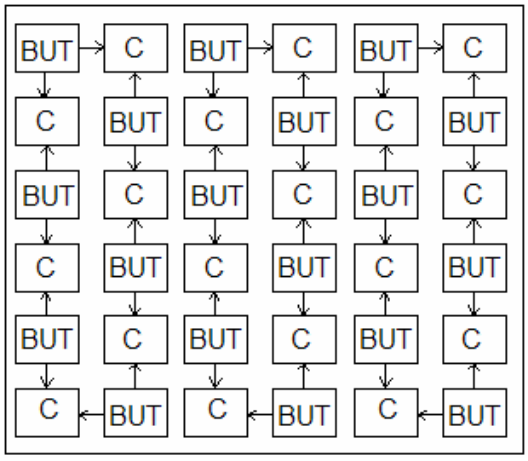


Fig. 4: Test Configuration for Proposed BIST

Since the defect rate is of the order of 10%-15%, it is assumed that the probability of two defective BUTs being compared by the same comparator is very low. Every block in the nanofabric has the ability to store the result of the comparison. It is assumed that a comparator

always generates correct results storing a “0” for a successful comparison and a “1” for an unsuccessful comparison. In other words, the BIST remembers which comparison went wrong and reports it to the external tester. This helps generate an intermediate defect map called the “Raw Defect Map” and, in turn, the final defect map. A test is run for each type of fault to be targeted to create Raw Defect Maps for corresponding faults. Combining all the Raw Defect Maps gives the final defect map, which the logic synthesizer can use to synthesize a given logic by avoiding the defective blocks in the nanofabric.

B. BIST Algorithm and Illustration

A block is declared fault-free only if it does not manifest any of the faults targeted. If the BIST can generate tests for “f” number of faults, 2f sets of test vectors are needed to test all blocks. These vectors create a Raw Defect Map for every fault targeted. All Raw Defect Maps are then read together to create the final defect map.

The following algorithm describes the BIST sequence:

FOR *i=1 to* ***types of faults targeted ---BIST STEP 1 to f***

Generate test vectors for fault (i) in the first fault

Detection loop;

Generate Raw_Defect_Map(i);

END FOR

Initialize final_defect_map=NULL;

FOR *i=1 to types of faults* ***------BIST STEP f+1***

Final_defect_map = final_defect_map +

Raw_Defect_Map(i);

END FOR

During the initial "f" steps, the BUTs are configured for a certain logic that detects the targeted fault.

Figure 5 illustrates the two possible Fault Detection Loops into which the whole nanofabric can be divided. To enable the conversion of every block in a nanofabric into a BUT, two such loops are needed. A Comparator compares the outputs of two neighboring BUTs and stores the results of comparison. Differences in the outputs of the two BUTs indicate the presence of a fault. At this time, the Comparator does not know which of the BUTs possesses the fault. Thus, the Comparator marks both BUTs as defect *suspects*. When the next Fault Detection Loop is applied, the actual faulty member is identified and is marked as "1." The Raw Defect Map is updated accordingly. This process is illustrated in Example 1.

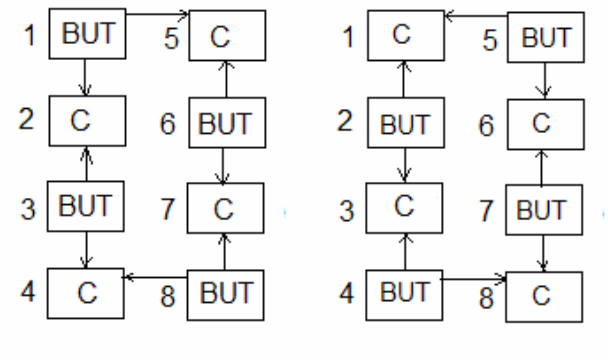


Fig. 5: An Illustration of Fault Detection Loops

Raw Defect Maps are obtained by sequencing through all the possible fault types. A final defect map is obtained by combining all of their 0s. The presence of a "1" essentially signals the

inability of the corresponding block to be configured for a given logic function. The following example illustrates how a defect map is obtained using our BIST approach.

C. Example 1

Assumptions made:

1. The fault universe has 2 faults: F1 and F2.
2. The size of the nanofabric is 4 x 4 blocks.
3. Blocks 3 and 11 have fault F1, and Block 10 has fault F2, as shown in Figure 6.
4. All the comparisons generate the correct results.

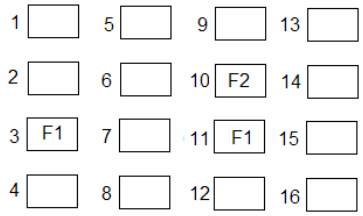


Fig. 6: A Case Supporting Example 1

Fault Simulation:

Each fault will be simulated in one-hot two bits. MSB will correspond to the presence/absence of fault F1, and LSB will correspond to the presence/absence of fault F2. The presence of a fault will set the corresponding bit. Therefore, in the simulated fault list following is obtained:

B3: "10" (F1 present)

B11: "10" (F1 present)

B10: "01" (F2 present)

The rest of the blocks will contain "00". Note that this encoding of faults is only done for the example in consideration. Actual BIST can encode a fault in other formats, too.

BIST Step 1

Fault F1 is targeted in this step. The configuration applied to the nanofabric is shown in Figure 7.

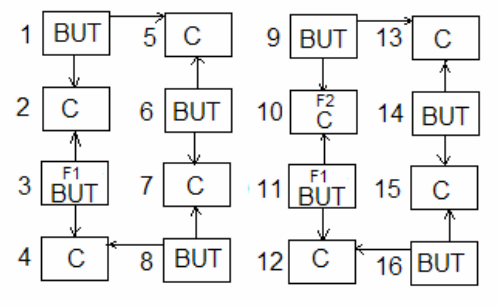


Fig. 7: Configuration to Target Fault F1

Blocks B2, B4, B5, B7, B10, B12, B13, and B15 are used as comparators to compare the outputs of their neighboring BUTs per the arrow directions.

The comparison would produce the following results:

C2: B1 or B3 has fault F1

C4: B3 or B8 has fault F1

C10: B9 or B11 has fault F1

(Although C10 is faulty, it generates the right result because the comparisons are assumed to generate the correct results.)

C12: B11 or B16 has fault F1

The remaining comparisons are successful.

Analyzing these four results, following is obtained:

B1 reported once

B3 reported twice

B8 reported once

B9 reported once

B11 reported twice

B16 reported once

Therefore, eliminating all the blocks reported once and retaining blocks reported twice as *suspects* to generate the following *"Raw Defect Map for F1."*

0 0 0 0

0 0 0 0

1 0 1 0

0 0 0 0

BIST Step 2

Fault F2 is targeted in this step. The configuration applied to the nanofabric is shown in Figure 8.

Blocks B2, B4, B5, B7, B10, B12, B13, and B15 are now used as BUTs. Their outputs are given to the respective comparators, as shown in Figure 8. The comparisons would produce the following results:

C9: B10 or B13 has fault F2

C11: B10 or B12 has fault F2

The remaining comparisons are successful.

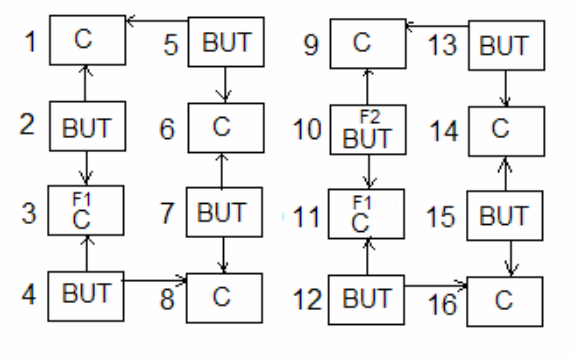


Fig. 8: Configuration to Target Fault F2

It is seen that:

B10 is reported twice

B13 is reported once

B12 is reported once

Eliminating the blocks reported once and retaining the block reported twice as a *"suspect."* Therefore, B10 is a suspect.

The *"Raw Defect Map for fault F2"* follows*:*

0 0 0 0

0 0 1 0

0 0 0 0

0 0 0 0

BIST Step 3: Obtaining the final defect map

By combining the two defect maps obtained, the final defect map as follows is obtained:

0 0 0 0

0 0 1 0

1 0 1 0

0 0 0 0

This map detects all faults. The blocks labeled “1” can be ignored by synthesis tools whenever a logic is to be implemented using the given nanofabric.

IV. RESULTS AND ANALYSIS

The coding and simulations were carried out using MATLAB for the proposed BIST on a machine with the following configuration:

AMD Turion 64 Processor 1.6 GHz

1280 MB RAM

128 KB split L1 Cache

1024 KB L2 Cache

The fault universe consisted of two to five faults at a given time. The results were obtained in terms of recovery and computation time. The Defect Density or Defect Rate was varied from 10% to 70% for all fault universes to obtain the output parameters, namely recovery and computation time.

$$\text{Recovery} = \frac{\text{Number of identified non - defective Logic Blocks}}{\text{Actual Number of non - defective Logic Blocks}} \qquad (1)$$

$$\text{Defect Density} = \frac{\text{Number of defective Logic Blocks}}{\text{Total number of Logic Blocks}} \qquad (2)$$

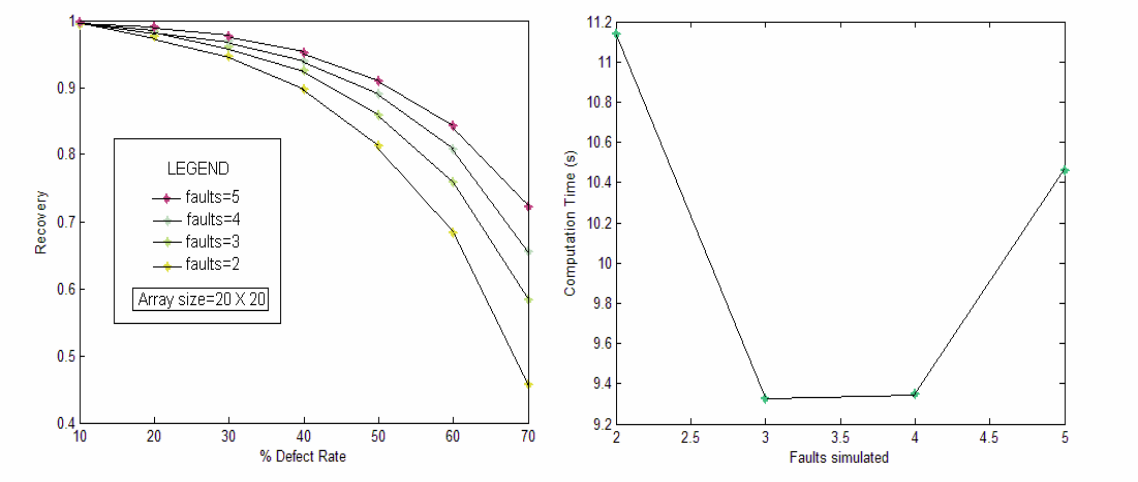


Fig. 9: Recovery and Computation time for 20 x 20 Array

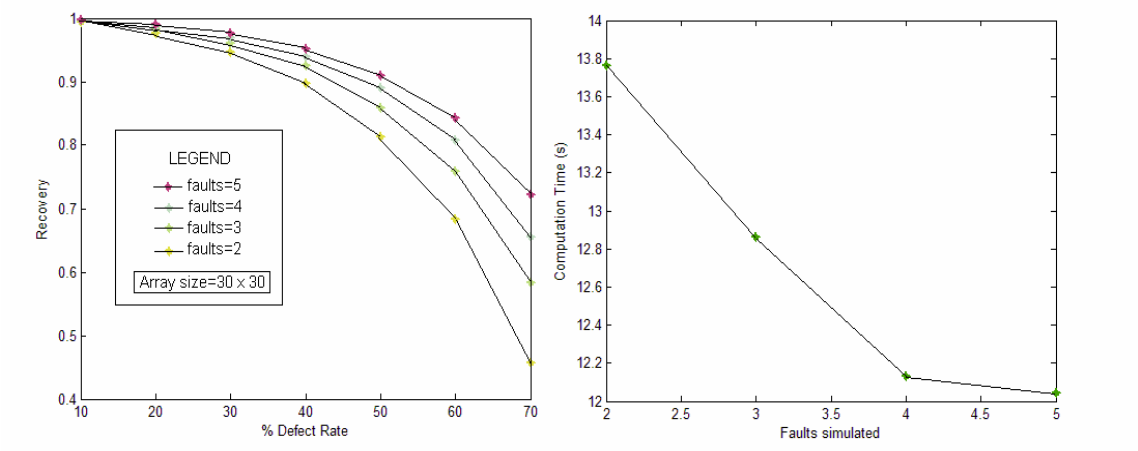


Fig. 10: Recovery and Computation time for 30 x 30 Array

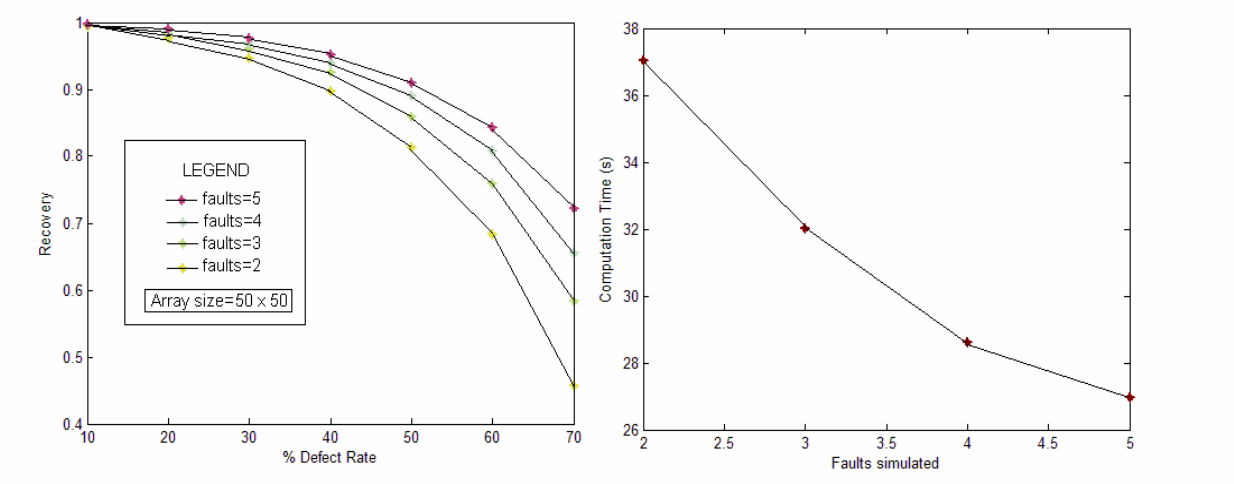


Fig. 11: Recovery and Computation time for 50 x 50 Array

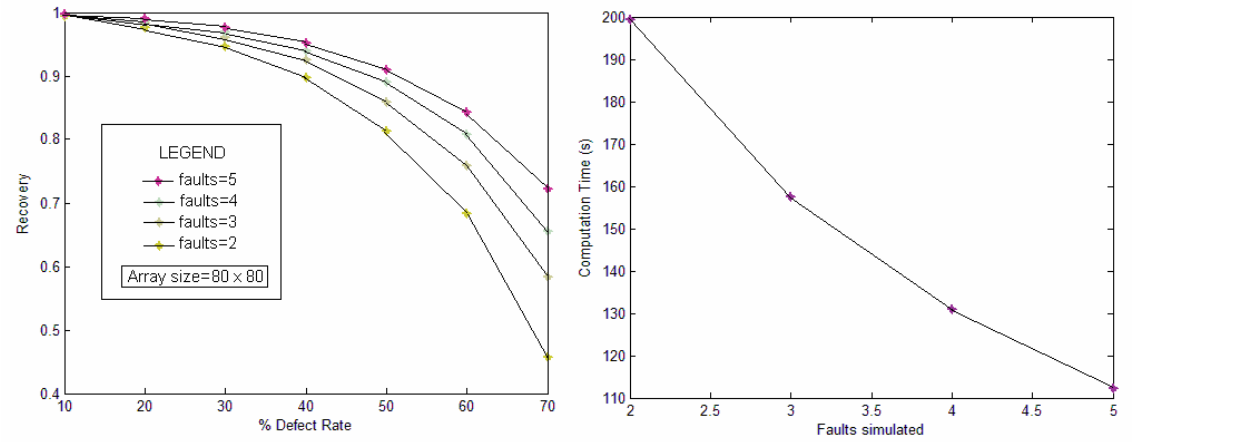


Fig. 12: Recovery and Computation time for 80 x 80 Array

A. Effect of Defect Density

As the defect density increases, the ambiguity between the blocks marked as suspects and the actual defective blocks increases. Moreover, the BIST assumes that only one of the two blocks being compared is defective. This assumption ceases to hold true for higher defect densities, resulting in lower recovery rates at higher defect densities. This trend is independent of the array sizes and, thus, results in exactly identical recovery values for all array sizes.

The results were based on different array sizes varying from 20x20 to 80x80. Figures 9-12 illustrate the recovery versus defect density. Because the recovery largely depends on the defect orientation rather than array size, it remains constant for all array sizes, as seen in figures 9-12.

B. Effect of Array size

The simulations were conducted on square arrays to maintain symmetry. Since the computational complexity has a square relationship to the size, the computation time grows exponentially as size increases, as seen in Figures 9-12. The mathematical relationship of array size and computation will be established in section 4.4. For faults simulated = 2, the effect of array size on computation time is illustrated in Figure 13.

C. Effect of Number of Faults in Fault Universe

Number of Faults plays a crucial role in terms of BIST performance, recovery, and computation time. As the fault universe develops more faults, more Raw Defect Maps are generated. At a constant defect density, as the number of Raw Defect Maps increases, the number of defects identified in each Raw Defect Map decreases substantially reducing the computation time needed to locate defects. Therefore, the computation time is reduced for more faults in the fault universe, as shown in Figures 9-12. Due to the scattered nature of faults, the ambiguity of identifying a non-defective element as defective (false positives) is reduced. This, in turn, increases the recovery and proves useful in terms of both computation time and recovery.

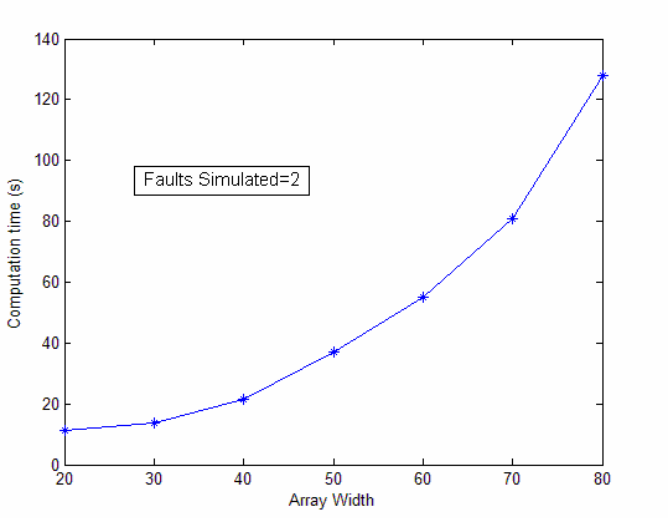


Fig. 13: Computation time Vs. Array width

D. Computation Time

The following computations are involved in the completion of self-test:

1. Configuration time for each Fault Detection Loop $=T_{cfg}$
2. Comparison time consumed by each comparator $=T_{com}$
3. Calculation time consumed by external Tester when computing the final defect map $=T_{calc}$

It follows that the configuration time, $T_{cfg,}$ is taken by each fault type and is repeated twice because there are two Fault Detection Loops per fault. Similarly, the comparison takes place twice. Given the above considerations, the time complexity "T" is given by

$$T = O((T_{cfg} \times f) + (T_{com} \times f) + T_{calc})$$

$$= O(T_1+T_2+T_{calc}) \qquad (3)$$

where

f = Number of faults in the fault universe

$T_1 = T_{cfg} \times f$

$T_2 = T_{com} \times f$

When the number of 'f' increases, the values of T_1 and T_2 increase linearly, whereas that of T_{calc} decreases rapidly due to the reasons discussed above. This increase reflects in decreased value of overall computation time. This change happens for all array sizes except 20 x 20.

The computation time gain caused by the increase in faults simulated is less than the penalty paid for comparison time for more arrays. Therefore seen that in case of the 20 x 20 array, the computation time is greater for f=5, as shown in Figure 9.

E. Comparison with previously proposed BISTs

In the previous BIST techniques discussed in section II, the BUT is tested using Test Pattern Generator and Output Response Analyzer. This follows that, out of three (or more in some cases) blocks used, only one is tested for presence of a fault. In our technique, out of every two blocks used, one is tested. Therefore it takes fewer cycles to complete the testing of the entire nanofabric array. The previous BIST techniques require different configurations for checking elements (ORA's) namely AND, OR etc. depending on the tests. Our techniques, on the other hand always require a fixed configuration (comparator) for all the checking elements. This ease of configuration helps reduce the configuration complexity of the BIST, and the external tester requires less memory to store all the configurations. It is also seen that the recovery stays constant for different sizes at a given defect rate. This is a great advantage, as bigger arrays can be tested effectively and quickly.

V. CONCLUSION

BIST technique discussed here is substantially faster than the previously proposed BIST techniques. Only two configurations are needed to cover all the NanoBlocks to test a particular block, whereas the other techniques require a set of configurations depending on size and the type of fault targeted. In our technique, the number of blocks tested at any time is a constant and equals half the total blocks. This technique is much more area efficient because two of the three NanoBlocks configured in our technique are tested at a time and there is no need to dedicate two blocks exclusively to pattern generation and response analysis. It is flexible in terms of fault analysis. The fault set can be previously defined, and the configurations can be developed based on each fault. Two NanoBlocks in the test architecture are tested at the same time. The entire nanofabric is tested in just two configuration sequences, which reduces the overall time required to test the complete fabric for a given fault. Another advantage of the new BIST approach is its constant recovery rate with respect to array size. Scaling of arrays without loss of recovery becomes possible.

REFERENCES

[1] M. Mishra and S. Goldstein, "Defect Tolerance at the End of the Roadmap," International Test Conference (ITC) 2003, pp. 1201-1210.

[2] Y. Cui, Z. Zhong, D. Wan, W. Wang and C. Lieber, "High Performance Silicon nanowire Field Effect Transistors," Nano Letters 2003, vol. 3, no. 2, pp. 149-215.

[3] S. Cha, J. Jang, Y. Choi, G. Amaratunga, D. Kang, D. Hasko, J. Jung and N. Kim, "Fabrication of a Nanoelectromechanical Switch Using a Suspended Carbon Nanotube," Applied Physics Letters 2005, vol. 86, no. 8, id. 083105.

[4] A. DeHon and M. J. Wilson, “nanowire-Based Sublithographic Programmable Logic Arrays,” International Symposium on Field Programmable Gate Arrays (FPGA’04), pp. 123-132.

[5] M. V. Joshi and W. K. Al-Assadi, "nanofabric PLA architecture to minimize configuration time complexity," to IEEE Region 5 Technical Conference, April 2007.

[6] M. V. Joshi and W. K. Al-Assadi, "nanofabric PLA architecture with Flexible nanowire-redundancy,” NSTI Nanotech conference, May 2007.

[7] M. Tehranipoor, “Defect Tolerance for Molecular Electronics-Based NanoFabrics Using Built-In Self-Test Procedure,” Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), 2005.

[8] Z. Wang and K. Chakraborty, “Built-in Self-Test of Molecular Electronics-Based nanofabrics,” Proceedings of European Test Symposium (ETS), 2005.

[9] J. G. Brown and R. D. Blanton, “CAEN-BIST: Testing the NanoFabric,” Proceedings of International Test Symposium (ITC), pp. 462-471, 2004.

VITA

Mandar V. Joshi was born in Nasik, India, on June 27, 1982. In July 2004, he received his Bachelor of Engineering with First Class in Electronics Engineering from the University of Mumbai, India. He joined Don Bosco Institute of Technology, Mumbai, India in July 2004 as a lecturer in the department of Computer Engineering where he taught undergraduate level courses in digital electronics. After one and half years of lectureship in India, he got an offer from UMR to be a research assistant for M.S. degree. From September 2007 to January, 2008, he did internship in Intel's Server Platforms Group, Chandler, Arizona as a Front End Validation engineer and took a part into the pre-silicon validation of Storage Controller Chip. In May 2008, he received his M.S. in Computer Engineering from Missouri University of Science & Technology (formerly University of Missouri-Rolla), Rolla, Missouri, USA.