Spring 2008

Asynchronous nanowire crossbar architecture for manufacturability, modularity and robustness

Ravi Bonam

Department:

ASYNCHRONOUS NANOWIRE CROSSBAR ARCHITECTURE FOR MANUFACTURABILITY, MODULARITY AND ROBUSTNESS

by

RAVI KIRAN BONAM

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:

Minsu Choi, Advisor
Waleed Al-Assadi
Theodore McCracken

PUBLICATION THESIS OPTION

This thesis consists of three articles that have been published or submitted for publication as follows:

The first paper presented in pages 2-15 entitled “Clock-Free Nanowire Crossbar Architecture based on Null Conventional Logic (NCL)”, was published in the PROCEEDINGS OF IEEE INTERNATIONAL CONFERENCE ON NANOTECHNOLOGY, 2007.

The second paper presented in pages 16-26 entitled “Defect-Tolerant Gate Macro Mapping and Placement in Clock-Free Nanowire Crossbar Architecture”, was published in the PROCEEDINGS OF INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS, 2007.

The third paper presented in pages 27-47 entitled “Evaluation of Defect-Tolerant Mapping and Placement Techniques for Asynchronous Nanowire Crossbar Architecture”, is submitted to the ACM JOURNAL ON EMERGING TECHNOLOGIES IN COMPUTER SYSTEMS, 2008.

ABSTRACT

The end photolithography as a driver of Moore's Law is expected to end in seven to twelve years. Numerous nanoscale logic devices, crossbar-based computing architectures based on Carbon Nanotubes (CNTs) and Silicon Nanowires (SiNWs) have been proposed and these emerging nanotechnologies are expected to continue the technological revolution. To be a viable technological paradigm, several intrinsic issues with the nanowire crossbar architecture, such as high defect density and various parametric variations caused by imperfect nanoscale fabrication has to be overcome. In this work, we have proposed and validated a new asynchronous nanowire crossbar architecture based on Null Convention Logic (NCL) to address aforementioned issues with its clocked counterpart. Since the newly proposed asynchronous architecture does not need a complex clock distribution network and is free from all timing-related failure modes and can be designed with much less timing analysis, it is anticipated to enhance the manufacturability, modularity and robustness of the system.

This thesis is organized into three papers, describes the proposed architecture and evaluates related mapping and placement algorithms.

The first paper describes the Asynchronous Crossbar Architecture in detail with illustrations of the Programmable Gate Macro Block (PGMB) which is complementary to a threshold gate in Null Conventional Logic (NCL).

The second and third papers focus on four different mapping and placement techniques which are evaluated on the basis of programmability. The algorithms were subject to extensive parametric simulations and their programmability yields are illustrated.

ACKNOWLEDGMENTS

It gives me great pleasure to thank all the people who have supported me and made this thesis possible. I would like to thank Dr. Minsu Choi, who has been a great advisor throughout my Master's program and it has been a pleasure working with him. He has been a continuous source of motivation and has helped me develop my skills.

I would like to express my sincere gratitude to Dr. Waleed Al-Assadi who is serving as a committee member. I have enjoyed his Introduction to VLSI course which has been a great source of knowledge. I am grateful to my other committee member, Dr. Theodore McCracken, with whom I have taken three courses which helped me gain tremendous amount of knowledge and hands-on experience. I would like to thank Dr. Shoukat Ali and Dr. Chang-Soo Kim who have taught me two courses. I am grateful to Dr. Kurt Kosbar for providing me a Teaching Assistantship position in the Department. I am grateful to the department secretary, Mrs. Regina Kohout who has guided me through departmental obstacles and paperwork.

I give sincere thanks to Yadunandana Yellambalase, Shikha Chaudhary and Yong-Bin Kim for associating with me in research and publications.

I would like to also thank my friends and roommates who have supported me continuously throughout my Degree program and provided me a refreshing environment.

Most importantly, I would like to thank my parents, Prabhakara Rao, Kamala, my sister, Anupama and my cousins, Benerji, Madhavi, Jaya whose continuous support made this degree possible.

TABLE OF CONTENTS

Page

PUBLICATION THESIS OPTION iii

ABSTRACT iv

ACKNOWLEDGMENTS v

LIST OF ILLUSTRATIONS viii

LIST OF TABLES x

SECTION

1. INTRODUCTION 1

PAPER

I. CLOCK-FREE NANOWIRE CROSSBAR ARCHITECTURE BASED ON NULL CONVENTIONAL LOGIC (NCL) 2

Abstract 2

1. INTRODUCTION 2

2. PRELIMINARIES AND REVIEW 4

2.1. Null Conventional Logic 4

2.2. Asynchronous Crossbar Architecture and its Advantages 7

3. PROPOSED ARCHITECTURE 9

3.1. Programmable Gate Macro Block (PGMB) 9

3.2. Physical Structure 10

4. IMPLEMENTATION OF ONE BIT FULL ADDER 11

5. CONCLUSION 11

6. REFERENCES 14

II. DEFECT-TOLERANT GATE MACRO MAPPING AND PLACEMENT IN CLOCK-FREE NANOWIRE CROSSBAR ARCHITECTURE 16

Abstract 16

1. INTRODUCTION 17

2. PROPOSED MAPPING AND PLACEMENT TECHNIQUES 18

2.1. Defect Unaware Approach 18

2.2. Defect Aware Approach 19

3. PARAMETRIC SIMULATION RESULTS 21

4. FUTURE WORK AND CONCLUSION 25
5. REFERENCES 25
III. EVALUATION OF DEFECT-TOLERANT MAPPING AND PLACEMENT TECHNIQUES FOR ASYNCHRONOUS CROSSBAR ARCHITECTURE 27
Abstract 27
1. MAPPING AND PLACEMENT TECHNIQUES 28
1.1. Defect Unaware Approach - Shift Algorithm 28
1.2. Defect Unaware Approach - Modified Shift Algorithm 29
2. PARAMETRIC SIMULATION RESULTS 32
2.1. Defect Unaware Approach 32
2.2. Defect Unaware Approach - Shift Algorithm 33
2.3. Defect Unaware Approach - Modified Shift Algorithm 36
2.4. Defect Aware Approach 37
3. CONCLUSION AND FUTURE WORK 41
4. REFERENCES 42
APPENDIX 48
VITA 75

LIST OF ILLUSTRATIONS

Figure Page

PAPER I

1. Basic Structure of PGMB 9
2. TH23 realized on a PGMB.....9
3. 1-bit adder in NCL.....12
4. TH34w2 realized on PGMB.....12
5. 1-bit adder using proposed architecture.....13
6. NCL one bit register on proposed architecture 13

PAPER II

1. Defect Unaware Approach Control Flow 19
2. TH23 represented in 6 different ways.....20
3. Defect Aware Approach control flow.....22
4. Programmability of various gates at different defect rates – Defect Unaware Approach 23
5. Programmability of various gates at different defect rates – Defect Aware Approach.....23
6. Programmability at varying dimensions of PGMB for various defect rates – Defect Unaware Approach 24
7. Programmability at varying dimensions of PGMB for various defect rates – Defect Aware Approach.....24

PAPER III

1. Defect Unaware with Circular Shift control flow 30
2. Defect Unaware with Circular Shift of AND and OR planes control flow 31
3. Variation of Programmability with change in PGMB dimensions for various gates at 5 percent defect rate - Defect Unaware Approach 33
4. Variation of Programmability with change in PGMB dimensions for various gates at 10 percent defect rate - Defect Unaware Approach 34
5. Variation of Programmability with change in PGMB dimensions for various gates at 5 percent defect rate - Defect Unaware- Shift Approach 34

6. Variation of Programmability with change in PGMB dimensions for various gates at 10 percent defect rate - Defect Unaware- Shift Approach.....35

7. Variation of Programmability with change in PGMB dimensions for various gates at 15 percent defect rate - Defect Unaware- Shift Approach.....36

8. Variation of Programmability with change in PGMB dimensions for various gates at 10 percent defect rate - Defect Unaware- Modified Shift Approach 37

9. Variation of Programmability with change in PGMB dimensions for various gates at 15 percent defect rate - Defect Unaware- Modified Shift Approach 38

10. Variation of Programmability with change in PGMB dimensions for various gates at 20 percent defect rate - Defect Unaware- Modified Shift Approach.....38

11. Variation of Programmability with change in PGMB dimensions for various gates at 15 percent defect rate – Defect Aware Approach 39

12. Variation of Programmability with change in PGMB dimensions for various gates at 20 percent defect rate – Defect Aware Approach 40

13. Variation of Programmability with change in PGMB dimensions for various gates at 30 percent defect rate – Defect Aware Approach 40

LIST OF TABLES

Table Page

PAPER III

1. Average Column shifts for various gates at different defect rates - Defect Unaware – Shift Approach 45

2. Average AND plane (row) shifts for various gates at different defect rates - Defect Unaware – Modified Shift Approach 46

3. Average PGMB column shifts for various gates at different defect rates - Defect Unaware – Modified Shift Approach 47

1. INTRODUCTION

Novel nanotechnologies are being proposed to replace their CMOS counterparts and nanowire crossbar architecture is one of the most promising paradigms. These architectures are similar to their CMOS counterparts due to their synchronous nature (i.e., the components in the circuit require a clock to run). For these newer architectures to be successfully conceived by commercial manufacturers they should exhibit significant number of advantages over the present day technologies.

The clock is an important part of a circuit in any computer architecture system and is a source of considerable issues which are to be addressed to ensure the continual of advancement in circuit technologies. Null Conventional Logic (NCL) is a delay-insensitive asynchronous paradigm which integrates data and control into a single signal and eliminates the need of a clock.

This thesis spotlights the dawn of a promising new nanowire crossbar architecture, the Asynchronous crossbar architecture, in the form of three different articles. It combines the reduced size of the nanowire crossbar architecture with the clock-free nature of Null Conventional Logic, which are the primary advantages.

The first paper explains the proposed architecture with illustrations, including the design of an optimized full adder. This architecture has an elementary structure termed as a Programmable Gate Macro Block (PGMB) which is analogous to a threshold gate in NCL. The other two papers concentrate on mapping and placement techniques which are important due to defects involved in crossbars. These defects have to be tolerated and logic has to be routed appropriately for successful functioning of the circuit.

Paper I

CLOCK-FREE NANOWIRE CROSSBAR ARCHITECTURE BASED ON NULL CONVENTIONAL LOGIC (NCL)

Ravi Bonam, Shikha Chaudhary, Yadunandana Yellambalase and Minsu Choi
Dept of ECE, University of Missouri-Rolla, MO, USA

***Abstract*—There have been numerous nanowire crossbar architectures proposed till date, although all of them are envisioned to be synchronous (i.e., clocked). The clock is an important part in a circuit and it needs to be connected to all the components to synchronize their operation. Considering nondeterministic nature of nanoscale integration, realizing them on a nanowire crossbar system would be quite cumbersome. Unlike the conventional clocked counterparts, a new clock-free crossbar architecture is proposed to resolve the issues with clocked counterparts in this paper, where the use of clock is eliminated from the architecture. This has been done by implementing delay-insensitive logic encoding technique called Null Convention Logic (NCL). A delay-insensitive full adder has been implemented on the proposed architecture to demonstrate the feasibility in this paper.**

Index Terms — **Nanowire crossbar, Asynchronous computing, Null conventional logic (NCL), Manufacturability, Robustness, Scalability, Defect & fault-tolerance.**

1. INTRODUCTION

The end of photolithography as the driver for Moore's Law is predicted within seven to twelve years and nanotechnologies are emerging that are expected to continue the technological revolution. Recently, numerous nanoscale logic devices have been proposed based on nanoscale components such as CNTs and SiNWs; computing architectures are also being proposed using them as primitive building blocks. One of the most promising nanotechnologies is the crossbar based architecture; a two-dimensional array

(i.e., nanoarray) formed by the intersection of two orthogonal sets of parallel and uniformly-spaced nanometer-sized wires, such as carbon nanotubes (CNTs) and silicon nanowires (SiNWs). Experiments have shown that such wires can be aligned to construct an array with nanometer-scale spacing using a form of directed self-assembly and the formed crosspoints of nanoscale wires can be used as programmable diodes, memory cells or FETs (Field-Effect Transistors); therefore, nanoscale logic devices can be realized.

Nanowire crossbars offer both an opportunity and a challenge. The opportunity is to achieve ultra-high density which has never been achieved by photolithography. The challenge is to make them *simple enough to be manufactured and reliable enough to be used in everyday computing applications*, since high-density systems consisting of nanometer-scale elements assembled in a bottom-up manner are likely to have many imperfections (much higher raw fabrication defect densities, as high as 10%, are expected [1, 2]) and parametric variations. A computing system designed on conventional design basis and top-down lithographic manufacturing would not be practical. Ultra-high density fabrication could potentially be very inexpensive if researchers can actualize a chemical self- assembly, but such a circuit would require laborious testing, repair and reconfiguration processes, implying significant overhead costs. Also, all reconfigurable computing architectures based on nanowire crossbars are commonly envisioned to be used for synchronous circuits and systems. Thus, a clock distribution network should be fabricated along with nanowire crossbars and precise timing control should be practiced to avoid all timing-related faults induced by physical design parameter variations caused by nanoscale non-deterministic assembly.

In order to be a viable nanotechnology, the nanowire crossbar based systems should be:

1. Structurally simple and scalable enough to be fabricated by bottom-up manufacturing technique,
2. Robust enough to tolerate extreme parametric variations,
3. Defect and fault-tolerant enough to overcome the extreme defect densities, aging factors and transient faults, and

4. Able to support at-speed verification and reconfiguration.

Unlike the conventional clocked counterparts, the proposed research is to propose a new asynchronous architecture for carbon nanotube (CNT) and silicon nanowire (SiNW) based reconfigurable nano computing systems and to address aforementioned issues in doing so.

The proposed asynchronous nano-architecture is based on a delay-insensitive data encoding and self-timed logic encoding scheme - therefore, it is totally clock-free. Thus, no clock distribution network is needed and all failure modes related to the timing will be also eliminated. Potential benefits from the proposed asynchronous architecture include enhanced manufacturability, scalability, robustness and defect and fault tolerance.

2. PRELIMINARES AND REVIEW

2.1 Null Convention Logic

Most of the traditional Boolean circuits that we have been using are clock driven. The clock is one of the most important parts of the circuit and is also a parameter determining the speed and performance of the circuit. All the devices in a circuit have to be connected to the clock; hence the clock network is quite cumbersome. The traditional Boolean circuits do not check for input completion at the time of evaluating an expression i.e. whether all the inputs have arrived to start computation of the expression. Hence the Traditional Boolean circuits are symbolically incomplete in terms of evaluating expressions as they are dependent on the clock. Null Conventional Logic integrates data and control into a single signal thus yielding inherently clock less, delay insensitive circuits and systems [5]. This technology uses two states, DATA and NULL, which are used for synchronizing and I/O control. DATA wave front contains the data that has to be processed by the combinational circuit. The Null wave front is a non-data value used to reset the logic gates in the circuit and is also used as a delimiter between two DATA wave fronts [5]. Circuits communicate with each other using local hand shakes which provide synchronization. The concept of global clock is removed and this in turn removes the clock network that has to be circulated inside the circuit. The removal of clock reduc-

es the power consumption and the circuit becomes data driven (i.e. data is processed as soon as it is available). In the DATA combinational evaluation period the combinational circuitry processes the data passed on by the register and the results are stored in the successive register. The successive register generates the Request for NULL signal in the DATA completion Acknowledgement period and propagates the signal to the previous register. The previous register will then transfer a NULL to the combinational circuitry which is evaluated during the NULL combinational evaluation period.

The evaluated result is passed to the successive register which then generates a Request for DATA signal. If the output of a particular gate is NULL, it does not change until and unless all the inputs to the gate are DATA. When all the inputs receive DATA then the output changes to data and remains asserted as long as all the inputs do not change to NULL. This attribute of the threshold gates helps in achieving input completeness feature enabling the circuits to function without the clock [7]. To achieve this property the inputs to the gates are to be encoded using an encoding scheme. In a dual rail encoding scheme, each bit is represented using two rails. According to the representation in the Table 1 the combination of rails (rail1, rails0) represents a single Boolean value. The value “00” is regarded as NULL state which resets the circuit and does not represent any Boolean value. The value “11” is an undefined expression in the dual rail encoding scheme. NCL uses symbolic completeness [11] of expression to achieve self-timed behavior. A symbolically complete expression is defined as an expression that only depends on the relationships of the symbols present in the expression without a reference to the time of evaluation. This is achieved by keeping the following conditions in mind [11]:

1. The input-completeness criterion, which NCL circuits must maintain in order to be self-timed, requires that the outputs of a circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA or vice versa.
2. In circuits with multiple outputs, outputs that are dependent on arrived inputs can make transition, but all outputs can change only when all inputs arrive which eliminates the possibility of a data cycle and null cycle overlapping.

3. No orphans may propagate through a gate. An orphan is defined as a wire that transitions during the current DATA wavefront, but is not used in the determination of the output. Orphans are caused by wire forks and can be neglected through the isochronic fork assumption, as long as they are not allowed to cross a gate boundary. This observability condition ensures that every gate transition is observable at the output.

The primary advantages of the use of NCL for the proposed clock-free nano-architecture are as follows:

1. Circuits are less complex and are large circuits can be designed in a bottom-up manner and integrated directly without any trouble of synchronizing each module [5].
2. In clock-driven circuits, major part of power is consumed by the clock and its network. By removing the clock from the circuit, cumulative power consumption decreases [5].
3. The use of NCL makes the circuit insensitive to delay and the circuits operate at the rate of the flow of data. The circuits can be called as delay insensitive and self timed circuits [5, 7].
4. The circuits become more reliable than the clocked circuits as the problems caused due to clock such as clock skew, race conditions etc. are eliminated [5].

There are 27 threshold gate macros that are implemented in NCL. The significance of these 27 NCL gates is that any possible expression involving two or three or four variables can be implemented using these functions. Inversion can be implemented by interchanging the rail1 and rail0 in case of the dual rail encoding scheme.

2.2 Asynchronous Crossbar Architecture and its Advantages

Using the normal crossbar architecture is similar to the conventional Boolean circuits i.e. clock has to be circulated throughout the circuit for synchronizing various blocks. The normal crossbar circuit cannot decide when to receive or release data; therefore a clock must be added to control the flow of input and output. In contrast, the asynchronous crossbar architecture would be data driven; Instructions are acted upon the moment they are available and output is available the moment it is completed. This architecture employs discrete threshold gates [5] that recognize only certain simultaneous combinations of values. Each of the gate acts as "synchronization node", making the circuit as a whole and symbolically complete. The DATA state follows the Null state and is processed by the gates and output is passed on to a register. The register contains completion circuitry that enables synchronization and checks the state of the output and generates an appropriate signal indicating the previous register to send the complementary state i.e. if the circuit is processing a Null state then the register on arrival of the output will send a request for data signal requesting for data to the previous register. The primary advantages of the Asynchronous architecture would be,

1. **Manufacturability**

Asynchronous crossbar Architecture significantly increases the manufacturability of the nanowire crossbar systems in large scale manufacture. Manufacturing of these kinds of circuits would be easier compared to their clocked counterparts. Clocked synchronous architectures are difficult to map on crossbars architectures as they require complex placement and routing algorithms. In case of Asynchronous crossbar architecture, discrete blocks of crossbars can be used to map gates onto them and there is no need of a global synchronous signal to coordinate all the blocks. All clock related hardware components can be removed from the overall hardware design. Circuits would be less complex and easier to design.

2. Scalability

The overall circuit is self time i.e. timing information is integrated with data in the encoding. As the timing of each circuit is handled locally, Scalability of these circuits would be higher. The timing complexity remains the same even though the size of the circuit gets larger i.e. time taken for any particular computation will not change on the basis of the size of the circuit.

3. Robustness

Due to non-determinism of the directed self-assembly paradigm, nanowire crossbar circuits are anticipated to exhibit large variations in physical parameters. Since any physical variation in an electrical parameter may have its own negative effect on the timing behavior of the circuit, being able to design delay-insensitive circuits (i.e., correct operation of the circuit is independent of the timing) is a significant capability and it would greatly increase the robustness of the circuit to design parameter variations. As explained in Null Conventional logic for asynchronous logic subsection, there is no delay in processing data due to clock cycles as in clocked synchronous circuits, instead data would be processed as and when it is available.

4. Defect and Fault Tolerance

As NCL circuits have a definite flow patter i.e. DATA or NULL and vice versa the output can be checked if it is a data or null. In addition to the complete removal of all timing-related failure modes, testing complexity is reduced in that stuck-at-1 faults simply halt the circuit, since the NCL circuit cannot make a transition from DATA to NULL. Also, in case of dual-rail encoding, 11 is considered as an invalid code. So, any permanent or transient fault that results in this invalid codeword can be eventually detected. Only stuck-at-0 faults and some other transient faults need to be exercised with applied patterns. Design time and risk as well as circuit testing requirements are expected to be decreased because of the elimination of the complexity of the clock with its critical timing issues.

In this paper we are going to implement Null Conventional Logic on Nanowire crossbar architecture to realize "Asynchronous Crossbar Architecture". We also show the implementation of a full adder using the new crossbar architecture and discuss feasibility of a Multi-bit adder.

3. PROPOSED ARCHITECTURE

3.1 Programmable Gate Macro Block

The basic unit of the proposed architecture is a programmable gate macro block (PGMB). Each block is made of an AND plane and a OR plane formed by the diode

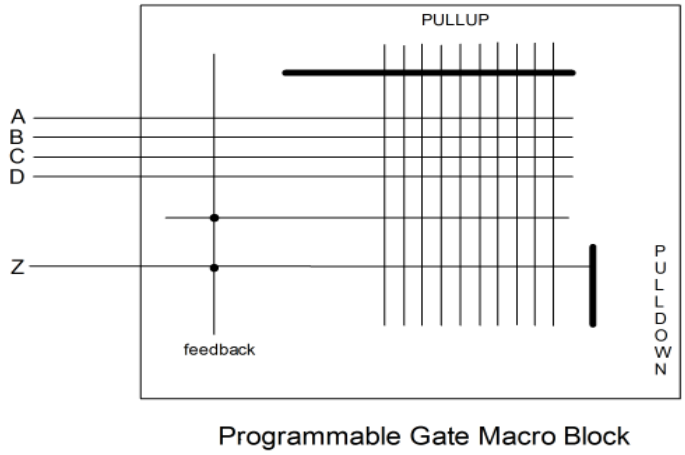


Figure 1 Basic Structure of PGMB

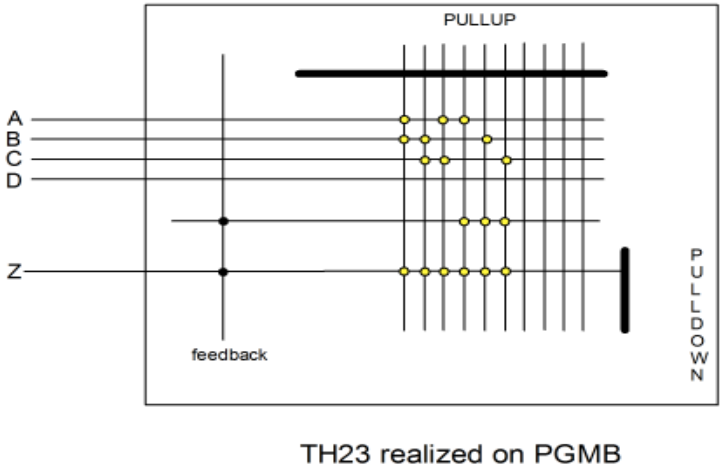


Figure 2 TH23 realized on a PGMB

crossbars. Vertical nano wires with pull up resistors form product terms and horizontal wire with pull down resistor add them using OR logic. It also has a feedback loop which drives the output back to an input wire. The maximum number of inputs to any threshold gate is 4 and along with this it needs a feedback to implement any of the 27 threshold gates [7]. Figure 1 shows the basic structure of a Programmable Gate Macro Block. It is a 6x10 crossbar structure which can take a maximum of 4 inputs as illustrated. Figure 2 shows the implementation of TH23 gate in the programmable gate macro block. The output of the TH23 gate is given by the logic Z = AB+BC+CA + (A+B+C)Z*. Z* is the previous output of the TH23 gate which is fed back to an input nanowire.

3.2 Physical Structure

The new architecture consists of array of PGMBs which are interconnected in the form of 2D grid structure. These blocks are surrounded by nano wires which are used to route the signals inside the grid structure. The PGMB's input and output nanowires cross these routing wires forming programmable cross points. By programming these cross points we can route the signals to any of the programmable gate macro blocks. The input stage consists of programmable resistor cross points formed by the micro wires and nano wires. By programming relevant cross points we can route the signals to the required PGMB. Each block can in turn be programmed to implement any of the threshold gates [7]. These blocks can tap the input signals by programming corresponding cross point formed by the nanowire column carrying input signal and nanowire row which is an input to the macro block. The output of the implemented threshold gate [7] can be routed to the other gates in the similar fashion. Thus the number of columns of nanowires between programmable macro blocks determines the amount of cross points available for routing signals. This number has to be sufficient to route all the required inputs and outputs to the macro blocks. The number of rows and columns of PGMBs in the grid are limited by the amount of signal degradation caused by the propagation. Before the complete degradation of the signal, a buffering stage can be implemented to restore the strength.

4. IMPLEMENTATION OF ONE BIT FULL ADDER

A full adder can be implemented using threshold gates as shown in Figure 3 [7]. Now, let us implement 1 bit full adder using the proposed architecture. The 1 bit full adder can be implemented using two th23 gates and two th34w2 gates as shown in the Figure 3. This requires 3 input bits a, b for addition and c for carry encoded in dual rail logic. These bits are represented by a0, a1, b0, b1, c0 and c1 respectively. By programming required cross points at the input cross bar these signals routed to the programmable gates.

The complete implementation of the 1 bit full adder is shown in the Figure 5.

The blocks present in row 1 and columns 1, 2 are programmed as th23 gates and blocks in row 2 and columns 1, 2 are programmed as th34w2 gates. The th23 gates require 3 inputs and therefore 1 input row is unused where as in th34w2 all the 4 input rows are used. The realized threshold gates on PGMB are shown in the Figures 1 and 4. Next we have to route required signal into the corresponding input rows. Outputs from the threshold gates should also be routed to the input of other gates or to the output block. This can be achieved by programming routing cross points and using free nano wires.

The NCL register stage consists of two TH22 and a TH12 gates that are used to generate a handshaking signal that helps in synchronizing the circuit. There are two kinds of signals, request for data and request for null, generated by the registers that are passed on to the previous register. The Ki (input from successive stage) and Ko (output to previous stage) are the handshaking signals and Do, D1 are input data rails and Q0, Q1 are the output rails. The single bit register stage is shown in the Figure 6.

5. CONCLUSION

In this paper, we have proposed a new clock-free nanowire crossbar architecture based on delay-insensitive logic known as Null Convention Logic. The complex clock distribution network can be removed from the hardware and many clock related failure modes can be intrinsically eliminated by the proposed clock-free architecture. To demon-

strate the feasibility, delay-insensitive full adder design has been implemented on the proposed clock-free architecture. Our future direction is to develop automated design optimization tools, testing schemes and defect-tolerant logic mapping techniques for the proposed architecture.

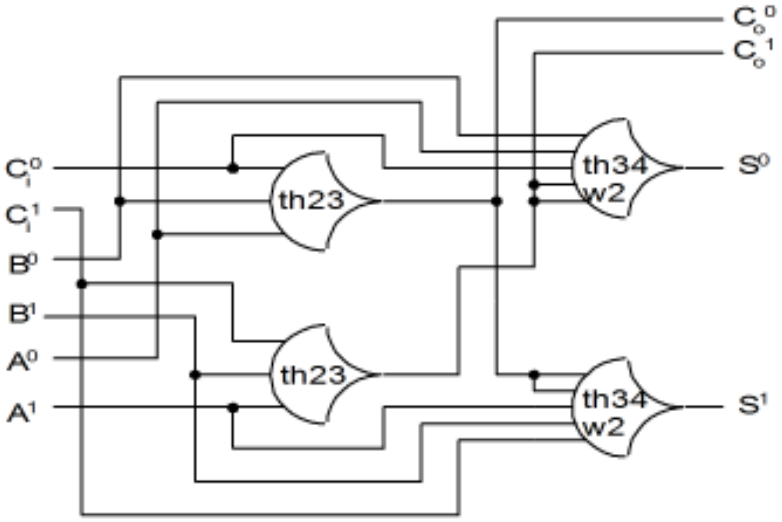
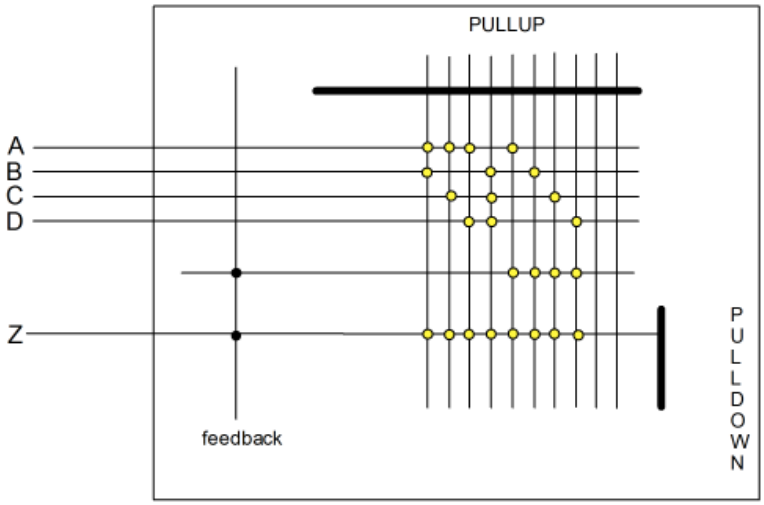


Figure 3 1-bit adder in NCL



TH34w2 realized on PGMB

Figure 4 TH34w2 realized on PGMB

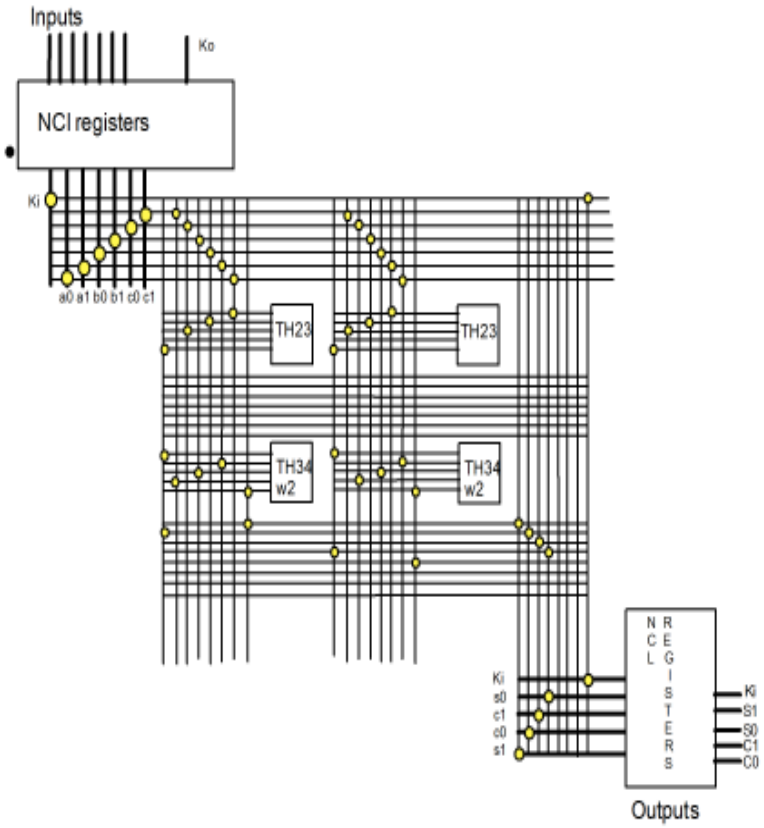


Figure 5 1-bit adder using proposed architecture

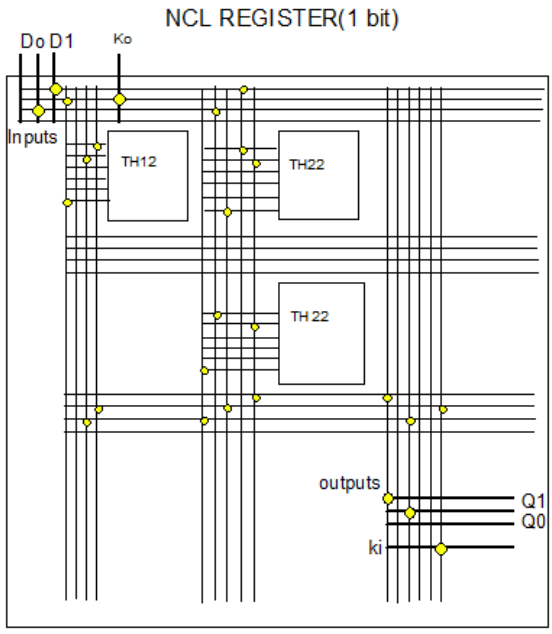


Figure 6 NCL one bit register on proposed architecture

6. REFERENCES

[1] J. Huang, M. B. Tahoori and F. Lombardi, "On the defect tolerance of nano-scale two-dimensional crossbars", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 96-104, Oct 2004.

[2] M. Jacome, C. He, G. de Veciana, and S. Bijansky, "Defect tolerant probabilistic design paradigm for nanotechnologies", IEEE/ACM Design Automation Conference (DAC), pp. 1-6, 2004.

[3] Jiangtao Hu et. al., "Chemistry and Physics in One Dimension : Synthesis and properties of Nanotubes and Nanowires," Acc. Chem. Res. , Vol. 32, pp. 435-445, 1999.

[4] Matthew M. Ziegler and Mircea R.Stan, "Design and analysis of Crossbar Circuits for Molecular Nanoelectronics," IEEE Nanotechnology Conference, pp. 323-327, 2002.

[5] Karl M. Fant, Scott A. Brandt, "NULL Convention Logic : A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis," IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 261-273, 1996.

[6] R. Smith and M. Ligthart, "High-Level Design for Asynchronous Logic," Design Automation Conference, pp. 431-436, 2001.

[7] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," Integration, The VLSI Journal, Vol. 37, No. 3, pp. 135-165, 2004.

[8] D. Whang, S. Jin and C. M. Lieber, "Large-Scale Hierarchical Organization of Nanowires for Functional Nanosystems," Japanese Journal of Applied Physics, Vol. 43, No. 7B, 2004.

[9] Y. Cui and C. M. Lieber, “Functional nanoscale electronic devices assembled using silicon nanowire building blocks,” Science, Vol. 291, pp. 851-853, 2001.

[10] Nicolas A. Melosh, Akram Boukai, Frederic Diana, Brian Geradot, Antonio Badolato, Pierre M. Petroff, James R. Health, “Ultrahigh-Density Nanowire Lattices and Circuits,” Science, Vol. 300, pp. 112-115, 2003.

[11] S. Smith, R. DeMara, J. Yuan, M. Hagedorn and D. Ferguson, “Delay- Insensitive gate-level pipelining,” Integration, the VLSI journal, Vol. 30,
pp. 103-131, 2000.

PAPER II

DEFECT-TOLERANT GATE MACRO MAPPING AND PLACEMENT IN CLOCK-FREE NANOWIRE CROSSBAR ARCHITECTURE

Ravi Bonam[1], Yong-Bin Kim[1] and Minsu Choi[1]

[1]Dept of ECE, University of Missouri-Rolla, Rolla, MO 65409-0040, USA

{rkbcdf, choim}@umr.edu

[2]Dept of ECE, Northeastern University, Boston, MA 02115, USA

ybk@ece.neu.edu

Abstract

Recently, we proposed a new clock-free nanowire crossbar architecture based on a delay insensitive paradigm called Null Convention Logic (NCL). The proposed architecture has simple periodic structure that is suitable for non-deterministic nanoscale assembly and does not require a clock distribution network - so it is intrinsically free from timing-related failure modes. Even though the proposed architecture offers improved manufacturability, it is still not free from defects. This paper elaborates on the different programming techniques to map a given threshold gate macro on a random PGMB (Programmable Gate Macro Block) with predefined dimension. Defect-Aware and Defect Unaware approaches have been considered to map a given threshold gate onto a PGMB without affecting its functionality. Defect aware approach uses a defect map, gate table which help in efficient programming and also conservative use of resources. Defect unaware approach on the other hand is faster than defect aware approach, does not use defect maps and is not as efficient as defect aware approach. Parametric simulation results using MATLAB are used to show the programmability of these approaches under various circumstances.

1. INTRODUCTION

Many of the nanoscale computing architectures proposed in recent years are clock driven. These architectures are mainly based on the two-dimensional nanowire crossbar architecture. In this architecture two sets of parallel, doped silicon nanowires or carbon nanotubes, are crossed over each other orthogonally to form a grid-like structure [1, 2]. The crossing over of these nanowires forms programmable junctions called crosspoints [1, 2, 3, 4]. The primary challenge in designing clocked architectures is to route the clock to all the components of the circuit. Due to imperfections in nanowires fabricated using current manufacturing processes, high defect densities are anticipated and realizing complex synchronous circuits on them is intricate. Hence, nanowire crossbars offer an opportunity and a challenge.

The opportunity is to achieve ultra-high density which has never been achieved by photolithography. The challenge is to make them *simple enough to be manufactured and reliable enough to be used in everyday computing applications*, since high-density systems consisting of nanometer-scale elements assembled in a bottom-up manner are likely to have many imperfections (much higher raw fabrication defect densities, as high as 10%, are expected ([5, 6]) and parametric variations. Asynchronous crossbar architecture efficiently helps utilize the opportunity and keep up to the challenge of fabricating reliable complex circuitry.

The proposed asynchronous nano-architecture is based on a delay-insensitive data encoding and self-timed logic - therefore, it is totally clock-free. Thus, no clock distribution network is needed and all failure modes related to the timing will be also eliminated. Potential benefits from the proposed asynchronous architecture include enhanced manufacturability, scalability, robustness and defect and fault-tolerance.

2. PROPOSED MAPPING AND PLACEMENT TECHNIQUES

2.1 Defect Unaware Approach

In this technique a predefined pattern of a gate is mapped on to a PGMB directly without the knowledge of any defects. We need not spend much time on generating the defect map of a given PGMB. The idea is to speed up the process of placement by compromising a few defective placements. The use of this technique is dependent on the consumer and manufacturer's capability to fabricate defect free PGMB's. The values of *m* and *n* for the predefined gate pattern are accessed from a gate table database which consists of all the required information (number of crosspoints, minimum rows and columns required to program the gate etc.) concerning a threshold gate. This table can be accessed by the algorithm while mapping and placement. The purpose of changing the dimensions of the predefined gate is to ensure precise placement of crosspoints. The algorithm is illustrated in Figure 1 using a flow chart.

The placed PGMB will be tested for functionality before it can be used in any of the circuits. Among the techniques mentioned in this paper the defect unaware approach is the fastest one because of the following reasons:

1. It does not scan the entire PGMB for generating a defect map which reduces the time it takes to program.
2. It does not use any kind of intelligence for tolerating defects which reduces the time it takes to program PGMB.
3. The space complexity and time complexity are minimal because it does not utilize memory for storing any kind of data related to programming the gates and the time complexity is restricted to time taken to test the functionality of the gate.

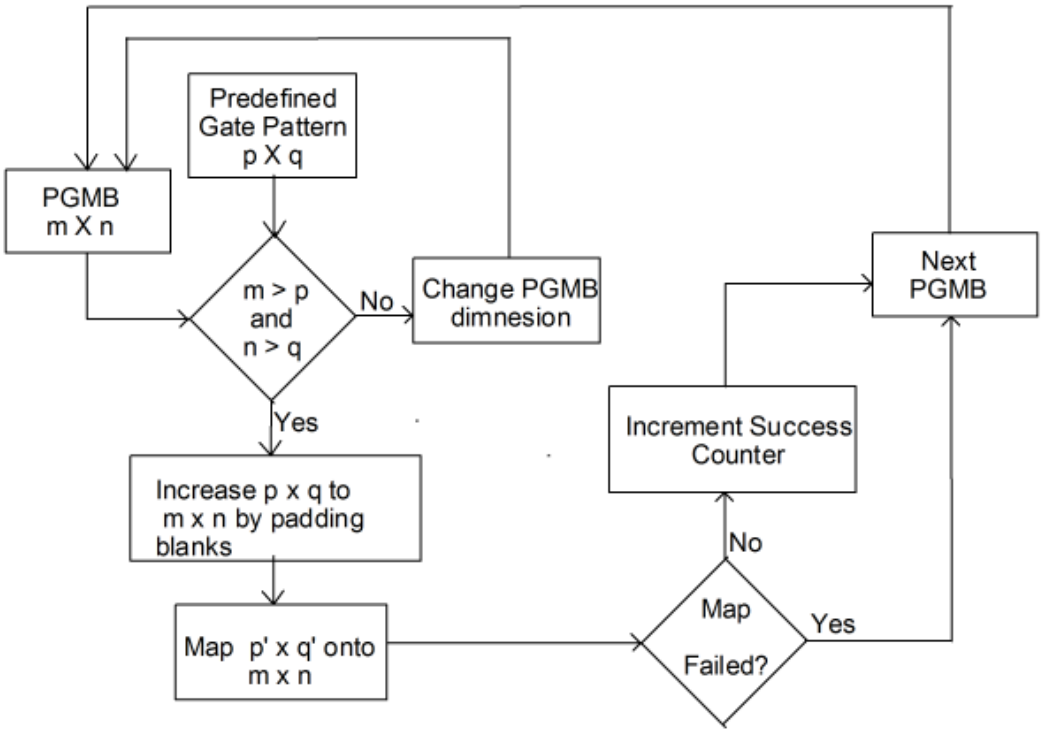


Figure 1 Defect Unaware Approach Control Flow

The drawback of this technique is that it does not yield significant programmability as defect rates of PGMB increases. Its programmability plummets as the defect rates increase because of a single representable pattern of each of the threshold gates.

2.2 Defect Aware Approach

As most of the current fabrication techniques cannot guarantee on their defect rates, the defect unaware process of placement is not as efficient in utilizing the inherent redundancy of a given PGMB. The defect aware approach on the other hand makes use of the available redundant crosspoints to program a given gate onto a PGMB. The main challenge associated with inherent redundancy is to use it without affecting the functionality of the gate. On observing the pattern of crosspoints of a gate we can infer that columns represent the intermediate product outputs and they can be interchanged in any fashion without affecting the functionality. A TH23 gate is represented in 6 ways with different

patterns of crosspoint placement without affecting the functionality of the gate in Figure 2. Figure 3 illustrates the Defect Aware Approach using a flow chart.

The defect aware technique utilizes the fact that any given gate can be represented in different ways without affecting the gate's functionality. This coupled with the inherent redundancy would give us a good scope of being able to map and place crosspoints on highly defective PGMB's.

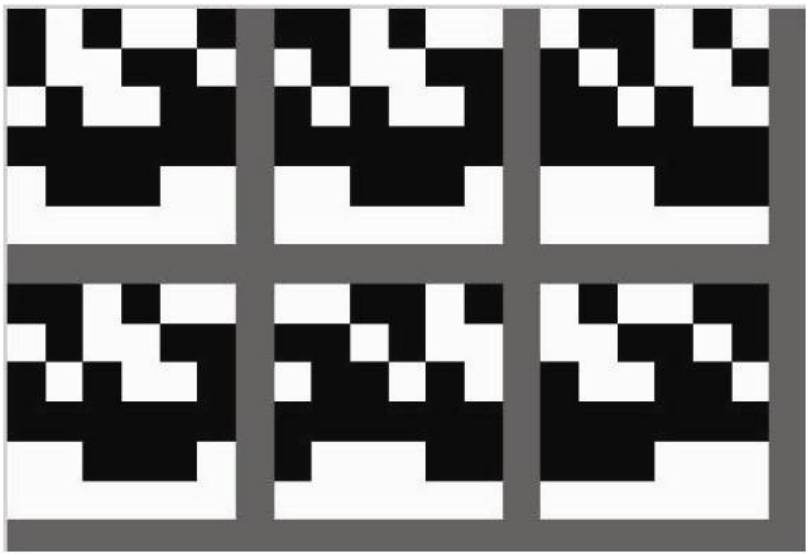


Figure 2 TH23 represented in 6 different ways

The Defect aware approach generates a defect map of a given PGMB and compares the pattern of defect free crosspoints with the required crosspoint pattern.
The algorithm for the defect aware approach is as follows:

1. Get the dimensions of PGMB (i.e., $m \times n$) and minimum required rows × columns for programming the required gate (i.e., $p \times q$).
2. If m > p and $n > q$ proceed to step 3, else roll back to step 1 and get next PGMB.
3. Get the count of defect-free crosspoints corresponding to each row on the PGMB, and also the required count of OR crosspoints for programming the Gate.
4. Consider a row from the PGMB, if available crosspoints in the row are greater than the required crosspoints then proceed to step 5. Else rollback get next row. If

there are no more rows with required crosspoints, then go to step 1 and get next PGMB.

4. Tabulate available crosspoints of each column corresponding to the selected row.
5. Get the programmability count of each of the required columns on each of the available columns for the selected row.
6. Starting with least programmability count, start placing the crosspoints.
7. If placed crosspoints is equal to the number of crosspoints to be placed then go to step 1 and start with next PGMB Else if columns are available go to step 7 and the next column. Else go to step 4 and select next row.

The significance of using this defect aware strategy is that it starts programming with the column (of the gate to be programmed) having least programming capability on a given PGMB which will help in maximum utilization of the inherent redundancy.

3. PARAMETRIC SIMULATION RESULTS

The simulation results for Defect Unaware Approach algorithm are shown in Figure 4. On analyzing the simulation results we can infer that as defect rate increases the programmability of the gate decreases. We can clearly observe that this method is suitable only for defect rates ranging from 0-10%. The simulations are performed by placing various gates onto a 6x10 PGMB.

The result of using the defect-aware approach for programming is illustrated in Figure 5. On analyzing the graph we can observe that almost all the gates are programmed even at 30% defect rate and then programmability (ratio of successfully programmed PGMB's to the total number of PGMB's) reduces which is inevitable. The simulation has been performed on a randomly generated defective 6x10 PGMB.

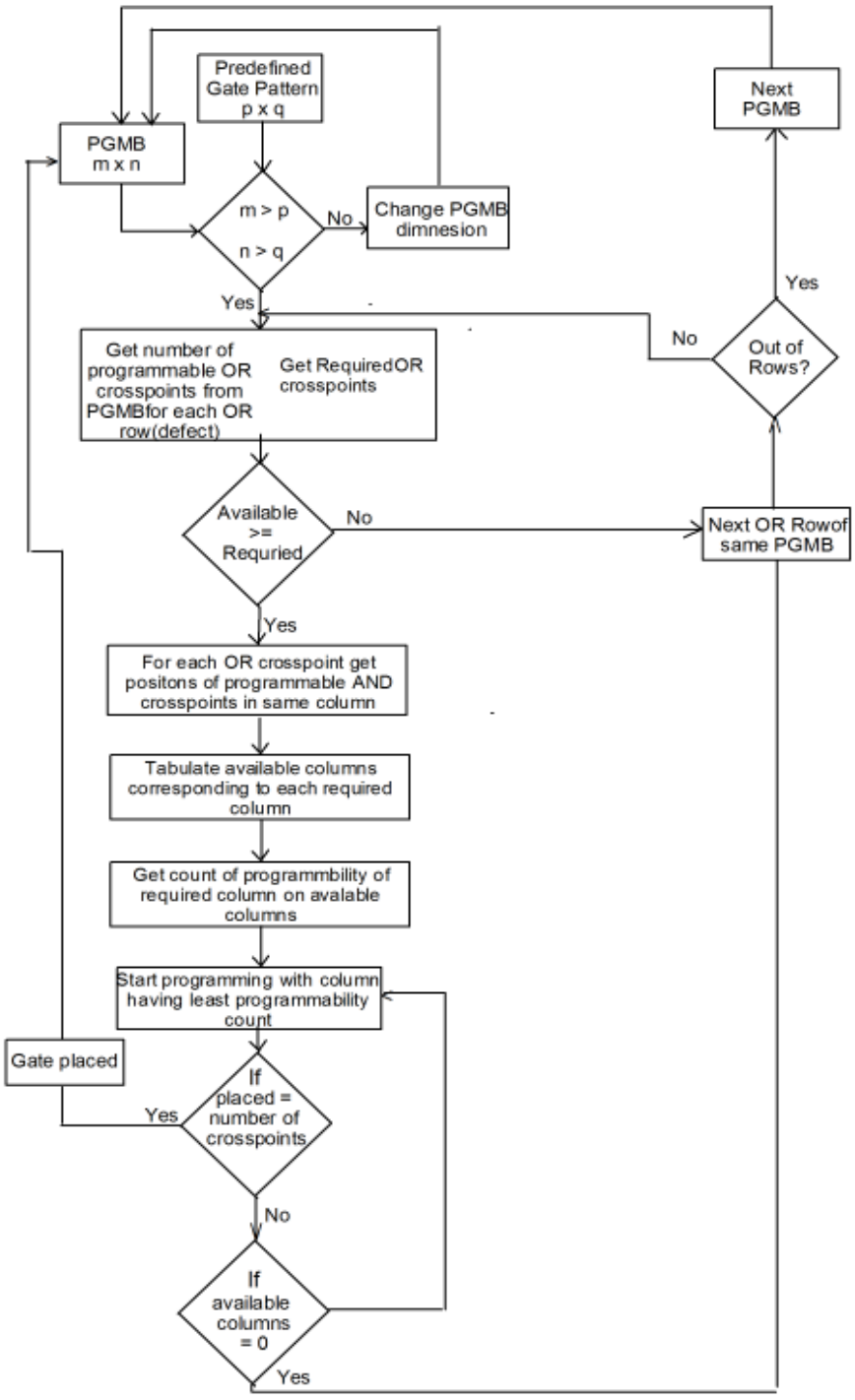


Figure 3 Defect Aware Approach control flow

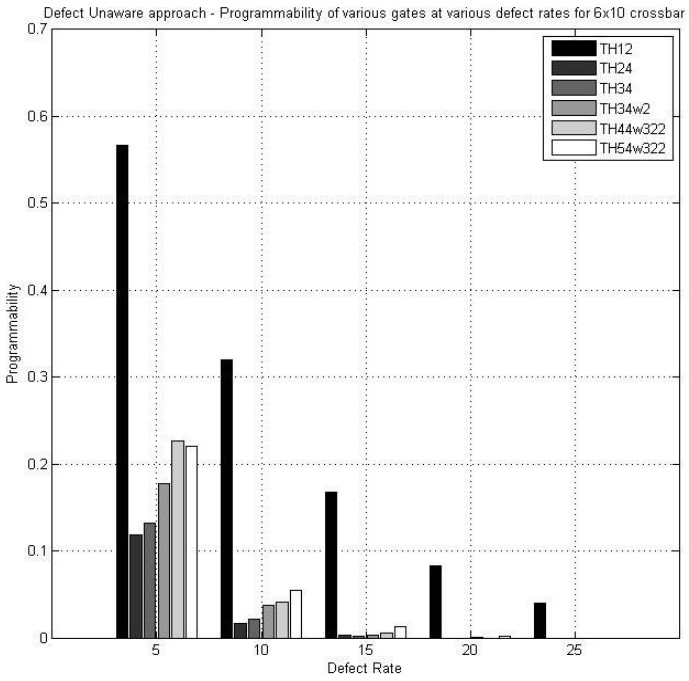


Figure 4 - Programmability of various gates at different defect rates - Defect Unaware Approach

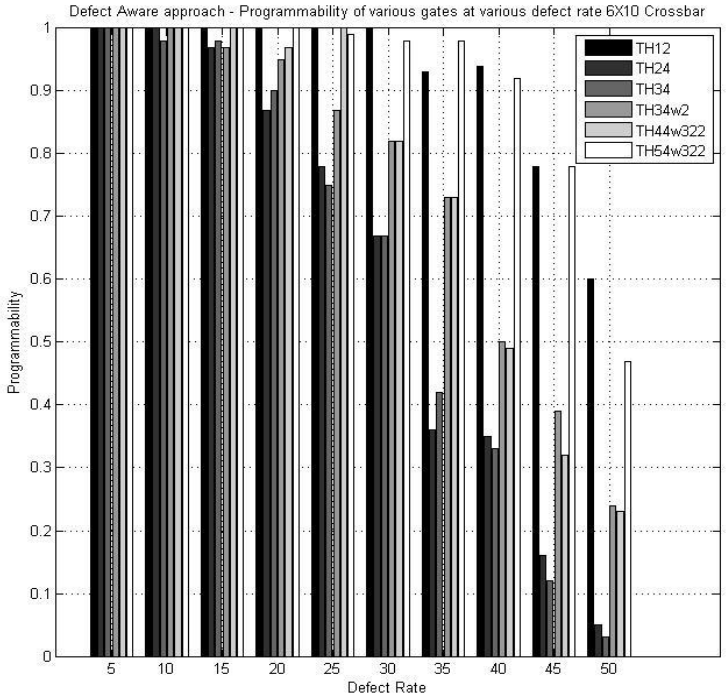


Figure 5 -Programmability various gates at different defect rates - Defect Aware Approach

Figures 6 and 7 illustrate the effect of increasing the size of the PGMB beyond 6x10. On analyzing the graphs we can infer that the programmability increases as inherent redundancy increases. There is significant increase in programmability for the defect aware approach due the fact that it is more efficient in utilizing inherent redundancy.

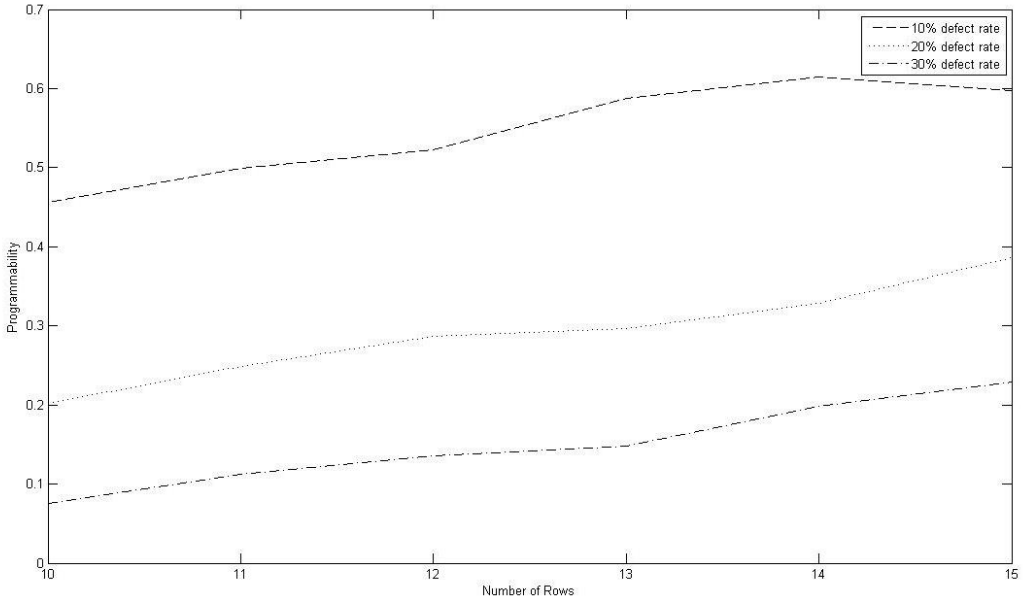


Figure 6 - Programmability at varying dimensions of PGMB for various defect rates - Defect Unaware Approach

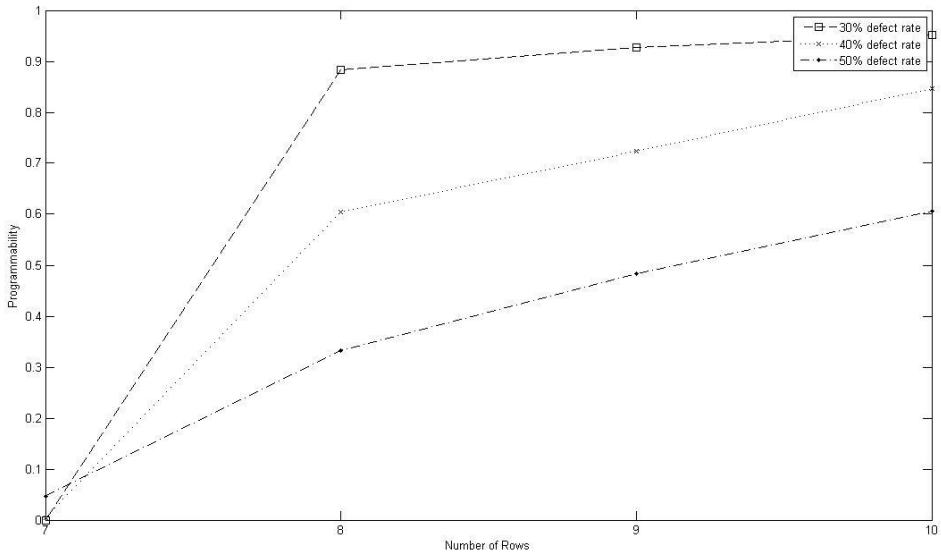


Figure 7 - Programmability at varying dimensions of PGMB for various defect rates - Defect Aware Approach

4. FUTURE WORK AND CONCLUSION

Even though the defect-aware approach is better than the defect-unaware approach especially when the defect rate is higher, it requires much more laborious testing (i.e., each PGMB and switch block should be tested to locate all defective crosspoints) and re-configuration (i.e., all defective crosspoints should be avoided when the netlist is actually placed and routed) tasks. However, the defect-unaware approach can be simpler since the netlist is directly mapped without considering any defects. After that, PGMBs and switch blocks can be functionally tested to locate ones with faults. These faulty ones then can be tested and reconfigured to avoid defects.

5. REFERENCES

[1] Matthew M. Ziegler and Mircea R.Stan, "Design and analysis of Crossbar Circuits for Molecular Nanoelectronics," IEEE Nanotechnology Conference, pp. 323-327, 2002.

[2] D.Whang, S. Jin and C. M. Lieber, "Large-Scale Hierarchical Organization of Nanowires for Functional Nanosystems," Japanese Journal of Applied Physics, Vol. 43, No. 7B, 2004.

[3] Y. Cui and C. M. Lieber, "unctional nanoscale electronic devices assembled using silicon nanowire building blocks," Science, Vol. 291, pp. 851-853, 2001.

[4] Nicolas A. Melosh, Akram Boukai, Frederic Diana, Brian Geradot, Antonio Badolato, Pierre M. Petroff, James R. Health, "Ultrahigh-Density Nanowire Lattices and Circuits," Science, Vol. 300, pp. 112-115, 2003.

[5] J. Huang, M. B. Tahoori and F. Lombardi, "On the defect tolerance of nano-scale two-dimensional crossbars", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 96-104, Oct 2004.

[6] M. Jacome, C. He, G. de Veciana, and S. Bijansky, "Defect tolerant probabilistic design paradigm for nanotechnologies", IEEE/ACM Design Automation Conference (DAC), pp. 1-6, 2004.

[7] Karl M. Fant, Scott A. Brandt, "NULL Convention Logic : A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis," IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 261-273, 1996.

[8] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," Integration, The VLSI Journal, Vol. 37, No. 3, pp. 135-165, 2004.

[9] S. Smith, R. DeMara, J. Yuan, M. Hagedorn and D. Ferguson, "Delay-Insensitive gate-level pipelining,"Integration, the VLSI journal, Vol. 30, pp. 103-131, 2000.

[10] R. Bonam, S. Chaudhary, Y. Yellambalase and M. Choi, "Clock-Free Nanowire Crossbar Architecture based on Null Convention Logic (NCL)", Accepted to appear in the 7th IEEE International Conference on Nanotechnology (IEEE-Nano), Apr 2007.

[11] R. Bonam, Y. Yellambalase and M. Choi, "Redundancy Optimization for Clock-Free Nanowire Crossbar Architecture", Accepted to appear in the 7th IEEE International Conference on Nanotechnology (IEEE-Nano), Apr 2007.

[12] S. C. Smith, "Gate and Throughput Optimizations for NULL Convention Self-Timed Digital Circuits", Ph.D. Dissertation, School of Electrical Engineering and Computer Science, University of Central Florida, May 2001.

[13] M. Mishra and S. Goldstein, Scalable defect tolerance for molecular electronics, Workshop Non-Silicon Computation (NSC-1), pp. 78, 2002.

[14] M. Tehranipoor, "effect tolerance for molecular electronics-based nanofabrics using built-in self-test procedure", 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005, Oct. 2005 pp 305 - 313

Paper III

EVALUATION OF DEFECT-TOLERANT MAPPING AND PLACEMENT TECHNIQUES FOR ASYNCHRONOUS NANOWIRE CROSSBAR ARCHITECTURE

Ravi Bonam, Minsu Choi

Dept of ECE, Missouri University of Science & Technology

(Formerly, University of Missouri-Rolla)

Rolla, MO 65409-0040, USA

{rkbcdf, choim}@mst.edu

Abstract

To deal with photolithographic barrier of CMOS, numerous nanoscale devices have been proposed. Nanowire crossbar is one of the emerging nanotechnologies with considerable merits. There have been numerous nanowire crossbar architectures proposed till date, although all of them are envisioned to be synchronous. The clock is an important part in a circuit and it needs to be connected to all the components to synchronize their operation. Considering non-deterministic nature of the nano-integration, realizing a clock distribution network on nanowire crossbar system would be difficult. Also, numerous timing-related failure modes should be addressed to assure the correctness of operations. To deal with such clock-related issues, a new clock-free crossbar architecture has been recently proposed to resolve the issues with clocked counterparts. The proposed architecture is built around uniformly-sized programmable crossbar block, that can be configured to any given threshold gate function, called Programmable Gate Macro Block (PGMB). Although the proposed clock-free architecture has simpler periodic structure that enhances manufacturability and provides robust operations by eliminating timing-induced failures, it is still not free from manufacturing defects caused by nondeterministic nature of nanoscale assembly. To address this issue, we have proposed numerous defect-tolerant gate mapping and reconfiguration algorithms, each with a unique performance profile. These

algorithms are evaluated on the basis of parametric simulations consisting of expected defect rates and the inherent redundancy. The observations made from these evaluations will help select the best one basing on a variety of fabrication parameters.

******* This manuscript is an extension to paper II.

1. MAPPING AND PLACEMENT TECHNIQUES

1.1 Defect Unaware Approach – Shift Algorithm

This approach is an extension of the Defect-Unaware approach and employs a circular shift procedure which shifts the columns in both the AND and OR planes collectively. Shifting the planes creates greater number of representable patterns of the gates which ensure successful mapping while maintaining proper functionality. This approach creates a better trade-off between the time required to program and programmability. This approach would yield better programmability when compared to the Defect-Aware Approach described in the previous section. The control flow for placement of crosspoints using the defect-unaware strategy is as illustrated in Figure 1.

Detailed steps of the defect-aware-shift algorithm are as follows:

1. Obtain the dimension of PGMB (i.e., p x q) and minimum required rows x columns for programming the required gate macro (i.e., m x n).
2. If $p > m$ and $q > n$ proceed to step 3, else go back to step 1 and proceed to the next PGMB.
2. Map the threshold gate onto the PGMB and test for functionality.
3. If functionality of the PGMB matches with that of the threshold gate being programmed proceed to step 7, else proceed to step 5.
4. If shift counter is less than the threshold shift value then proceed to step 6, else proceed to step 8. The threshold value for shift counter is equal to the number of columns in the predefined gate pattern.

5. Shift columns in the threshold gate and generate a new pattern from the predefined gate pattern and go to step 3.
6. Increment the successfully programmed counter.
7. Go on to next PGMB and reinitialize the shift counter.

This technique is a compromise between the pure defect-unaware and defect-aware approaches, but is partially a defect-unaware technique based on the fact that it does not generate a defect map of the PGMB - therefore, defect locations are unknown throughout the mapping procedure. As described in the flow chart in figure 1, it shifts the columns creating better logical representations of the threshold gate thereby increasing the probability of programming the PGMB successfully without affecting the functionality of the threshold gate. This technique compensates the limited representations of the threshold gate in aforementioned approach at the cost of tenuous increase in time complexity while having no effect on space complexity.

1.2 Defect-Unaware Approach – Modified Shift Algorithm

This approach is an annexure to the Shift Algorithm and applies the shift algorithm's property on the AND plane's rows and columns, which is a part of PGMB, in addition to the shifting the columns of the OR of the particular gate. This technique creates greater number of gate patterns than the previous methods which increases the probability of successfully programming a defective PGMB. This approach yields better programmability of the PGMB's when compared to the previous methods at noticeable defect rates. The control flow for placement of crosspoints using the defect-unaware strategy is as illustrated in Figure 2.

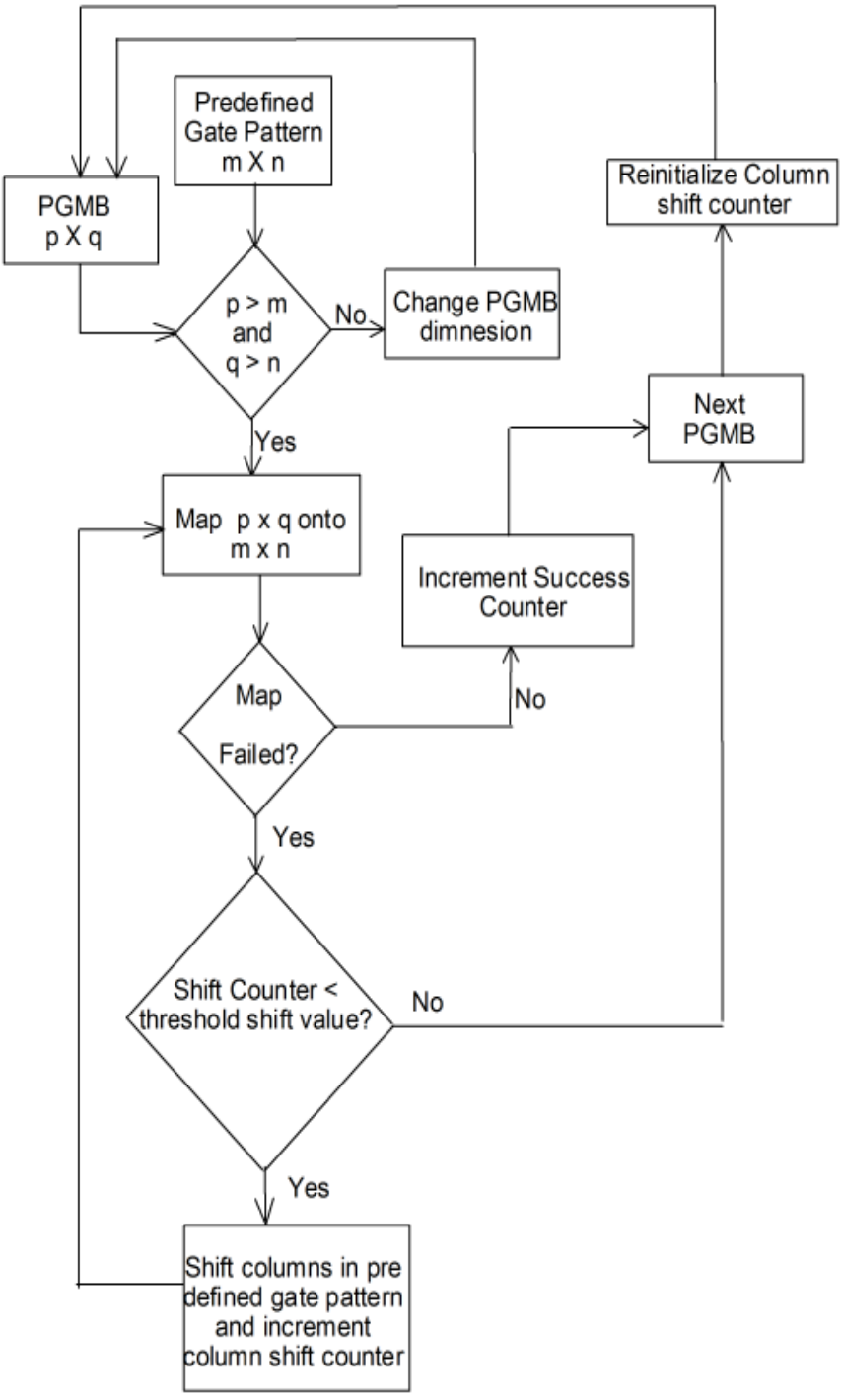


Figure 1- Defect Unaware with Circular Shift control flow

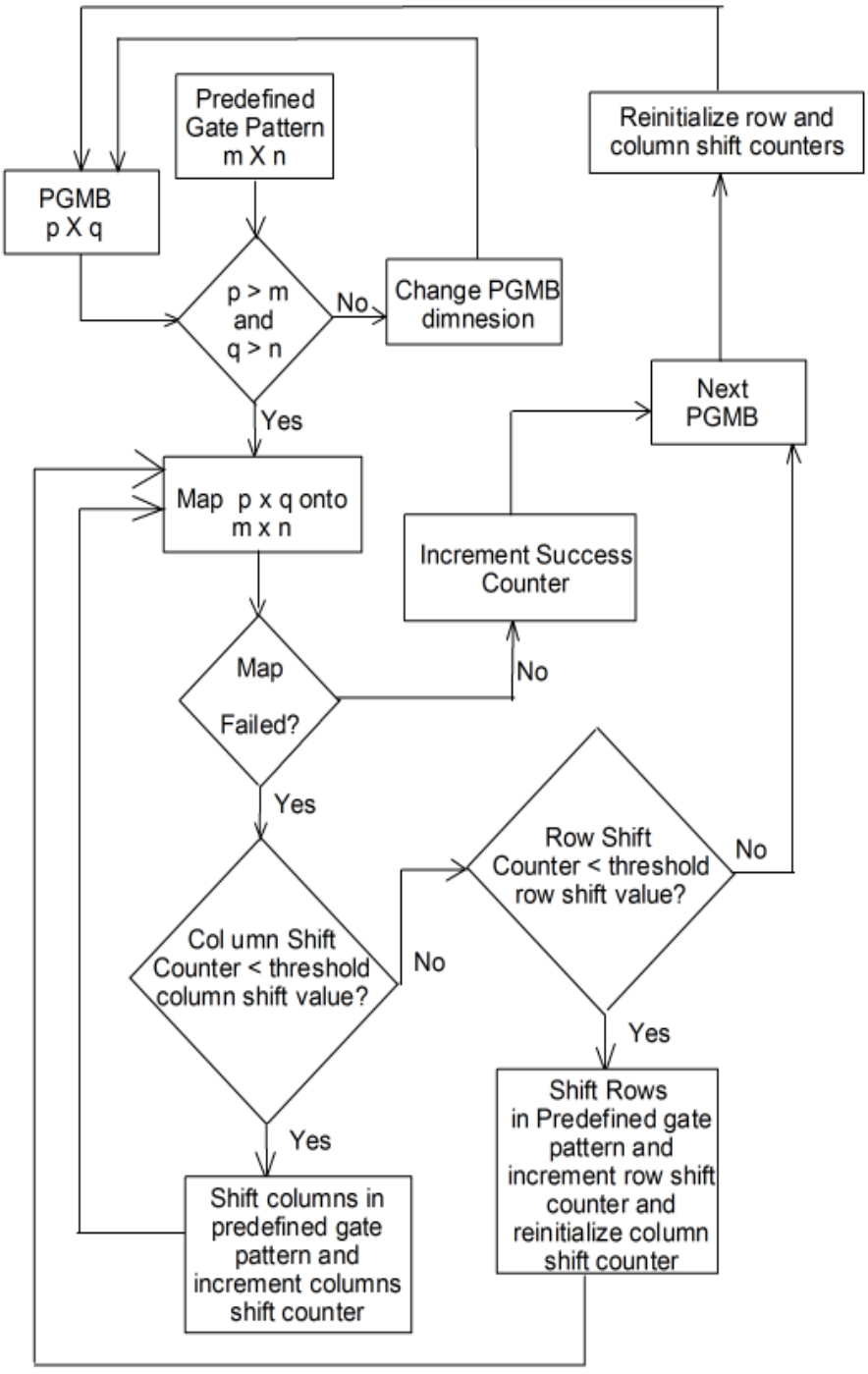


Figure 2- Defect Unaware with Circular Shift of AND and OR planes control flow

This technique breaks up the construct of the PGMB into its basic parts, the AND plane and the OR plane, and applies the shift process to the rows and columns of the AND plane and shifts only columns in the OR plane. It is partially a Defect Unaware technique based on the fact that it does not generate a defect map of the PGMB, and utilizes the fact that the expression of the gate is in the form of sum-of-product (SOP). As described in the flow chart in figure 2, it shifts the columns and rows in the AND plane and creates greater number of logical representations of the threshold gate thereby increasing the probability of programming the PGMB successfully. This technique increases the programmability considerably with a slight increase in time complexity while not effecting space complexity. These shift Algorithm approaches would use the same gate macro library as the Defect-Unaware Approach.

2. PARAMETRIC SIMULATION RESULTS

Aforementioned gate macro mapping algorithms have been tested with various parameter sets to obtain parametric simulation data. Each of the results describes the variation of programmability of various TH gate macros. Six representative TH gates with various complexities, TH12 ($Z = A+B$), TH24 ($Z = AB+BC+CD+AD+BD+CD$), TH34 ($Z = ABC+ABD+BCD+ACD$), TH34w2($Z = AB+BC+AD+BCD$), TH44w322 ($Z =AD+BC$), TH54w322($Z = AB+AC+BCD$) have been used in the simulations for testing their programmability at different defect rates on PGMB's with various dimensions.

In this section programmability results for all the four mapping algorithms are provided (including the algorithms from paper II).

2.1 Defect-Unaware Approach

Figures 3 and 4 illustrate the variation of programmability of the aforementioned six threshold gates with PGMB's of various dimensions at 5% and 10% defect rates respectively. On analyzing the results from Figures 3, 4, we can observe the rapid decrease of programmability when the defect rate increases from 5% to 10%. Careful observation of each of the given figures shows us that the gates with lesser number of crosspoints,

which depend on the number of AND terms in their expression, have higher programmability as the redundancy increases (e.g., a TH12 gate has high programmability compared to the TH24).

This method assures a profitable yield while manufacturing simple circuits at low PGMB defect rates. It will ensure lesser cost and faster mapping process rather than using the other complex techniques. We can also use this technique if manufacturers are able to manufacture PGMBs with lesser defect rate and also can tolerate the cost of higher redundancy overhead.

2.2 Defect-Unaware Approach – Shift Algorithm

Figures 5, 6 and 7 show the variation of programmability with change in PGMB dimensions at 5, 10 and 15% defect rates, respectively. Comparison of figures 5, 6, 7 infers that this process is better than the initially shown defect unaware approach based on the programmability at similar defect densities.

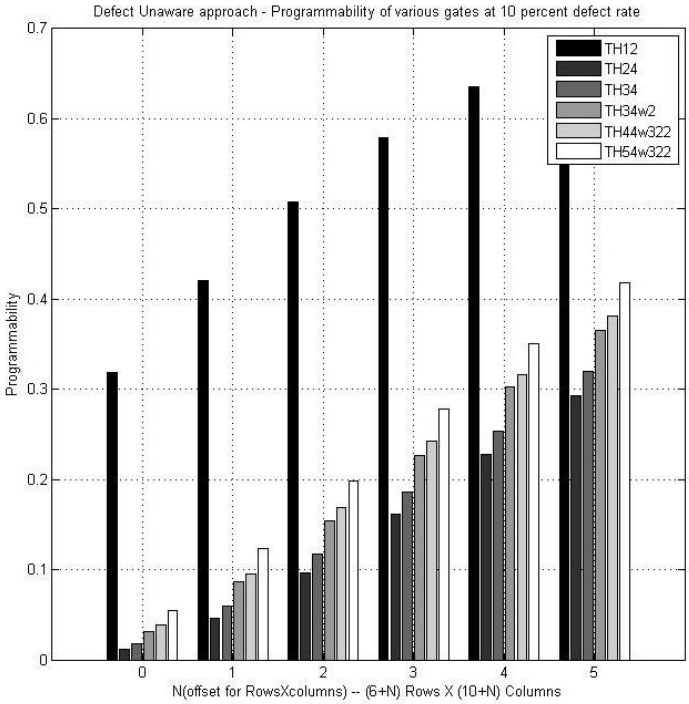


Figure 3 -Variation of Programmability with change in PGMB dimensions for various gates at 5 percent – Defect Unaware Approach

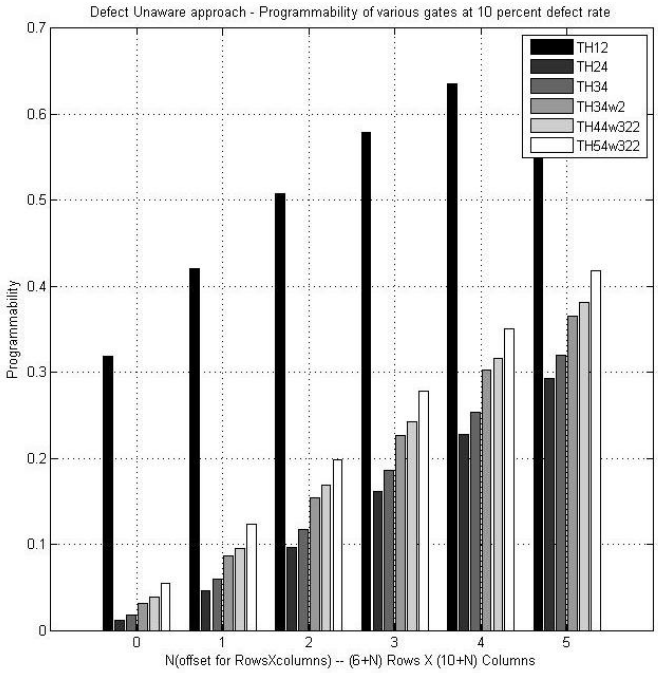
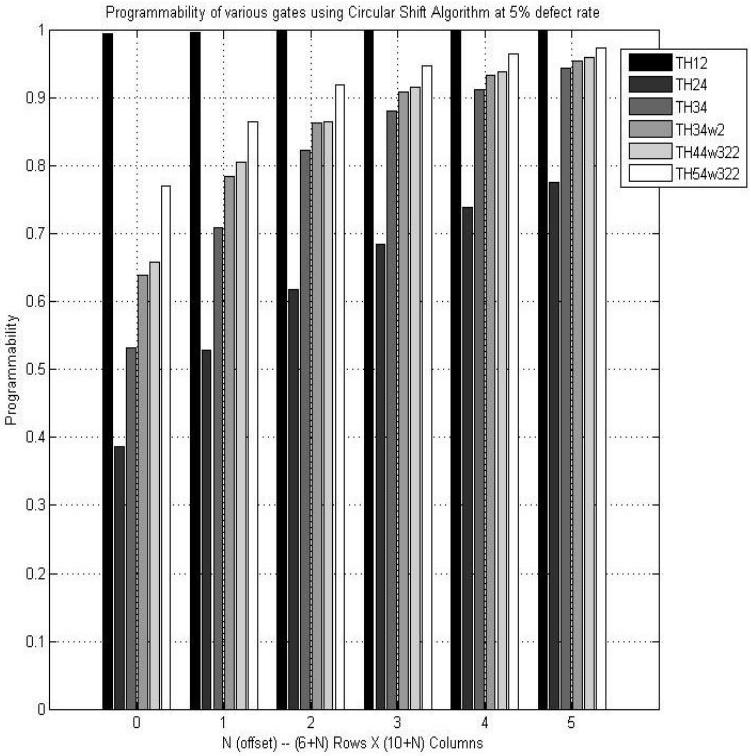


Figure 4 -Variation of Programmability with change in PGMB dimensions for various gates at 10 percent - Defect Unaware Approach



\

Figure 5- Variation of Programmability with change in PGMB dimensions for various gates at 5 percent defect rate – Defect Unaware – Shift Approach

This can be used as an immediate upgrade if the manufacturer is not willing to tolerate high redundancy at similar defect rates. Table 1 shows the average column shifts pertaining to a successfully programmed PGMB. Average Shift rates of six different gates at three different defect rates have been presented. These average numbers of shifts directly affect the time complexity of the algorithm since the overall time to program the given gate macro directly depends on the number of shifts in this algorithm. The average number of shifts per successfully-mapped gate is also directly proportional to the amount of inherent redundancy. Hence based on these results, this method can be considered as a compromise between the defect-aware and defect-unaware approaches which are at the two opposite extremes.

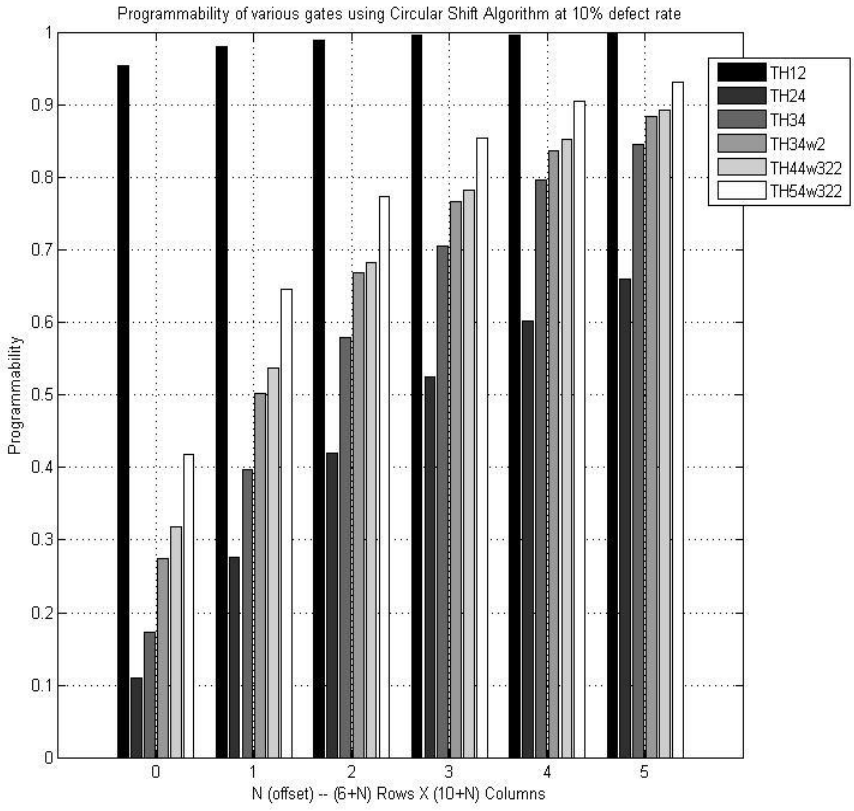


Figure 6 - Variation of Programmability with change in PGMB dimensions for various gates at 10 percent defect rate – Defect Unaware – Shift Approach

2.3 Defect-Unaware Approach - Modified Shift Approach

Figures 8, 9 and 10 illustrate the variation of programmability with change in PGMB dimensions at 10%, 15% and 20% defect rates, respectively. Careful analysis of figures 8, 9, 10 indicates an improvement in programmability when compared to the Defect Unaware and the Shift approach.

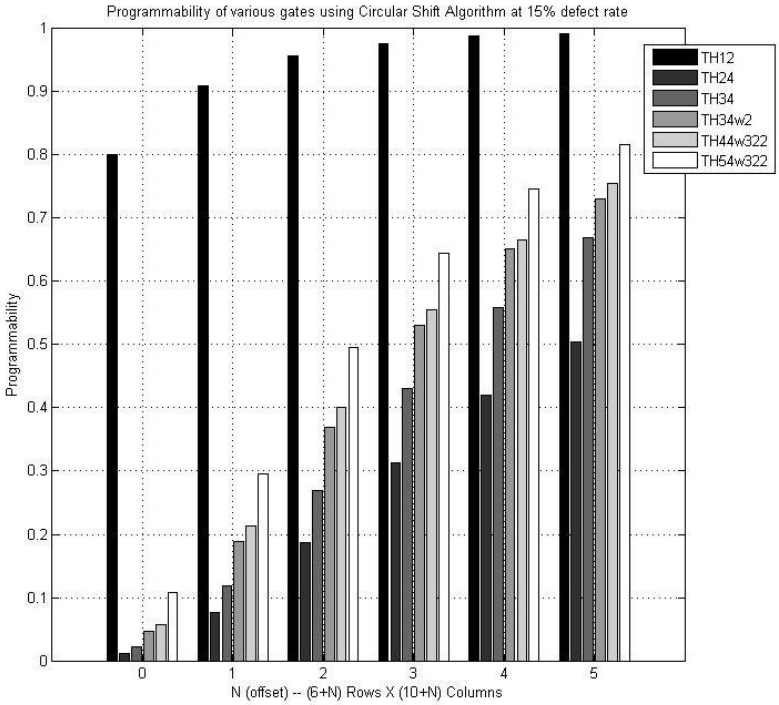


Figure 7 - Variation of Programmability with change in PGMB dimensions for various gates at 15 percent defect rate – Defect Unaware – Shift Approach

The modified shift algorithm is considered to be an annexure for the shift approach which has a considerable increase in programmability at similar defect rates. The probable difference between the two approaches would be increase in time complexity which accounts for increased programmability in the latter approach.

Average shift rates of the AND plane's rows and columns for a successfully programmed gate are presented in tables II and III, respectively. The row and column shifts

directly affect the time complexity of the algorithm similar to the shift approach. Shifts in rows and columns increase the time taken to program a gate, consequently increasing the programmability. Manufacturers can prefer this method while programming moderately complex circuits, which would yield considerably higher programmability compared to the Shift approach.

2.4 Defect-Aware Approach

This subsection illustrates the variation of programmability for the defect aware approach at different defect rates. Figures 11, 12 and 13 show variations in programmability with change in PGMB dimension at 15 %, 20% and 30% defect rates, respectively.

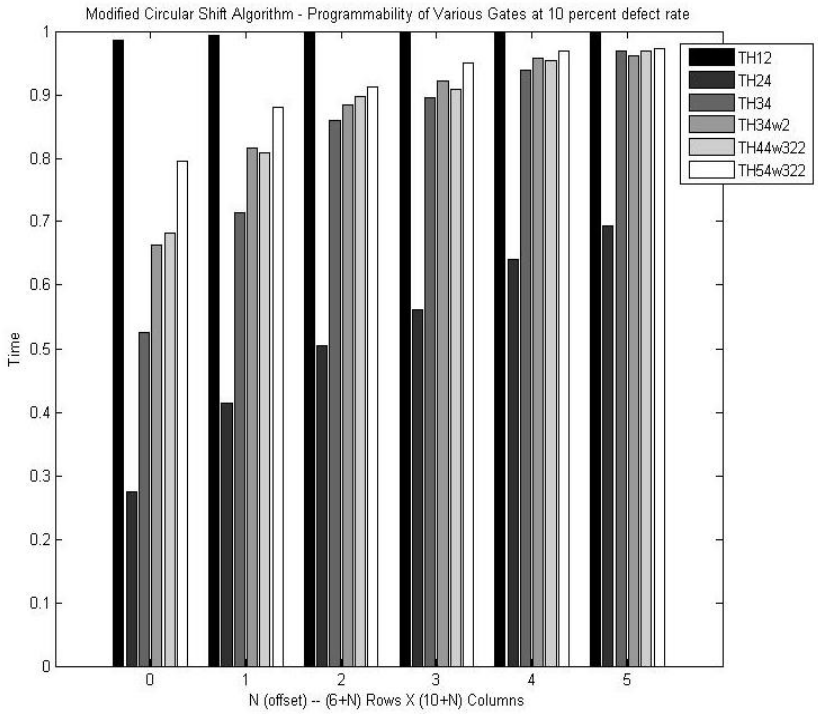


Figure 8 - Variation of Programmability with change in PGMB dimensions for various gates at 10 percent defect rate – Defect Unaware – Modified Shift Approach

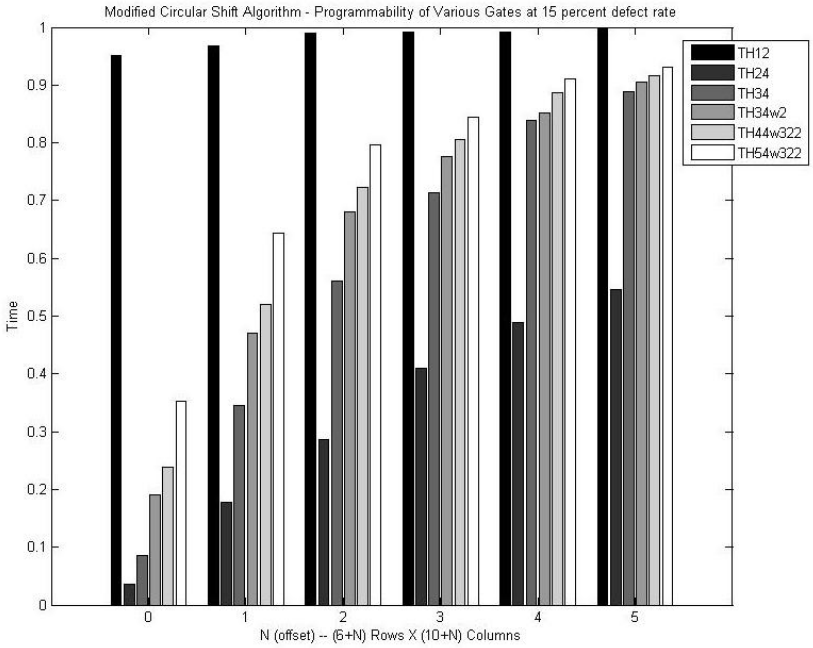


Figure 9 - Variation of Programmability with change in PGMB dimensions for various gates at 15 percent defect rate – Defect Unaware – Modified Shift Approach

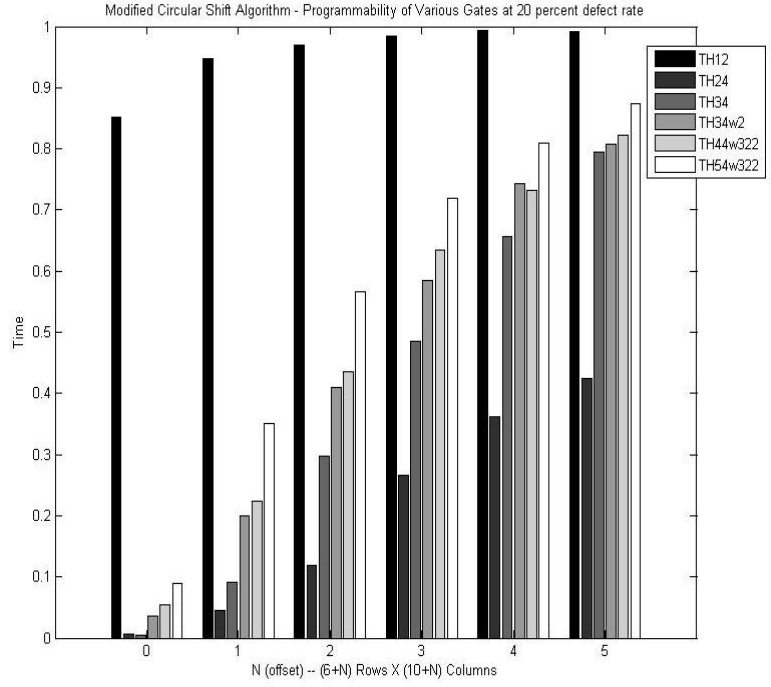


Figure 10 - Variation of Programmability with change in PGMB dimensions for various gates at 20 percent defect rate – Defect Unaware – Modified Shift Approach

Parametric data shown in Figures 11, 12, 13 infer that the defect aware approach is far more superior to all the mentioned defect-Unaware techniques considering the fact that it has steady levels of significant programmability at high defect rates. This is due to the fact that it generates a defect map of the PGMB on which the gate has to be programmed. This algorithm can be used to program all kinds of realizable circuits on highly defective PGMBs with profitable programmability.

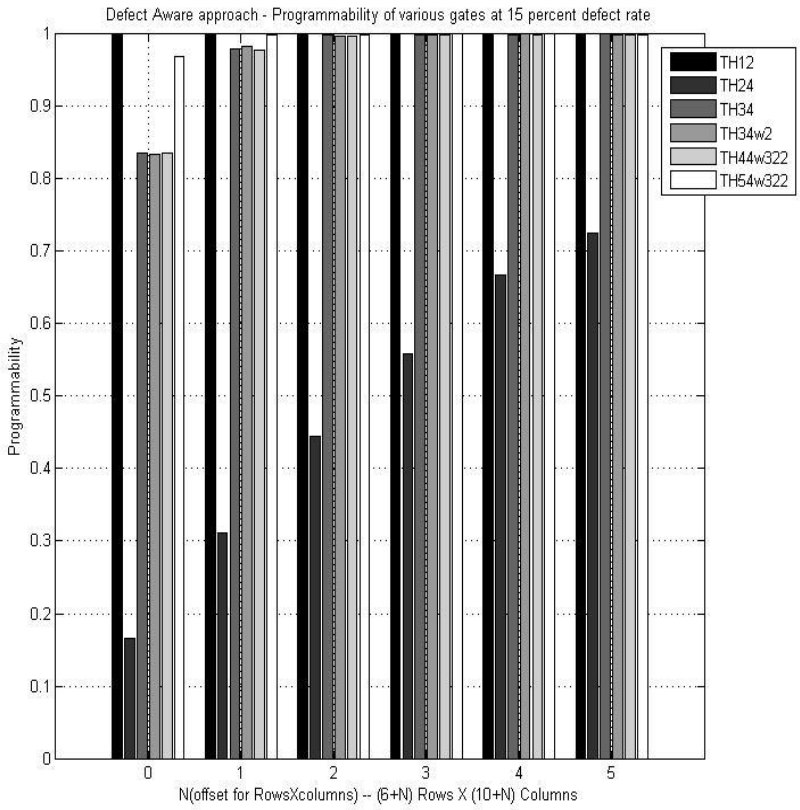


Figure 11 - Variation of Programmability with change in PGMB dimensions for various gates at 15 percent defect rate – Defect Aware Approach

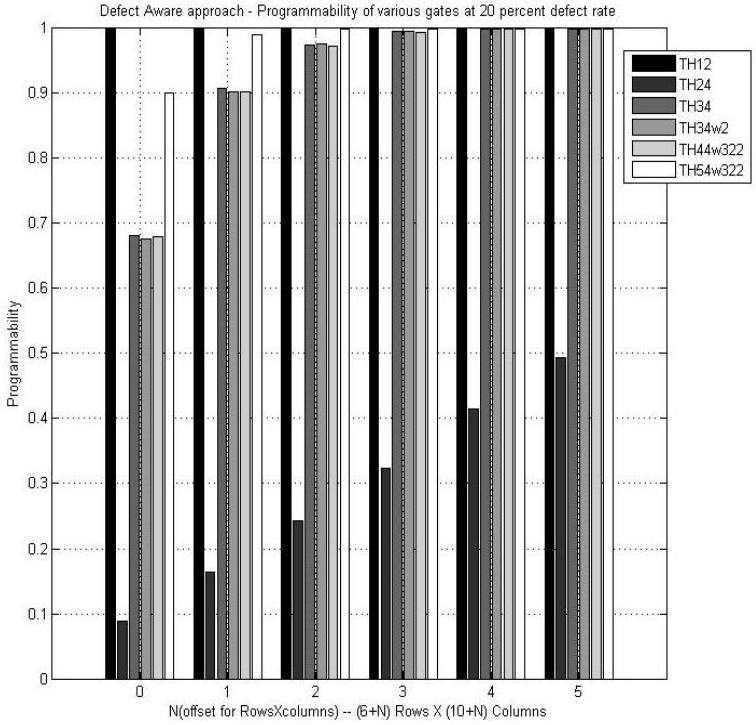


Figure 12 - Variation of Programmability with change in PGMB dimensions for various gates at 20 percent defect rate

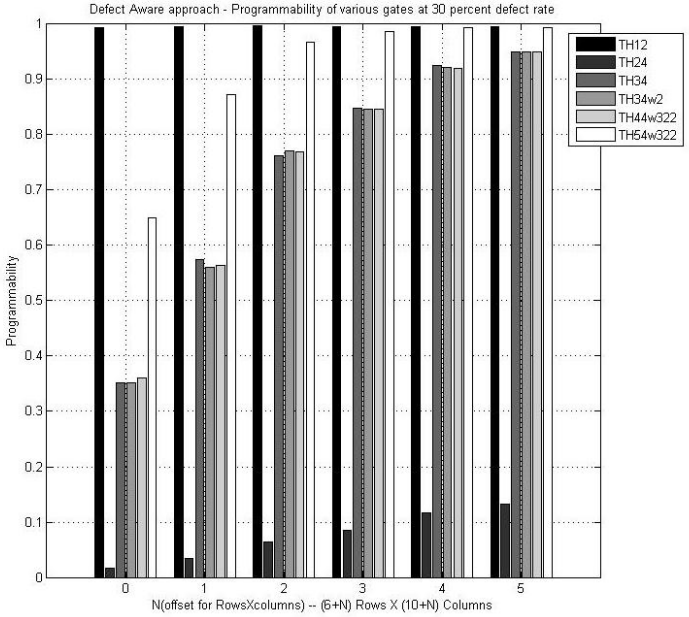


Figure 13 - Variation of Programmability with change in PGMB dimensions for various gates at 30 percent defect rate

3. CONCLUSION AND FUTURE WORK

A new asynchronous nanowire crossbar architecture has been recently proposed by authors' research group. Even though the proposed clock-free architecture has various advantages over its clocked counterpart - such as improved manufacturability, robustness, scalability and defect-tolerance, it is still not free from issues caused by higher defect rates due to nondeterministic nanoscale assembly. In order to address these issues, a few defect-tolerant mapping techniques for TH gate macros are proposed and numerically evaluated in this work.

Defect-aware and defect-unaware approaches are considered to be the extreme cases for mapping and placement of threshold gate macros onto PGMB. The Shift approach and Modified Shift approach serve as compromised approaches, whose attributes seem to be in between the extremities. The manufacturer can decide whichever technique suits best based on various parameters such as programmability, defect rate and complexity of the circuit. Even though the defect-aware approach is better than the defect-unaware approach especially when the defect rate is higher, it requires much more laborious testing (i.e., each PGMB and switch block should be tested to locate all defective crosspoints) and reconfiguration (i.e., all defective crosspoints should be avoided when the netlist is actually placed and routed) tasks. However, the two variants of shift algorithms described in this paper will serve as compromised approaches.

A combination of these techniques based on the complexity of the circuit being programmed will prove advantageous and provide an excellent balance between time complexity and space complexity, having maximum programmability. Efficient differentiation of the complexity of the circuit will prove advantageous to the manufacturer to decide the best technique suited for mapping and placement of the crosspoints. Each of the techniques illustrated in this paper have their own advantages and disadvantages and can be further analyzed on the basis of time complexity and space Complexity. These issues will give a comprehensive view of the techniques. We will be addressing these issues in future work.

4. REFERENCES

[1] Yuan Taur, Senior Member, IEEE, Douglas A. Buchanan, Member, Ieee, Wei Chen, Member, IEEE, David J. Frank, Member, IEEE, Khalid E. Ismail, Shih-Hsien Lo, George A.Sai-Halasz, Fellow, IEEE, Raman G. Viswanathan, Hsing-Jen C. Wann, Shalom J. Wind, Member, IEEE, And Hon-Sum Wong "CMOS Scaling into the Nanometer Regime", Proceedings Of The IEEE, Vol. 85, No. 4, April 1997

[2] Scott E. Thompson, Senior Member, IEEE, Robert S. Chau, Senior Member, IEEE, Tahir Ghani, Kaizad Mistry, Sunit Tyagi, Member, IEEE, and Mark T. Bohr, Fellow, IEEE, "In Search of Forever, Continued Transistor Scaling One New Material at a Time", IEEE Transactions On Semiconductor Manufacturing, Vol. 18, No. 1, February 2005

[3] Shekhar Borkar "Design challenges of technology scaling", IEEE Micro, Volume 19, Issue 4, Jul-Aug 1999

[4] Mohab Anis, Student Member, IEEE, Mohamed Allam, Member, IEEE, and Mohamed Elmasry, Fellow, IEEE "Impact of Technology Scaling on CMOS Logic Styles", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMSII: ANALOG AND DIGITAL SIGNAL PROCESSING, VOL. 49, NO. 8, AUGUST 2002

[5] Mark T. Bohr, Senior Member, IEEE "Nanotechnology Goals and Challenges for Electronic Applications", IEEE TRANSACTIONS ON NANOTECHNOLOGY, VOL. 1, NO. 1, MARCH 2002

[6] Forshaw, M., Stadler, R., Crawley, D., Nikolic, K. "A short review of nanoelectronic architectures " Nanotechnology, Volume 15, Issue 4, April 2004

[7] G. Snider, P. Kuekes, T. Hogg, R. Stanley Williams "Nanoelectronic architectures", Applied Physics A: Materials Science and Processing,Volume 80, Number 6, March 2005

[8] Andr Dehon "Nanowire-based programmable architectures" ACM Journal on Emerging Technologies in Computing Systems (JETC), Volume 1 , Issue 2 , July 2005

[9] Matthew M. Ziegler and Mircea R.Stan, "Design and analysis of Crossbar Circuits for Molecular Nanoelectronics," IEEE Nanotechnology Conference, pp. 323-327, 2002.

[10] D. Whang, S. Jin and C. M. Lieber, "Large-Scale Hierarchical Organization of Nanowires for Functional Nanosystems," Japanese Journal of Applied Physics, Vol. 43, No. 7B, 2004.

[11] Y. Cui and C. M. Lieber, "Functional nanoscale electronic devices assembled using silicon nanowire building blocks," Science, Vol. 291, pp. 851-853, 2001.

[12] Nicolas A. Melosh, Akram Boukai, Frederic Diana, Brian Geradot, Antonio Badolato, Pierre M. Petro®, James R. Health, "Ultrahigh-Density Nanowire Lattices and Circuits," Science, Vol. 300, pp. 112-115, 2003.

[13] J. Huang, M. B. Tahoori and F. Lombardi, "On the defect tolerance of nano-scale two-dimensional crossbars", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 96-104, Oct 2004.

[14] M. Jacome, C. He, G. de Veciana, and S. Bijansky, "Defect tolerant probabilistic design paradigm for nanotechnologies", IEEE/ACM Design Automation Conference (DAC), pp. 1-6, 2004.

[15] Karl M. Fant, Scott A. Brandt, "NULL Convention Logic : A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis," IEEE International Conference on Application-Speci¯c Systems, Architectures and Processors, pp. 261-273, 1996.

[16] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," Integration, The VLSI Journal, Vol. 37, No. 3, pp. 135-165, 2004.

[17] S. Smith, R. DeMara, J. Yuan, M. Hagedorn and D. Ferguson, "Delay-Insensitive gate-level pipelining," Integration, the VLSI journal, Vol. 30, pp. 103-131, 2000.

[18] R. Bonam, S. Chandhary, Y. Yellambalase and M. Choi, "Clock-Free Nanowire Crossbar Architecture based on Null Convention Logic (NCL)", Accepted to appear in the 7th IEEE International Conference on Nanotechnology (IEEE-Nano), Apr 2007.

[19] R. Bonam, Y. Yellambalase and M. Choi, "Redundancy Optimization for Clock-Free Nanowire Crossbar Architecture", Accepted to appear in the 7th IEEE International Conference on Nanotechnology (IEEE-Nano), Apr 2007.

[20] S. C. Smith, "Gate and Throughput Optimizations for NULL Convention Self-Timed Digital Circuits", Ph.D. Dissertation, School of Electrical Engineering and Computer Science, University of Central Florida, May 2001.

[21] M. Mishra and S. Goldstein, Scalable defect tolerance for molecular electronics, Workshop Non-Silicon Computation (NSC-1), pp. 78, 2002.

[22] M. Tehranipoor "Defect tolerance for molecular electronics-based nanofabrics using built-in self-test procedure", 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005, Oct. 2005 pp 305 - 313

Table 1

Average Column shifts for various gates at different defect rates - Defect Unaware – Shift Approach

Gate	Dimension	5% Defect Rate	10 % Defect Rate	15% Defect Rate
Th12	6x10	2.4629	3.633	6.7
	7x11	2.057	2.8572	4.5
	8x12	1.8251	2.42	3.4
	9x13	1.6596	2.0679	2.8
	10x14	1.5258	1.8731	2.5
	11x15	1.4348	1.719	2.2
Th24	6x10	20.5164	87.0202	1070.8
	7x11	12.5918	30.8044	127.8
	8x12	8.9666	17.7316	47.6
	9x13	7.0065	12.2701	27.6
	10x14	5.8143	9.1379	17.6
	11x15	4.8528	7.5191	13.5
Th34	6x10	13.8244	54.8639	560.7
	7x11	8.0084	19.8859	84.8
	8x12	5.7581	11.3518	32
	9x13	4.4042	7.832	17.5
	10x14	3.6642	5.9707	11.8
	11x15	3.2211	4.8621	8.6
Th34w2	6x10	10.2226	31.8819	218.4
	7x11	6.6096	14.4823	47.8
	8x12	4.9699	9.1901	21.8
	9x13	4.0001	6.6679	13.8
	10x14	3.2657	5.0761	9.2
	11x15	2.8779	4.3285	7.1
Th44w322	6x10	9.7971	26.9044	153
	7x11	6.1777	13.1715	40.9
	8x12	4.7481	8.4291	19.4
	9x13	3.751	6.1887	12.5
	10x14	3.2664	5.0209	8.7
	11x15	2.7991	4.1631	6.8
Th54w322	6x10	7.3379	18.7533	89.3
	7x11	5.1628	10.228	28.5
	8x12	3.9452	6.78	14.8
	9x13	3.1992	5.1192	9.6
	10x14	2.7672	4.2462	7.1
	11x15	2.4355	3.5746	5.6

Table 2

Average AND plane(row) shifts for various gates at different defect rates – Defect Unaware – Modified Shift Approach

Gate	Dimension	10 % Defect Rate	15% Defect Rate	20% Defect Rate
Th12	6x10	1.047	1.243	1.8
	7x11	1.0163	1.0881	1.3
	8x12	1.009	1.043	1.1
	9x13	1.0041	1.026	1.1
	10x14	1.0016	1.016	1
	11x15	1.0016	1.007	1
Th24	6x10	9.1258	106.413	2495.5
	7x11	3.5232	13.4013	81.3
	8x12	2.364	5.1738	16.1
	9x13	1.8577	3.1962	6.4
	10x14	1.6583	2.3156	4.2
	11x15	1.4944	1.9906	3
Th34	6x10	6.0487	48.5051	711.1
	7x11	2.5073	8.2125	46.3
	8x12	1.6886	3.5554	10
	9x13	1.4009	2.278	4.6
	10x14	1.2493	1.7823	2.8
	11x15	1.1695	1.502	2.1
Th34w2	6x10	3.6653	23.9323	197.9
	7x11	1.9743	5.3141	18
	8x12	1.4833	2.7503	6
	9x13	1.3039	2.0113	3.3
	10x14	1.1872	1.5217	2.2
	11x15	1.1276	1.3611	1.8
Th44w322	6x10	3.2297	15.2249	116.6
	7x11	1.8539	4.7673	16.8
	8x12	1.4554	2.5943	4.9
	9x13	1.2777	1.7739	3
	10x14	1.1704	1.489	2.1
	11x15	1.1235	1.332	1.7
Th54w322	6x10	2.3914	10.6571	55.8
	7x11	1.5556	3.3607	10.3
	8x12	1.2859	2.0582	3.8
	9x13	1.177	1.516	2.5
	10x14	1.1124	1.3654	1.8
	11x15	1.074	1.2104	1.4

Table 3

Average PGMB column shifts for various gates at different defect rates – Defect Unaware – Modified Shift Approach

Gate	Dimension	10% Defect Rate	15% Defect Rate	20% Defect Rate
Th12	6x10	3.6927	6.6	13
	7x11	2.8316	4.4	7
	8x12	2.3675	3.5	5
	9x13	2.0614	2.9	4
	10x14	1.8289	2.6	3
	11x15	1.7091	2.2	3
Th24	6x10	86.5781	1060	24947
	7x11	29.8115	129.3	808
	8x12	17.5299	46.4	156
	9x13	11.8936	26	58
	10x14	9.4662	16.9	36
	11x15	7.4874	13.2	23
Th34	6x10	55.8607	480.1	7107
	7x11	19.9413	77.7	458
	8x12	11.1897	30.4	95
	9x13	7.8451	17.4	41
	10x14	6.011	12	23
	11x15	4.8625	8.9	16
Th34w2	6x10	31.8957	234.5	1974
	7x11	14.3311	48.2	175
	8x12	8.8342	22.3	55
	9x13	6.6224	14.4	28
	10x14	5.1097	9.3	17
	11x15	4.2402	7.3	12
Th44w322	6x10	27.3753	147.9	1162
	7x11	13.0505	42.9	163
	8x12	8.4621	20.7	45
	9x13	6.362	11.9	25
	10x14	4.9135	8.6	15
	11x15	4.1094	6.9	11
Th54w322	6x10	19.0009	102	554
	7x11	9.9543	28.7	99
	8x12	6.7085	15.1	33
	9x13	5.1932	9.3	19
	10x14	4.1919	7.6	12
	11x15	3.5152	5.5	8

APPENDIX

MATLAB CODE USED FOR SIMULATIONS

++

Program used to generate a random defect matrix. This is a function used by all the mapping and placement algorithms to place defects onto a given PGMB at given defect rate.

++

```
function def_mat = random_defect(m,n,defect_rate);
defects = 0;
def_mat = zeros(m,n);                    % initialize the defect matrix
max_defects = (defect_rate*6*10)/100;    % calculate the maximum defect tolerance
                                         using the defect rate

 d1 = ceil(m.*rand(100,1));              % generates uniform set of integers in the
                                         interval of 1:m
 d2 = ceil(n.*rand(100,1));              % generates uniform set of integers in the
                                         interval of 1:n

                                         %initialize variables
 i=1;                                    %index of random defect location
 j=1;                                    %index of random defect location
%place defects onto the PGMB
while(defects<max_defects)

    if(i<100 && j <100)
       if(def_mat(d1(i),d2(j))~=255)
                 def_mat(d1(i),d2(j)) = 255;
                 i = i+1;
                 j=j+1;
                 defects = defects + 1;
          else
```

```
                i = i+1;
                j = j+1;
        end
        else
                disp('out of random defects');
        end
  end
end
```

++

Program used to simulate the Defect Unaware Approach. It programs logic of 6 threshold gate's logic onto a million gates each to effectively simulate the algorithm and plots the programmability against different dimensions.

++

```
clear all;
clc;
th =    [1 0 1 0 0 0 0 0 0 0;
         0 1 0 1 0 0 0 0 0 0;
         0 0 1 1 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0;
         1 1 1 1 0 0 0 0 0 0];                  % TH12 represented in a matrix
                                                format
th(:,:,2) =     [1 1 1 0 0 0 1 0 0 0;
                 1 0 0 1 1 0 0 1 0 0;
                 0 1 0 1 0 1 0 0 1 0;
                 0 0 1 0 1 1 0 0 0 1;
                 0 0 0 0 0 0 1 1 1 1;
                 1 1 1 1 1 1 1 1 1 1];          % TH24 represented in a matrix
                                                format
th(:,:,3) =     [1 1 1 0 1 0 0 0 0 0;
                 1 1 0 1 0 1 0 0 0 0;
                 1 0 1 1 0 0 1 0 0 0;
                 0 1 1 1 0 0 0 1 0 0;
                 0 0 0 0 1 1 1 1 0 0;
                 1 1 1 1 1 1 1 1 0 0];          % TH34 represented in a matrix
                                                format
th(:,:,4) =     [1 1 1 0 1 0 0 0 0 0;
                 1 0 0 1 0 1 0 0 0 0;
                 0 1 0 1 0 0 1 0 0 0;
```

```
                0 0 1 1 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];      % TH34w2 represented in a matrix
                                            format
th(:,:,5) =     [1 1 1 0 1 0 0 0 0 0;
                1 0 0 1 0 1 0 0 0 0;
                0 1 0 1 0 0 1 0 0 0;
                0 0 1 0 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];      % TH44w322 represented in a
                                            matrix format
th(:,:,6) =     [1 1 0 1 0 0 0 0 0 0;
                1 0 1 0 1 0 0 0 0 0;
                0 1 1 0 0 1 0 0 0 0;
                0 0 1 0 0 0 1 0 0 0;
                0 0 0 1 1 1 1 0 0 0;
                1 1 1 1 1 1 1 0 0 0];      % TH54w322 represented in a
                                            matrix format

rows_cols = [6 7 8 9 10 11;
        10 11 12 13 14 15];                % Different dimensions of PGMBs
gate_count = 1000000;                      % million gates to be programmed
                                           % variables for use in programming
placed_PGMB = [];                          %count of placed PGMBs
gates = [];
placed = [];                               %count of placed crosspoints
                                           %index variables for loops
l = 0;
m = 0;
i = 0;
j = 0;
```

```
k = 0;
gate_placed = 0;                                %count of placed gates
cpoint_count = 0;                               %count of crosspoints in threshold
                                                gates
acc = [];                                       %to store accuracy of each gate
accuracy = [];                                  %to store cumulative accuracy of
                                                gates
 [size_rows,size_cols] = size(rows_cols);

def_rate = 10; %[5 10 15 20 25];                %different defect rates
[def_rate_rows def_rate_cols] = size(def_rate);
    for k = 1:1:6
       for j = 1:1:numel(def_rate)
          for  n = 1:1:gate_count
             def_mat = random_defect(rows_cols(1,1),rows_cols(2,1),def_rate(j));
             placed_PGMB = def_mat;
             placed = 0;
             cpoint_count = 0;
             for l = 1:1:6
                for m = 1:1:10
                   if(th(l,m,k) == 1)
                      cpoint_count = cpoint_count + 1;
                   end
                   if(th(l,m,k) == 1 && def_mat(l,m) == 0)
                      placed_PGMB(l,m) = 10;
                      placed = placed + 1;
                   end
                end
             end
             if(placed == cpoint_count)
                 gate_placed = gate_placed + 1;
```

```
            end
        end
        acc = [acc gate_placed/gate_count];
        gate_placed = 0;

    end
  accuracy = [accuracy;acc];
  acc = [];

  end
cols = [0;
  1;
  2;
  3;
  4;
  5];

bar(cols,accuracy');                    % plots programmability against the
                                        PGMB dimensions
colormap(gray);
title(sprintf('Defect Unaware approach - Programmability of various gates at various de-
fect rates for 6x10 crossbar'));
legend('TH12','TH24','TH34','TH34w2','TH44w322','TH54w322');
xlabel('Defect Rate');
ylabel('Programmability');
 axis square; grid on
```

++

Program used to simulate Defect Unaware – Circular shift approach. It programs logic of 6 threshold gate's logic onto a million gates each to effectively simulate the algorithm and plots the programmability against different dimensions.

++

```
clear all;
clc;
th =    [1 0 1 0 0 0 0 0 0 0;
         0 1 0 1 0 0 0 0 0 0;
         0 0 1 1 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0;
         1 1 1 1 0 0 0 0 0 0];                  % TH12 represented in a matrix
                                                format
th(:,:,2) =     [1 1 1 0 0 0 1 0 0 0;
                 1 0 0 1 1 0 0 1 0 0;
                 0 1 0 1 0 1 0 0 1 0;
                 0 0 1 0 1 1 0 0 0 1;
                 0 0 0 0 0 0 1 1 1 1;
                 1 1 1 1 1 1 1 1 1 1];          % TH24 represented in a matrix
                                                format
th(:,:,3) =     [1 1 1 0 1 0 0 0 0 0;
                 1 1 0 1 0 1 0 0 0 0;
                 1 0 1 1 0 0 1 0 0 0;
                 0 1 1 1 0 0 0 1 0 0;
                 0 0 0 0 1 1 1 1 0 0;
                 1 1 1 1 1 1 1 1 0 0];          % TH34 represented in a matrix
                                                format
th(:,:,4) =     [1 1 1 0 1 0 0 0 0 0;
                 1 0 0 1 0 1 0 0 0 0;
                 0 1 0 1 0 0 1 0 0 0;
```

```
                0 0 1 1 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];          % TH34w2 represented in a
                                               matrix format
th(:,:,5) =     [1 1 1 0 1 0 0 0 0 0;
                1 0 0 1 0 1 0 0 0 0;
                0 1 0 1 0 0 1 0 0 0;
                0 0 1 0 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];          % TH44w322 represented in
                                               a matrix format
th(:,:,6) =     [1 1 0 1 0 0 0 0 0 0;
                1 0 1 0 1 0 0 0 0 0;
                0 1 1 0 0 1 0 0 0 0;
                0 0 1 0 0 0 1 0 0 0;
                0 0 0 1 1 1 1 0 0 0;
                1 1 1 1 1 1 1 0 0 0];          % TH54w322 represented in
                                               a matrix format

rows_cols = [6 7 8 9 10 11;
        10 11 12 13 14 15];                    %Different Dimensions of
                                               PGMBs
gate_count = 1000000;                          % number of gates to be
                                               programmed
placed_PGMB = [];                              % count of successfully
                                               programmed PGMBs
gates = [];
placed = 0;                                    % count of placed
                                               crosspoints.

l = 0;                                         %index variables
```

```
m = 0;
i = 0;
j = 0;
k = 0;
gate_placed = 0;                    %count of PGMBs placed
cpoint_count = 0;                   % count of number of crosspoints in
                                    a particular gate.
circ_shift_count = 0;               %count of circular shifts per
                                    successfully programmed gate
shift_count = 0;
average_shifts = [];                %average shifts per programmed
                                    gate
acc = [];
accuracy = [];                      %to store cumulative accuracy of
                                    gates
[size_rows,size_cols] = size(rows_cols);

def_rate = 10 %[5 10 15 20 25];     % defect rate for placing in the
                                    PGMB
[def_rate_rows def_rate_cols] = size(def_rate);
cols = [0;
    1;
    2;
    3;
    4;
    5];
     for k = 1:1:6
        cpoint_count = 0;
        placed = 0;
        for l = 1:1:6
            for m = 1:1:10
```

```
        if(th(l,m,k) == 1)
            cpoint_count = cpoint_count + 1;
        end
    end
end
for j = 1:1:size_cols
   placed = 0;
   for  n = 1:1:gate_count
      circ_shift_count = 0;
      def_mat = random_defect(rows_cols(1,j),rows_cols(2,j),def_rate);
      while(placed < cpoint_count && circ_shift_count < 10)
         th(:,:,k) = circshift(th(:,:,k),[0,1]);

         circ_shift_count  = circ_shift_count + 1;
         shift_count = shift_count + 1;
         placed_PGMB = def_mat;
         placed = 0;
         %cpoint_count = 0;
         for l = 1:1:6
            for m = 1:1:10
               if(th(l,m,k) == 1 && def_mat(l,m) == 0)
                  placed_PGMB(l,m) = 10;
                  placed = placed + 1;
               end
            end
         end
         if(placed == cpoint_count)
            gate_placed = gate_placed + 1;
            placed = 0;
            break;
         end
```

```
            end

        end
        acc = [acc gate_placed/gate_count];
        average_shifts = [average_shifts ; (shift_count/gate_placed)];
        shift_count = 0;
        gate_placed = 0;
    end
    accuracy = [accuracy;acc]; %
    acc = [];
  end

bar(cols,accuracy');
colormap(gray);
title(sprintf('Circular Shift Algorithm - Programmability of various gates at various defect rates for 6X10 crossbar'));
legend('TH12','TH24','TH34','TH34w2','TH44w322','TH54w322');
xlabel('Defect Rate');
ylabel('Programmability');
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Program used to simulate Defect Unaware – Modified shift approach. It programs logic of 6 threshold gate's logic onto a million gates each to effectively simulate the algorithm and plots the programmability against different dimensions.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
clear all;
clc;
th =    [1 0 1 0 0 0 0 0 0 0;
         0 1 0 1 0 0 0 0 0 0;
         0 0 1 1 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0;
         1 1 1 1 0 0 0 0 0 0];                  % TH12 represented in a matrix
                                                format
th(:,:,2) =     [1 1 1 0 0 0 1 0 0 0;
                 1 0 0 1 1 0 0 1 0 0;
                 0 1 0 1 0 1 0 0 1 0;
                 0 0 1 0 1 1 0 0 0 1;
                 0 0 0 0 0 0 1 1 1 1;
                 1 1 1 1 1 1 1 1 1 1];          % TH24 represented in a matrix
                                                format
th(:,:,3) =     [1 1 1 0 1 0 0 0 0 0;
                 1 1 0 1 0 1 0 0 0 0;
                 1 0 1 1 0 0 1 0 0 0;
                 0 1 1 1 0 0 0 1 0 0;
                 0 0 0 0 1 1 1 1 0 0;
                 1 1 1 1 1 1 1 1 0 0];          % TH34 represented in a matrix
                                                format
th(:,:,4) =     [1 1 1 0 1 0 0 0 0 0;
                 1 0 0 1 0 1 0 0 0 0;
                 0 1 0 1 0 0 1 0 0 0;
```

```
                0 0 1 1 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];      % TH34w2 represented in a matrix
                                            format
th(:,:,5) =     [1 1 1 0 1 0 0 0 0 0;
                1 0 0 1 0 1 0 0 0 0;
                0 1 0 1 0 0 1 0 0 0;
                0 0 1 0 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];      % TH44w322 represented in a
                                            matrix format
th(:,:,6) =     [1 1 0 1 0 0 0 0 0 0;
                1 0 1 0 1 0 0 0 0 0;
                0 1 1 0 0 1 0 0 0 0;
                0 0 1 0 0 0 1 0 0 0;
                0 0 0 1 1 1 1 0 0 0;
                1 1 1 1 1 1 1 0 0 0];      % TH54w322 represented in a
                                            matrix format

rows_cols = [6 7 8 9 10 11;
        10 11 12 13 14 15];                %Different Dimensions of PGMB
gate_count = 1000000;                      % number of gates to be
                                            programmed

placed_PGMB = [];                          %count of placed PGMBs
th_temp = [];
gates = [];
placed = 0;                                %count of placed crosspoints

a = 0;                                     %index variables
l = 0;
m = 0;
i = 0;
```

```
j = 0;
k = 0;
gate_placed = 0;                          % count of gates programmed
gate_placed_prev = 0;
cpoint_count = 0;                         %number of crosspoints in the gate
circ_shift_count = 0;                     % count of circular shifts
acc = [];
row_count = 0;
col_count = 0;
accuracy = [];                            % to store programmability of each
                                          of the gates

[size_rows,size_cols] = size(rows_cols);

def_rate = [5 10 15 20 25 30]; % defect rate
[def_rate_rows def_rate_cols] = size(def_rate);
% start with getting the crosspoint count
for k = 1:1:6
    th_temp = [];
   cpoint_count = 0;
   placed = 0;
   for l = 1:1:6
      for m = 1:1:10
         if(th(l,m,k) == 1)
            cpoint_count = cpoint_count + 1;
         end
         if(l<6)
            th_temp(l,m) = th(l,m,k);
         end
      end
   end
```

```
 % start placing crosspoints…
for j = 1:1:size_cols

     placed = 0;
    for n = 1:1:gate_count
      for a = 1:1:5
        if(gate_placed == gate_placed_prev)
              th_temp = circshift(th_temp, [1,0]);

          for row_count = 1:1:5
             for col_count = 1:1:10
               th(row_count,col_count,k) = th_temp(row_count,col_count);
             end
          end
          circ_shift_count = 0;

          def_mat = random_defect(rows_cols(1,j),rows_cols(2,j),def_rate);
          while(placed < cpoint_count && circ_shift_count < 10)
                th(:,:,k) = circshift(th(:,:,k),[0,1]);

            circ_shift_count  = circ_shift_count + 1;

            placed_PGMB = def_mat;
            placed = 0;
            %cpoint_count = 0;
            for l = 1:1:6
                for m = 1:1:10
                  if(th(l,m,k) == 1 && def_mat(l,m) == 0)
                     placed_PGMB(l,m) = 10;
                      placed = placed + 1;
                  end
```

```
                end
            end
            if(placed == cpoint_count)
                gate_placed_prev = gate_placed;
                gate_placed = gate_placed + 1;
                placed = 0;
                break;
            end
          end
        elseif (gate_placed > gate_placed_prev)
          break;
        end

      end

      gate_placed_prev = gate_placed;
    end
     acc = [acc gate_placed/gate_count];

     gate_placed = 0;

  end
  accuracy = [accuracy;acc];
  acc = [];

end
cols = [0;
  1;
  2;
  3;
  4;
```

```
  5];
gate = [10 10 10 10 10 10 20 20 20 20 20 20 30 30 30 30 30 30 40 40 40 40 40 40 50 50
50 50 50 50 60 60 60 60 60 60];
bar(cols,accuracy');                                    %plots programmability
                                                        against size of PGMBs

title(sprintf('Modified Circular Shift Algorithm - Programmability of Various Gates at
various defect rate 6X10 crossbar'));
legend('TH12','TH24','TH34','TH34w2','TH44w322','TH54w322');
xlabel('Defect Rate');
ylabel('Time');
colormap(gray);
```

++

Program used to simulate Defect Aware Approach algorithm. It programs logic of 6 threshold gate's logic onto a million gates each to effectively simulate the algorithm and plots the programmability against different dimensions.

++

```
clear all;
clc;

   l = 1;                                  %index variables
   x = 1;
   y = 1;
   z = 1;
   p = 1;

   failed = 0;                             %flag variable
   row_watch = [];                         %count watch variables
   col_watch = [];
   count_watch = [];
   count_placable = [];                    %to store available
                                           crosspoints
   accuracy_cumulative = [];               %stores cumulative
                                           programmability
   avail_rows = [];                        %store available row indexes
   th23_row = [];
   temp = [];
   gt = 0;
   acc = 0;
   th23_avail = [];
   programmable = [];
```

```
    placed = 0;                                    %stores count of placed
                                                   crosspoints

    placed_mat = [];
    accuracy = [];
    delim = [50;
             50;
             50;
             50;
             50;
             50;
             50;
             50;
             50];                                  %Delimiter variable for
                                                   appending with available
                                                   rows
rows = [6 7 8 9 10 11];                            %row values for PGMB
cols = [10 11 12 13 14 15];                        % column values for PGMB
prog = [];
ones = 0;

def_rate = 10; % [5 10 15 20 25 30 35 40 45 50];
total = 0;
th_23 = [1 0 1 0 0 0 0 0 0 0;
         0 1 0 1 0 0 0 0 0 0;
         0 0 1 1 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0 0;
         1 1 1 1 0 0 0 0 0 0];                     % TH12 represented in a matrix
                                                   format
th_23(:,:,2) =   [1 1 1 0 0 0 1 0 0 0;
```

```
                1 0 0 1 1 0 0 1 0 0;
                0 1 0 1 0 1 0 0 1 0;
                0 0 1 0 1 1 0 0 0 1;
                0 0 0 0 0 0 1 1 1 1;
                1 1 1 1 1 1 1 1 1 1];      % TH24 represented in a matrix
                                           format
th_23(:,:,3) =  [1 1 1 0 1 0 0 0 0 0;
                1 1 0 1 0 1 0 0 0 0;
                1 0 1 1 0 0 1 0 0 0;
                0 1 1 1 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];      % TH34 represented in a matrix
                                           format
Th_23(:,:,4) =  [1 1 1 0 1 0 0 0 0 0;
                1 0 0 1 0 1 0 0 0 0;
                0 1 0 1 0 0 1 0 0 0;
                0 0 1 1 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];      % TH34w2 represented in a matrix
                                           format
Th_23(:,:,5) =  [1 1 1 0 1 0 0 0 0 0;
                1 0 0 1 0 1 0 0 0 0;
                0 1 0 1 0 0 1 0 0 0;
                0 0 1 0 0 0 0 1 0 0;
                0 0 0 0 1 1 1 1 0 0;
                1 1 1 1 1 1 1 1 0 0];      % TH44w322 represented in a
                                           matrix format
Th_23(:,:,6) =  [1 1 0 1 0 0 0 0 0 0;
                1 0 1 0 1 0 0 0 0 0;
                0 1 1 0 0 1 0 0 0 0;
                0 0 1 0 0 0 1 0 0 0;
```

```
                0 0 0 1 1 1 1 0 0 0;
                1 1 1 1 1 1 1 0 0 0];          % TH54w322 represented in a
                                               matrix format

gate_count = 1000000;                          %number of gates to be programmed
   for g = 1:1:6
      % Scans the gate to be programmed to get positions of crosspoints.
      th23_avail = [];
      cross_count = 0;


      for i = 1:1:10
         for j = 1:1:5
            if(th23(j,i,g) == 1)
              cross_count = cross_count + 1;
              th23_avail = [th23_avail i j];
            end
         end
         th23_avail = [th23_avail 150];
      end



      for u = 1:1:6

         if(((rows(1,u)*cols(1,u))-
((def_rate*rows(1,u)*cols(1,u))/100))>=(1.5*cross_count))
%             n = ceil((20*rows(1,u))/100) - 1; % uses 20% of the rows for OR plane
            n = rows(1,u) - 5;
            m = 5;
            i = 1;

            prgmd_gates = 0;
            for gt = 1:1:gate_count
               def_mat = random_defect(rows(1,u),cols(1,u),def_rate);%(1,u));

               failed = 0;
```

```
row_watch = [];
col_watch = [];
count_watch = [];
count_placable = [];
avail_rows = [];
th23_row = [];
temp = [];
programmable = [];

placed = 0;

for j = rows(1,u)-n+1:1:rows(1,u)
   for k = 1:1:cols
      if(def_mat(j,k) == 0)
         row_watch = [row_watch;j k];
      end
   end
end

[row_watch_row row_watch_col] = size(row_watch);
for j = 1:1:cols
   for k = 1:1:rows(1,u)-n
      if(def_mat(k,j) == 0)
         col_watch = [col_watch;j k];
      end
   end
end

[col_watch_row col_watch_col] = size(col_watch);
count = 1;
j=0;
while(count<row_watch_row)
   if(row_watch(count,1) == row_watch(count+1,1))
      j = j+1;
   else
   count_watch = [count_watch;row_watch(count) j+1];
   j = 0;
   end
   if(count == row_watch_row-2)
      count_watch = [count_watch;row_watch(count) j+2];
   end
   count = count+1;
end
[count_watch_rows count_watch_cols] = size(count_watch);
k = 1;
l = 1;
```

```
%               placed = 0;
              for t = 1:1:count_watch_rows
                failed = 0;
%                   placed = 0;
                if(count_watch(t,2)>=th23_OR && placed < cross_count)
                  placed = 0;
                  avail_rows = [];
                  for k = 1:1:row_watch_row
                    if(count_watch(t,1) == row_watch(k,1))
                      for l = 1:1:col_watch_row
                        if(row_watch(k,2) == col_watch(l,1))
                          avail_rows = [avail_rows col_watch(l,1) col_watch(l,2)];
                        end
                      end
                      avail_rows = [avail_rows 150];
                    end
                  end
                  [avail_rows_rows avail_rows_cols] = size(avail_rows);
                  [th23_avail_rows th23_avail_cols] = size(th23_avail);
                  y=1;
                  x=2;
                  c=1;
                  i=2;
                  j=2;
                  while(j<th23_avail_cols)
                    while(th23_avail(1,y) ~= 150)
                      y = y + 1;
                    end
                    while(i<avail_rows_cols)
                      z = i;
                      while(avail_rows(1,z)~=150)
                        z = z+1;
                      end
                      p = z;
                      while(j<y && (avail_rows(1,z) ~= 150 || avail_rows(1,z-1) ~=
150))

                        for z = i:2:p
                          if(th23_avail(1,j) == avail_rows(1,z))
                            th23_row(z) = 1;
                          end
                        end
                        j = j + 2;
                      end
                      if(avail_rows(1,i-1) == 150)
                        i = i - 1;
                      end
```

```
        while(avail_rows(1,i) ~= 150)
            i = i + 1;
        end
        th23_row(i) = 150;
        i=i+2;
        j=x;
    end

    b = 1;
    k = 0;
    [th23_row_rows th23_row_cols] = size(th23_row);
    for a = 1:1:th23_row_cols
        if(th23_row(a) == 1)
            k = k + 1;
        end
        if(th23_row(a) == 150)
            if(k<1)
                k = 0;
            end
            if(k>1 || k == 1)
                programmable(c) = avail_rows(a-2);
                c = c + 1;
                b = b + 1;
                k = 0;
            end
        end
    end
    programmable(1,c) = 150;
    c = c+1;
    j = y+2;
    i = 2;
    y = y+1;
    x = j;
    th23_row = [];
end
[r s] = size(programmable);
placed = 0;

q = 0;
for r = 1:1:s
    if(programmable(1,r) ~=150)
        q = q + 1;
    end
    if(programmable(1,r) == 150)
        count_placable = [count_placable q];
        q = 0;
```

```
    end
end

[q count] = size(count_placable);
for f = 1:1:10
    if(th23(6,f,g) == 1)
        ones = ones+1;
    end
end
for q = 1:1:ones
    if(count_placable(1,q) == 0 || count_placable(1,q) == 150)
        failed = 1;
    end
end
while(placed < cross_count && failed == 0)
    least = 1;
    for q = 1:1:ones
        if(count_placable(1,least)>count_placable(1,q))
            least = q;
        end
    end
    q = 1;
    found = 1;
    for q = 1:1:s
        if(programmable(1,q)==150)
            found = found + 1;
        end
        if(found == least)
            while(programmable(1,q+1) ~= 150 && programmable(1,q+1)
== 0)

                q = q+1;
                if(programmable(1,q+1) ==150)
                    failed = 1;
                end
            end
            l = programmable(1,q+1);
            break;
        end
    end
    if(failed == 0)
        def_mat(count_watch(t,1),l)= 100;

        for q = 1:1:5
            if(th23(q,least,g) == 1)
                def_mat(q,l)= 100;
                placed=placed+1;
```

```
                        end
                    end

                    for q = 1:1:s
                        if(programmable(1,q) == l)
                            programmable(1,q) = 0;
                        end
                    end
%                       placed = placed + 2;
                    count_placable(1,least) = 150;
                end

                end
                ones = 0;
            end
            if(placed >= cross_count) % failed == 0
                prog_failed = 0;
                break;
%                   disp('programming failed');
            end
        end % count_watch_rows for loop
        if(prog_failed == 0)
            prgmd_gates = prgmd_gates + 1;

        end % gate count for loop
        prog_failed = 1;

    end%def_rate end
    acc = prgmd_gates / gate_count;

else
    acc = 0;
    disp('defect rate too high!');

end
accuracy = [accuracy; acc];

end
prog = [prog accuracy];
accuracy = [];
end


bar(def_rate,prog);

colormap(gray);
```

```
title(sprintf('Defect Aware approach - Programmability of various gates at various defect rate 6X10 Crossbar'));
legend('TH12','TH24','TH34','TH34w2','TH44w322','TH54w322');
xlabel('Defect Rate');
ylabel('Programmability');

axis square; grid on
```

VITA

Ravi Kiran Bonam was born in Srikakulam, Andhra Pradesh, India, on the 4th of August 1985. Most of his schooling was in Visakhapatnam, Andhra Pradesh, India. He received a bachelor's degree in Computer Science and Engineering from Andhra University, Visakhapatnam in April, 2006. He enrolled into the Master's Degree program in Computer Engineering in August 2006 at the Missouri University of Science and Technology (formerly University of Missouri – Rolla), Rolla, Missouri, U.S.A, and graduated in May 2008.