ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE

RELIABILITY AND COMPUTING TECHNIQUES FOR NANO SWITCHING ARRAYS

M.Sc. THESIS

Onur TUNALI

Department of Nanoscience and Nanoengineering

Nanoscience and Nanoengineering Programme

DECEMBER 2015

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE

RELIABILITY AND COMPUTING TECHNIQUES FOR NANO SWITCHING ARRAYS

M.Sc. THESIS

Onur TUNALI
(513131018)

Department of Nanoscience and Nanoengineering

Nanoscience and Nanoengineering Programme

Thesis Advisor: Asst. Prof. Mustafa ALTUN

DECEMBER 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

NANO ANAHTARLAMALI DİZİNLER İÇİN GÜVENİLİRLİK VE HESAPLAMA TEKNİKLERİ

YÜKSEK LİSANS TEZİ

Onur TUNALI
(513131018)

Nano-Bilim ve Nano-Mühendisliği Anabilim Dalı

Nano-Bilim ve Nano-Mühendislik Programı

Tez Danışmanı: Asst. Prof. Mustafa ALTUN

ARALIK 2015

Onur TUNALI, a M.Sc. student of ITU Graduate School of ScienceEngineering and Technology 513131018 successfully defended the thesis entitled "RELIABILITY AND COMPUTING TECHNIQUES FOR NANO SWITCHING ARRAYS", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Mustafa ALTUN** Istanbul Technical University	.....
Jury Members :	**Asst. Prof. Mustafa ALTUN** Istanbul Technical University	.....
	Prof. Müştak Erhan YALÇIN Istanbul Technical University	.....
	Asst. Prof. İlke ERCAN Boğaziçi University	.....

Date of Submission : **27 November 2015**
Date of Defense : **22 December 2015**

To my mother and dear friends,

FOREWORD

I would like to thank my advisor Dr. Mustafa Altun for his guidance and support throughout my graduate school experience.

I would like to acknowledge the support of TUBITAK Bideb c-2210 program scholarship.

Last but not least, for all their effort and support, I would like to thank my mother Meliha Yılanlı and my dear friends Hakkı Düzgün, Mehmet Aklabık, Aytaç Şahin and Ceylan Güney.

December 2015

Onur TUNALI
Mathematician

TABLE OF CONTENTS

Page

ABBREVIATIONS

PLA	**:** Programmable Logic Array
$\mathbf{I_{C,F}}$	**:** Column Index Set for logic function
$\mathbf{I_{R,F}}$	**:** Row Index Setfor logic function
$\mathbf{I_{C,C}}$	**:** ColumnIndex Sets for crossbar function
$\mathbf{I_{R,C}}$	**:** Row Index Set for crossbar function
IR	**:** Logic Inclusion Ratio
SOP	**:** Sum of Products
PI	**:** Prime Implicant
ISOP	**:** Irredundant Sum of Products
CMOS	**:** Complementary Metal Oxide Semiconductor
FET	**:** Field Effect Transistor

LIST OF TABLES

Page

LIST OF FIGURES

Page

RELIABILITY AND COMPUTING TECHNIQUES FOR NANO SWITCHING ARRAYS

SUMMARY

Lithographic top-down based production of integrated circuits are approaching the limits in a manner of both feasibility and commercial aspects. In spite of the fact that, Moore's Law keeps holding, emerging technologies need to be considered. Crossbar based nano switching arrays are shown to be a likely candidate to overcome shortcomings of current CMOS based paradigm or coexist as a complementary instrument. Abundant research papers in literature help to support this claim. Nano-arrays are produced with placing a group of nanowires aligned parallel to each other on another group of nanowires orthogonally. Crosspoints present between top and bottom nanowires act as a switching device. According to the preference, switches might show resistor, diode or FET like characteristics.

Computing with nano-arrays are similar to the Programmable Logic Arrays (PLA). Every switch can be appointed to the corresponding logic element found in the boolean function which is realized with the crossbar in question. Nevertheless, the nature of nano-fabrication contains random elements and devices obtained from the process are prone to have faulty components. As a result, realization of target logic functions with nano-arrays differ from PLA due to the number of considerable faulty components.

Since discarding faulty devices would not be practical and sustainable, fault tolerance and reliability of crossbar based nano switching arrays are extensively studied in this thesis. Most common faults occur in described switches can be categorized under two main titles which are permanent and transient. Also, two categories have subtitles such as stuck-open, stuck-closed and nanowire break-downs. Because of the immense effect of nanowire break-downs, they are excluded from the body of study.

Permanent faults are taken into account by independently assigning stuck-open and stuck-closed defect probabilities into crosspoints. After obtaining defective array, following step is determining whether there is a valid mapping of a given logic function on defective array. In the presence of permanent faults, a heuristic algorithm using index sorting, backtracking and matrix multiplication techniques is proposed. The algorithm's effectiveness is demonstrated on standard benchmark circuits that shows 99% accuracy in accordance with the results of an exhaustive search algorithm. Runtime and success rate of algorithm is presented with experimental results of simulation using standard industry benchmark circuits.

In the presence of transient faults, tolerance analysis is performed by recursively constructing equivalent sets of implemented logic functions. It is demonstrated that transient faults causing OFF-to-ON state changes in crosspoints do not necessarily cause the array to produce an incorrect output; they can be discarded. Difference between the assumed and the actual fault tolerance performances, which is obtained with the proposed algebraic method, is presented with standard benchmark circuits for several fault rates.

NANO ANAHTARLAMALI DİZİNLER İÇİN GÜVENİLİRLİK VE HESAPLAMA TEKNİKLERİ

ÖZET

Ticari ve uygulama yönü ele alındığında, yukarıdan aşağıya litografik entegre-devre üretimi limitine ulaşmaktadır. Moore Yasası'nın öngörüsü geçerliliğini sürdürse de yeni ortaya çıkan ve alternatif teknolojiler göz önünde bulundurulmalıdır. En güncel Yarıiletkenler için Uluslararası Teknoloji Yol Haritası raporlarında da belirtildiği gibi alternatif teknoloji arayışları devam etmektedir.

Özellikle nano boyuta inildiğinde ortaya çıkan sızıntı, hatalı üretimin yüksekliği gibi transistor sorunları, CMOS teknolojisinin üstesinden gelmesi gereken zorlukların en önemlileridir. Bahsedilen konular bu alanlarda çalışan araştırmacıları hesaplama, hafıza gibi devre yapılarında kullanılmak üzere farklı yaklaşımlar ve mimariler tasarlamaya itmiştir.

CMOS teknolojisi göz önünde bulundurulduğunda yeni ortaya çıkan teknolojiler fiziksel açıdan CMOS'a benzer ve benzer olmayan şeklinde iki kategoriye ayrılabilir.

Fiziksel açıdan CMOS teknolojisine benzer yapılar, silikon nano-teller ve karbon nano-tüpler kullanarak devre elemanlarını üretir. Çalışmada odaklanılan ızgara tabanlı nano dizinler bu yaklaşımın bir örneğidir.

Fiziksel açıdan CMOS teknolojisine benzer olmayan yapılar, kuantum hücresel otomat, spintronik, tek elektron transistörleri, moleküler elektronik, DNA ve biyolojik hesaplamadır.

Yeni ortaya çıkan teknolojilerin üretim teknikleri, yukarıdan aşağıya veya aşağıdan yukarıya yaklaşımlar şeklinde iki ana kategori altında toplanabilir.

Yukarıdan aşağıya teknikler klasik litografi üretiminin iyileştirilmesi şeklinde ilerlemektedir ve marjinal fayda gün geçtikçe azalmaktadır.

Aşağıdan yukarıya teknikler ise devre elemanlarının tek başına üretilip daha sonra montajlanmasına dayanır. Bu yaklaşımın avantajı yüksek derecede düzenli yapılar oluşturmaya elverişli olmasına rağmen elde edilen elemanların geleneksel üretim paradigmasına göre yüksek düzeyde hatalı eleman içermesidir. Tezde odaklanılan teknoloji ızgara yapısına benzer nano anahtarlamalı dizinlerdir.

Araştırmacıların gösterdiği gibi ızgara şeklinde üst üste yerleştirilmiş nano-tellerin kesişim (jonksiyon) noktaları yarı iletkenlik özelliklerine göre direnç, diyot veya FET benzeri yapılar ortaya çıkarmıştır. Bu özellikten yararlanan ızgara tabanlı nano anahtarlamalı dizinler, CMOS teknolojisinin eksikliklerinin üstesinden gelmeye veya eksiklerini tamamlayıcı bir enstrüman olma konusunda olası bir adaydır. Literatürdeki çalışmaların yoğunluğu bu iddiayı destekler niteliktedir.

Nano dizinlerler hesaplama gerçekleştirmek için ortaya atılan farklı mimariler ayrıntılı bir şekilde incelenmiş, aralarında farklar ve benzerlikler yapıya özgü karakteristik özellikleri göz önünde bulundurularak açıklanmıştır.

Teorik bir şekilde modellenmiş yapıların yanı sıra fiziksel olarak gerçeklenmiş işlemci ve sonlu durum makineleri de anlatılmıştır.

Tezin gövdesini, bu ızgara yapıların lojik sentezinde ve hesaplamada kullanılması, lojik fonksiyonların girdilerinin dağılımlarının belirlenmesi ve yapıda oluşan hatalara rağmen lojik fonksiyonun verilen ızgara yapıyla gerçeklenmesi oluşturur. Ayrıca, üretim sürecinden sonra ortaya çıkan geçici hataların devre üzerindeki etkileri ve güvenilirlik analizi de göz önünde bulundurulmuştur.

Nano üretim doğası gereği rasgele süreçler içerir ve üretilen yapılar hatalı elemanlar içermeye yatkındır. Tezin odak noktası üretimde oluşan hatalar sonucu çalışmayan anahtarların sürece nasıl dahil edileceğidir. Hem nano-tellerin üretilmesi hem de istenilen yapıların oluşturulması için gerekli teknoloji oldukça pahalı ve zaman alıcı olduğundan son ürünün hatalı olması sonucu ıskartaya çıkması söz konusu değildir. Bu yüzden hatalı ürünlerin dolaşıma yeniden sokulması gerekir.

Üretim öncesi ve sonrası ortaya çıkan hatalar iki ana başlık altında incelenebilir: kalıcı ve geçici hatalar. Bu hata çeşitleri ayrıca üç alt başlığa ayrılır: açık-durumda takılı kalmış, kapalı-durumda takılı kalmış hatalar ve nano-tel kırılmaları. Nano-tel kırılmalarının devreye etkilerinin büyüklüğü yüzünden araştırmanın içeriğine dâhil edilmemiştir.

Kalıcı hataların telafisi için sunulan algoritma lojik fonksiyonu ve hatalı nano-dizini incelemek için matris modelini kullanmaktadır. Algoritmanın amacı iki matris arasında bir eşleme bulmaktır. Algoritmanın yaralandığı buluşsal (*Heuristic*) yaklaşımlar indeks sıralaması, geri-izleme ve tek tek eleman çarpımlı matris çarpımı teknikleridir.

İndeks sıralaması, lojik ve nano-dizin matrisine eşlenmesi gereken elemanların sayılarına göre satır ve sütun değişimleri uygular. Geri-izleme önceden eşlenmiş bölümlerin takibini ve yeniden eşlemeye sokulmasını düzenler. Tek tek eleman çarpımlı matris çarpımı iki matris arasında eşleme olup olmadığını ortaya çıkarır.

Kalıcı hataların telafisi için izlenen yol, lojik sentez yaparken hatalardan kaçınılması veya hataların kullanılması şeklindedir. Bu çalışmada hatalar lojik sentez işlemine dahil edilmiş bir başka ifadeyle kullanılmıştır. Deneysel sonuçlar için anahtar görevi gören kesişim noktalarına rasgele hata atamaları yapılmıştır. Daha sonra standart bençmark devrelerinin, hatalı dizinle gerçeklenmesi veya gereçeklenememesi incelenmiştir.

Sunulan algoritma tüm olasılıkları göz önünde bulunduran kaba kuvvet algoritmasıyla karşılaştırıldığında %99 doğruluk oranı elde edilmiştir. Ek olarak algoritmanın her bençmark fonksiyonu için ihtiyaç duyduğu çalışma süreleri de deneysel sonuçlar kısmında belirtilmiş ve diğer algoritmalarla karşılaştırmaları sunulmuştur.

Üretim sonrası gerçekleştirilen lojik tasarım, hatalı yapıların yol açtığı bireysel düzenlemeden ötürü tasarım algoritmalarının koşma sürelerine verimlilik açısından yakından bağlıdır. Bu yüzden yüksek performansa sahip hızlı çalışma süreleri tasarım açısından göz ardı edilemeyecek önemdedir.

Geçici hatalar lojik fonksiyonun nano dizinle gerçeklenip üretilmesinden sonra ortaya çıktığı için hataların etkileri incelenmiştir. Açık-durumda takılı kalmış ve kapalı-durumda takılı kalmış hataların devreye olan etkileri farklıdır. Açık-durumda takılı kalmış hatalar devrede bulunan girdiyi devre dışı bırakırken, kapalı-durumda takılı kalmış hatalar devreye yeni bir girdi eklemektedir.

Çalışmada kullanılan lojik fonksiyonlar minimum formda yazıldığı için açık-durumda takılı kalmış hataların telafisi mümkün değildir. Herhangi bir girdinin devre dışı bırakılması minimum formda işlem yapıldığı için fonksiyondan alınan çıktıyı değiştirir.

Kapalı-durumda takılı kalmış hataların bazıları fonksiyonun karakterine göre telafi edilebilir. Nano dizinle elde edilmiş lojik fonksiyona denk fonksiyonların bulunması, telafi edilebilir hataların yerini göstermektedir. Çalışmada sunulan metot verilen bir lojik fonksiyona denk fonksiyonların cebirsel işlemlerle bulunmasının içerir. Bu şekilde telafi edilebilen hatalar belirlenmiş ve güvenilirlik analizi yapılmıştır.

Deneysel sonuçlar kısmında sunulan algoritmanın diğer algoritmalarla karşılaştırması verilmiş ve çalışma süreleri incelenmiştir. Ayrıca verilen lojik fonksiyonun gerçeklenmesi için verilen nano dizinin boyutunun algoritmanın çalışma süresine etkileri gösterilmiştir. Lojik fonksiyonun boyutundan daha büyük nano dizinlerle gerçeklemenin çalışma süresinin önemli seviyede etkilediği görülmüştür.

Algoritmada sunulan sıralama yaklaşımının etkinliği yapılan benzetim sonuçlarıyla açıklanmıştır. Nano-dizin boyutunun algoritmanın çalışma süresi üzerindeki etkisi farklı boyutların göz önünde bulundurulmasıyla gösterilmiştir.

1. INTRODUCTION

1.1 Overview of Emerging Technology

Dominant approach towards integrated circuit production and computing is consisted of working with perfect components until fabrication costs became hard to ignore any longer. Most important issues scientist and engineers are facing concerning the prevalent CMOS based integrated circuit design as follows [4]:

1. Ultra thin Gate Oxides
2. Short Channel Effects
3. Doping fluctuation

Developments in nanotechnology started to produce successful computational elements [5] [6], however rate of defective elements are beyond the conventional standard of industry. For this reason focus of researches begun heading through working with defective structures.

After Hewlett-Packard Laboratories' experimental parallel computer Teramac [7] is shown to be a very efficient defect-tolerant computing paradigm, it was clear that even with the presence of large defect rates it is possible to obtain successful computational results.

In this thesis, crossbar based switching nano-arrays are used as general computing structures. Nano-arrays are produced with placing a group of nanowires aligned parallel to each other on another group of nanowires orthogonally. Crossbar nanotechnologies are favorably achieved by nanotubes or nanowires [8] [1] such that each crosspoint behaves as switching component. Diagrammatic representation of a nano-array is shown in Figure 1.1. According to the technology preference, every junction might show resistor, diode or FET like characteristics.

Nano-arrays used in this study have reconfigurability features. After the production, switches can be adjusted as desired. Reconfigurability yields the flexibility required for working with faulty structures.

As mentioned before, fault-free nano-array production is time consuming and expensive. For this reason, reutilization of imperfect nano-arrays is important. Common faults occurs on switches are shown in figure 1.2. Most common faults occur in described switches can be categorized under two main titles which are permanent and transient. Also, two categories have subtitles such as stuck-open, stuck-closed and nanowire break-downs. Because of the immense effect of nanowire break-downs, they are excluded from the body of study.

Permanent faults occur during production phase and are know beforehand which means they can be avoided while the mapping process of boolean function on the nano-array.

Transient faults occur after the production and mapping process, so their avoidance is not possible with reconfiguration. Effects of transient faults are closely related to the boolean function mapped on nano-array.

1.2 Purpose of Thesis

Main aim of the study presented here is to propose a fast heuristic algorithm to map a boolean function on a defective nano-array in case of stuck-open and stuck-closed

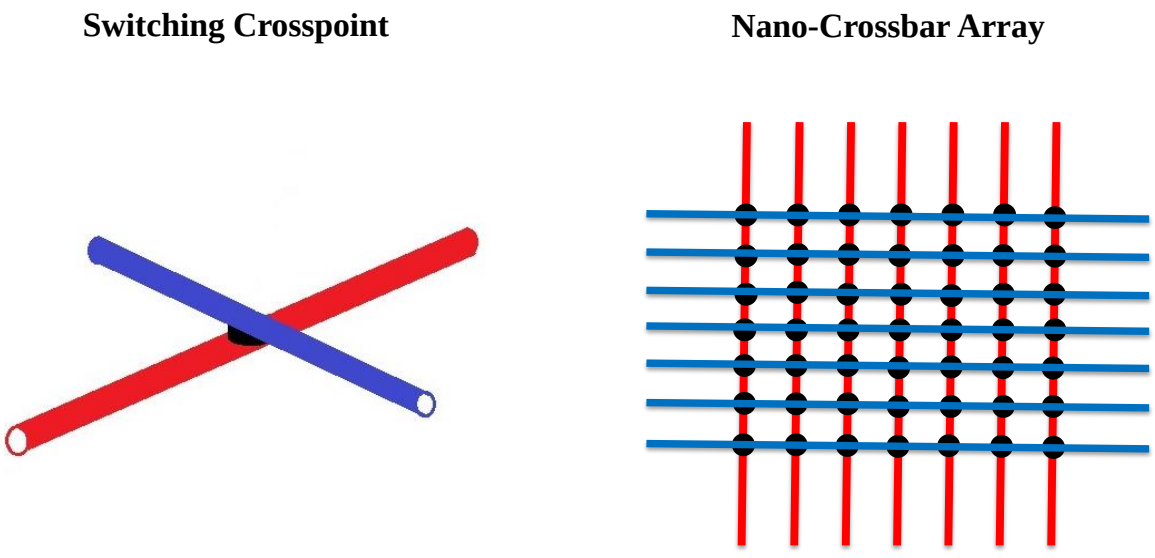


Figure 1.1 : Crossbar based switching array

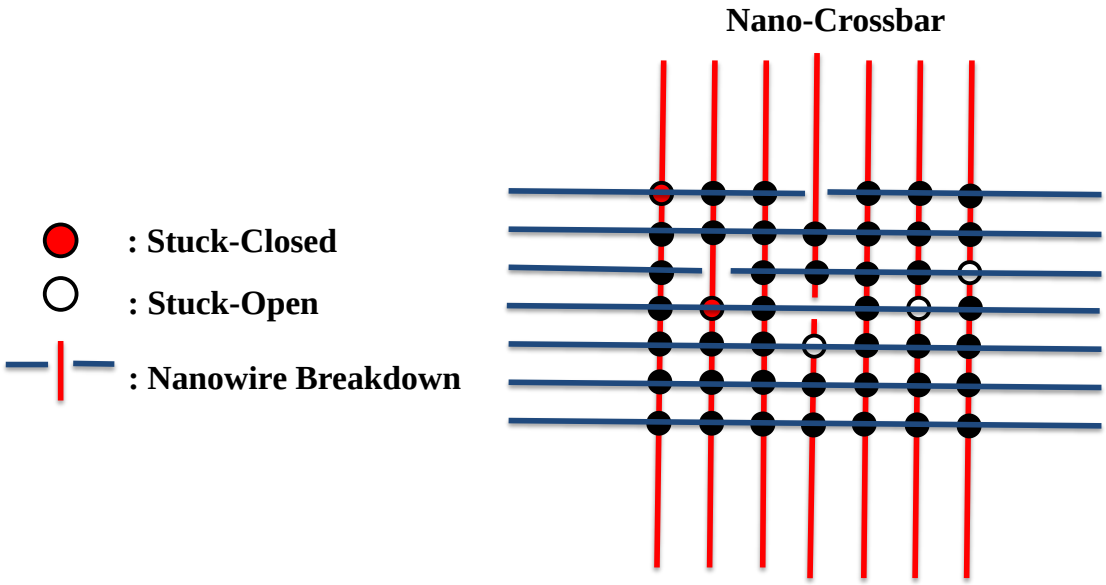


Figure 1.2 : Type of faults occur on array switches.

permanent faults and failure analysis of transient faults occur after the mapping process. As stated before, it is vital to overcome or benefit from faulty components exist in nano-arrays.

1.3 Literature Review

Mapping a target boolean function on a defective nano-array is an NP-complete problem [9]. In the worst-case scenario, an N x M logic function represented with a matrix has N!.M! permutations that is intractable for a reasonable computing time. Two main approaches to tackle this issue are defect-unaware finding k x k defect free sub-array in a N x N crossbar [10] [11] [12] and defect-aware mapping which uses graph based heuristics [13] [14] [15], Integer Linear Programming [16] [17] and memetic algorithm [18]. In graph based methods, an initial appointment is made for inputs to prune permutation space. However, in case of unfavorable appointment the number of reconfiguration increases drastically to find a valid mapping. In the proposed method, sorted matrices and index representations are used which helps to eliminate impossible appointment at the start.

Defect-unaware algorithms aim to find the largest possible $k \times k$ defect free sub-crossbar from a defective $N \times N$ crossbar where $k \leq N$ [10] [11] [12]. Detailed yield analysis of these algorithms shows a common shortcoming: the algorithms are

inefficient for high defect rates – k is much smaller than N [12]. When $N = 250$ and the defect rate is 15% that is a reasonable value for nano arrays, the fastest algorithm achieves k values as high as 30 [12]. It means that only 1% of the crossbar can be used. In this regard, defect-aware approach based algorithms perform much more satisfactorily [15] [19] [17]. A valid mapping is generally found using a 1.5 times larger crossbar than the optimal size crossbar to implement a target Boolean function. Note that for a specific target function, the larger the crossbar, the easier to find a valid mapping due to an increase in solution space. Therefore it is challenging, as well as desired for area considerations, to find a mapping with optimal size crossbars. Presented defect-aware heuristic algorithm satisfy this expectation.

Defect-aware algorithms which use graph based heuristics, transform the mapping problem into a graph isomorphism problem [13] [14] [15]. An initial input assignment is made to prune the permutation space. However, in case of an unfavourable assignment the number of reconfigurations needed to find a valid mapping increases drastically . Additionally, the runtime quickly grows beyond practical limits, especially for large-scale target functions. Other algorithms based on integer linear programming also suffer from runtime inefficiency for large-scale functions [16] [17]. Apart from the mentioned methods, a considerably fast memetic algorithm is proposed to tackle this problem. [18]. Here the drawback is that the starting conditions affect the results significantly. As an example, experimental results presented in [18] show as large as a 25 times difference in runtimes for the same size target functions. Proposed algorithm works considerably faster compared to the algorithms in the literature with nearly steady runtime values for the same size target functions and for large benchmarks such as “table5” and “t481”. Additionally, the proposed algorithm shows 99% accuracy in accordance with the results of an exhaustive search algorithm.

1.4 Key Concepts and Definitions

In this chapter, key concepts used throughout thesis are explained for both permanent defects and transient failures.

Definition 1: Consider k independent Boolean variables, $x_1, x_2, \ldots, x_k$. Boolean literals are Boolean variables and their complements, i.e., $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_k, \bar{x}_k$.

Definition 2: A product (P) is an AND of literals, e.g., $P = x_1\bar{x}_3x_4$. A set of a product (SP) is a set containing all the product's literals, e.g., if $P = x_1\bar{x}_3x_4$ then $SP = \{x_1, \bar{x}_3, x_4\}$. A sum-of-products (SOP) expression is an OR of products.

Definition 3: A prime implicant (PI) of a Boolean function f is a product that implies f such that removing any literal from the product results in a new product that does not imply f.

Definition 4: An irredundant sum-of-products (ISOP) expression is an SOP expression, where each product is a PI and no PI can be deleted without changing the Boolean function f represented by the expression.

Definition 5: Function matrix is a representation of a Boolean function in SOP form such that the function's literals and products are appointed to the matrix columns and rows, respectively. If a literal occurs in a product, it is denoted with 1; otherwise 0 is assigned. Figure 1.3 (a) shows an example of a function matrix.

Definition 6: Crossbar matrix is a representation of a crossbar array such that functional switches of crossbars are denoted with x; defective stuck-open and stuck-closed switches are denoted with 0 and 1, respectively. Figure 1.3 (b) shows examples of crossbar matrices by considering stuck-closed and stuck-open defects.

Definition 7: Logic inclusion ratio (IR) is defined as a ratio of the number of used switches to the total number of switches in a crossbar. As an example, consider the function matrix in Figure 1.3 (a). Here, the number of used switches is same as the number of 1's, so IR = 9/20.

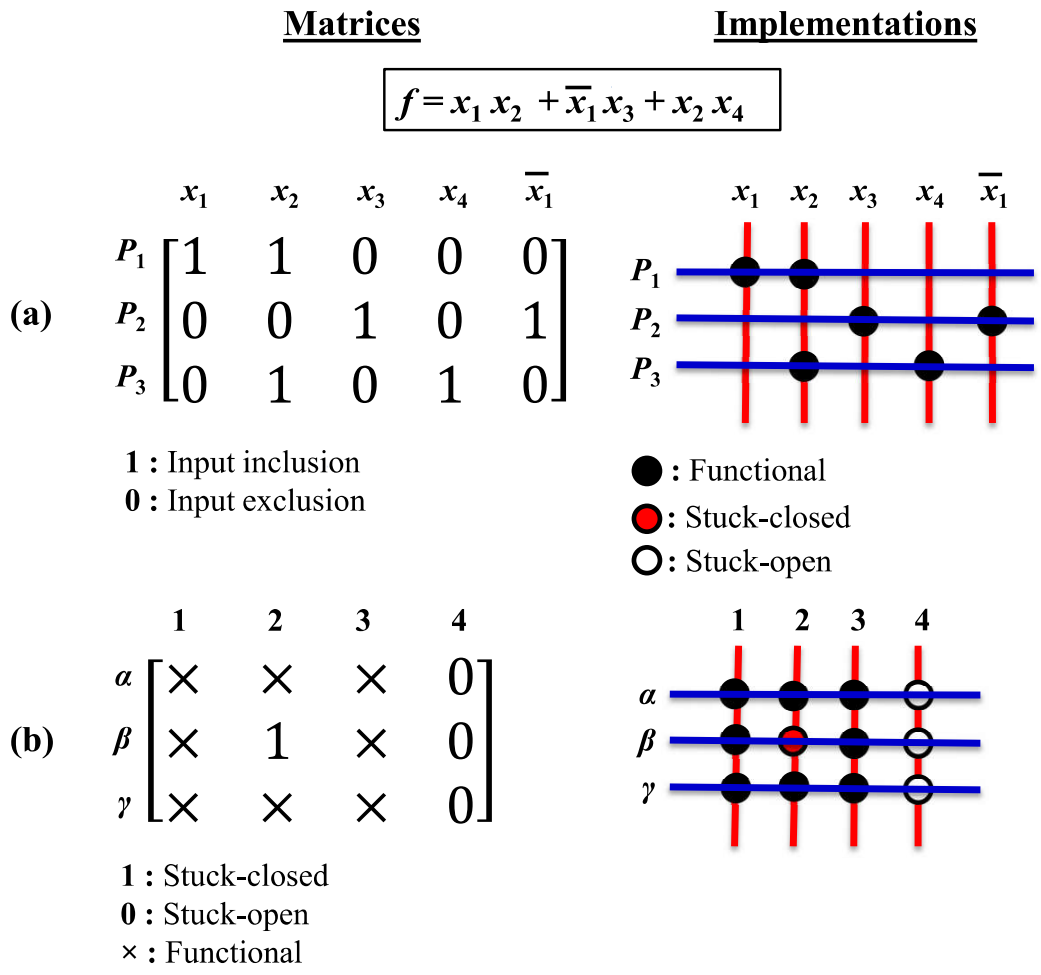


Figure 1.3 : Matrix representation of logic function (a) and defective crossbar (b)

2. COMPUTING WITH NANO-ARRAYS

Nano-arrays are similiar to the Programmable Logic Arrays in terms of logic mapping. There are AND and OR tiles for input and output lines for computation. At the early stage of crossbar structures, diode based technologies was common, however due to signal restoration need supplementary solutions was necessary.

In [20], coexisting CMOS and molecular electronics are proposed. Another paradigm proposed for crossbar based nano-arrays is Nanofabrics [1] which is composed of logic blocks as a compartmentalized tiles. Logic blocks are used for computation and signal routing.

Apart from all that, in 2011 [21], a working reconfigurable crossbar based nano-array is produced with fully functional switches. In addition, the same circuit is used as a full-subtractor, multiplexer, demultiplexer and clocked D-latch functions with the reconfiguration of switches.

It is clear that, nano-arrays can be used as a computing structures with the experimental results acquired in [21]. Yet another example is a nano-computer implemented as finite state machine in [6] which performs clocked multistage logic. It is demonstrated in the mentioned paper that, both sequential and arithmetic logic can be realized with nano-arrays which is also a promising answer towards integration issues voiced by researchers.

Different nano-array architectures are shown in Table 2.1 with their features as presented in [22] [23]. As can be seen from Table 2.1, every proposed architecture has prevailing characteristics. However, CMOS assistance is required in terms of signal restoration and other utilization for every architecture included as long as a FET element is not introduced for gain.

Detailed explanation of each architecture will be given in next section with comparison of crosspoint, nanofabrication, logic implementation, CMOS/Nanowire interface, restoration, nanodevice function and funtion of CMOS.

Table 2.1 : Nano-Array Architectures

Feature	**NanoFabrics**	**NanoPLA**	**CMOL**	**FPNI**
Crosspoint	Programmable diode	Programmabl diode	Programmable diode	Programmable diode
Nanofabrication	Nanopore template	Nanowire catalyst	Nanoimprint lithography	Nanoimprint lithography
Logic Implementation	Nanoscale wired-OR	Nanoscale wired-OR	Nanoscale wired-OR	Lithoscale (n)and2
CMOS/Nanowire Interface	-	Coded nanowire	Crossbar tilt	Crossbar tilt
Restoration	Resonant tunneling diode latch	Nanowire FET	CMOS	CMOS
Nanodevice Function	Circuit implementation, routing	NOR-NOR Logic	NOR logic, memory	routing, interconnect
Function of CMOS	Clock, Power, GND	Addressing, Routing	Inversion, demultiplexing, gain	Arbitrary Circuit

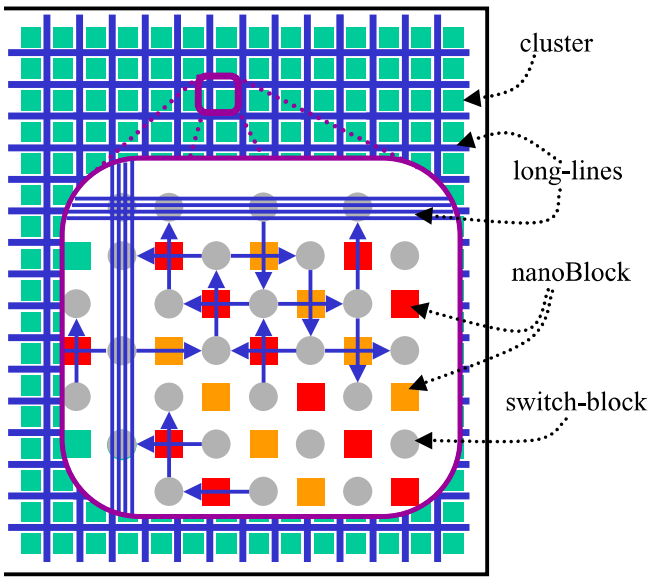


Figure 2.1 : Diagram of Nanofabrics [1]

2.1 Nano-array Based Architectures

Different architectures utilizing regular arrays of crossbars are demonstrated in this section. Nanowires [24] and nanotubes [25] are building blocks of these architectures. As a switching elements, programmable diodes and nFET-pFETs are implemented at the crosspoint of arrays. Producing logic gate from the mentioned structures are shown in [5].

Since computing with nano-arrays is an emerging paradigm, a single prevalent architecture is not present. Every design has its advantage and disadvantages. Following architectures are the promising candidates.

2.1.1 Nanofabrics

Nanofabrics consist of nanologic blocks that are connected with nanowires [1] and are produced with chemically assembled electronic nanotechnology. Crosspoints act as a programmable diode which implements a wired-OR logic.

Nanoblocks present in nanofabrics can be reconfigured in the post production and perform logic functions. Also, nanoblocks might act as routing device to be interface between different blocks. Schematic of nanoblocks are shown in Figure 2.1

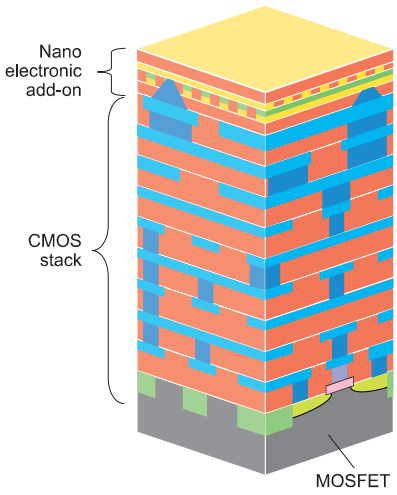


Figure 2.2 : Diagram of CMOL approach [2]

2.1.2 NanoPla

Nanosclae Programmable Logic Array also uses programmable diodes for logic implementation [26]. Unlike the Nanofabrics, NOR-NOr logic is used instead of wired-OR logic. For signal restoration naowire FETS are employed.

A particular decoder is used in order to produce a interface between CMOS and nano component. Nanowires are addressed through microwires.

2.1.3 CMOL

CMOL is combination of nano-arrays and CMOS technology [20]. Nano-arrays are placed on top of CMOS die in order to increase density of the whole circuit. CMOS is used as an inverter and gain mechanism. CMOL uses a NOR-logic and can be used a signal router or memory array. Crosspoints are programmable diodes. A generic diagram of CMOL approach is given in Figure 2.2.

Connection between top nano-array and down CMOS technology is practiced with different sized metal pins. The most challenging part of the CMOL is the production of metal pins an integration of two layer.

2.1.4 FPNI

Field-programmable Nanowire Interconnect is proposed to tackle issues on connection of nano-arrays and CMOS [3]. Diagram of FPNI is shown in Figure 2.3.

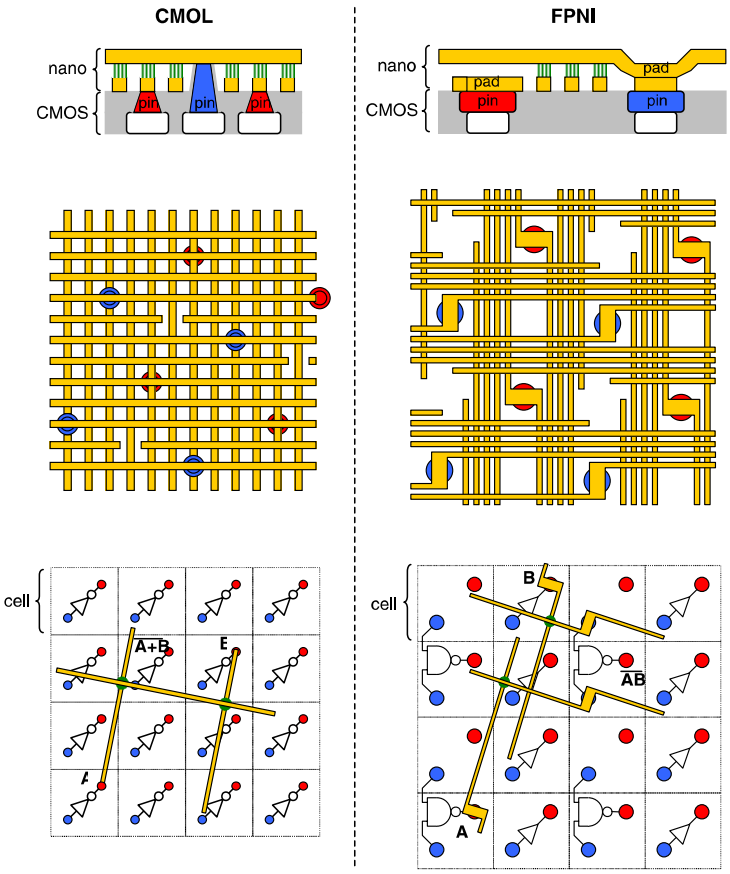


Figure 2.3 : CMOL and FPNI [3]

The previously mentioned CMOL (left side) installs a nanowire nano-array on top of CMOS circuit inverters. The nano-array is somewhat turned so that each nanowire is connected to a pin expanding up from the CMOS layer. CMOS is supplying gain and inversion.

The FPNI (right side) installs a sparser nano-array on top of CMOS gates and buffers. Nanowires are also turned so that each one connects to only one pin, but configured junctions (green, bottom panel) are used only for programmable interconnect, CMOS performs all the logic.

In FPNI, logical computation is performed with CMOS tile. However, it has lower density than CMOL approach due to the use of sparser nano-array. It can be said that FPNI is generic form of CMOL technology.

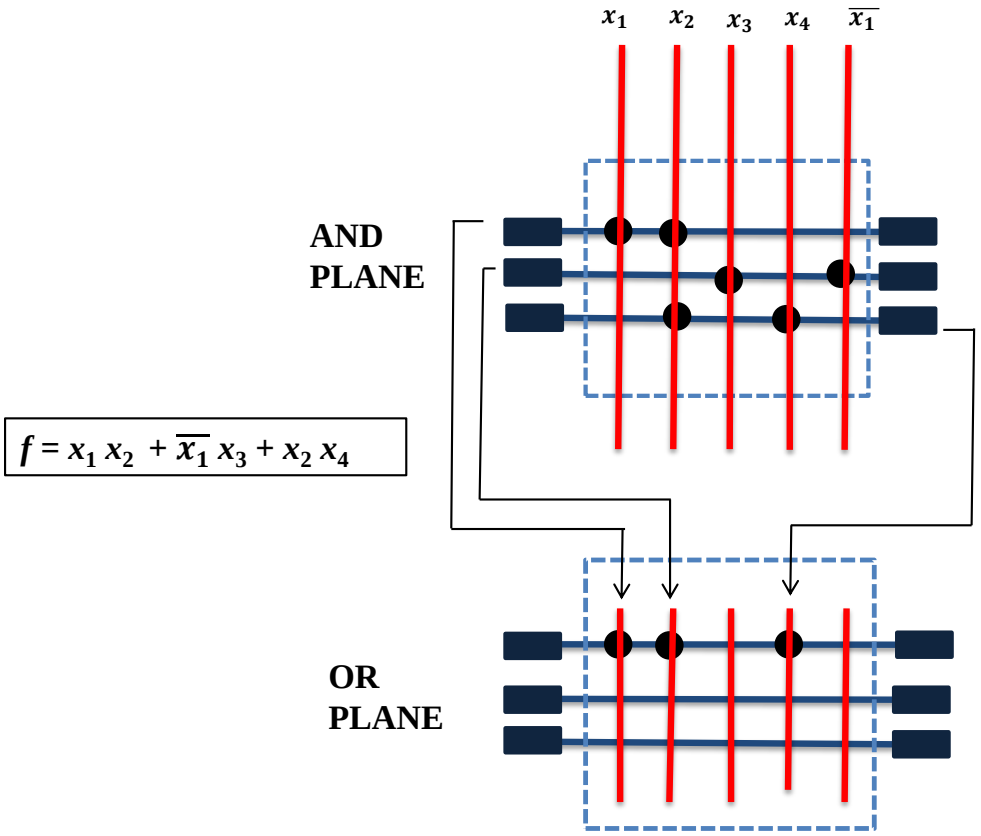


Figure 2.4 : Logic function mapping on a crossbar based switching array

2.2 Logic Implementation

In this study, logic computation method is approached with a technology independent view. Appointments of logic elements in the model of this study are shown in figure 2.4. Only mapping of AND tiles are considered as a problem since it is a common practice in literature.

Representation of logic function mapped on nano-array is a matrix model. Detailed explanation of matrix model and used key concepts are given in section 1.4.

It should be noted that, changing the appointment of input and product output lines does not alter the boolean function mapped on the nano-array. If the nano-array in question is defect-free, all implementations created equal, however as mentioned earlier in case of defective switches present in the array, defective switches should be avoided or benefited in order to obtain a valid mapping of target function on the defective nano-array. In Figure 2.5, swapping row and column of function matrix is shown. Since functional switches shown with $\times$ in crossbar matrix can be matched

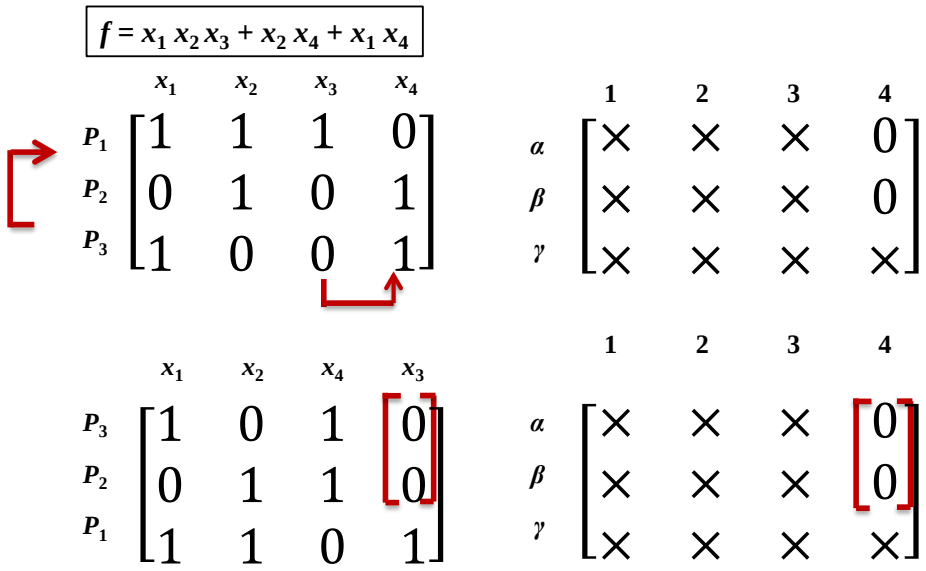


Figure 2.5 : Logic function manipulation to find a valid mapping

with either switch in logic matrix, considering only defective switches is sufficient to find a valid mapping.

Main distinction between conventional and emerging nano-array based computing lies in the defective nature of the latter. Defect rates up to 15% is inside the margin of projected results [27] considering the upcoming emerging nanoelectronic devices.

Random nature of defects gives nano-arrays used as a computing structure in this thesis a unique character which is why they are called snowflakes in [23]. For this reason, every nano-array needs to undergo individual tuning. In order to overcome financial and time related issues, new electronic design tools and fast algorithms will be in great demand.

3. PERMANENT FAULT TOLERANCE

In this section, key concepts used in permanent defects and mapping algorithm are explained. Aim is to find out whether it is possible or not to map a target function on a given defective crossbar in the presence of permanent faults. Before the explanation of the concepts, it should be noted that index approach is used in a distinct way for the algorithm. Index means number of same matrix elements for a chosen value in a row or column.

Algorithm fundamentally uses index representations of function and crossbar matrices as well as row/column permutations and matchings. These concepts are explained as follows.

3.1 Preliminaries

Row Index: The number of the same 0 or 1 valued elements in a matrix row. For example, the row represented by P_1 in Figure 3.1 has a row index of 3 for a chosen value of 1.

Column Index: The number of the same 0 or 1 valued elements in a matrix column. For example, the column represented by x_1 in Figure 3.1 has a column index of 1 for a chosen value of 0.

Row Index Set: A set of all row indices of a matrix for a chosen value of 0 or 1. In Figure 3.1, rows represented by P_1, P_2, and P_3 have row indices of 1, 2, and 2, respectively, for a chosen value of 0. So its set of row indices is $I_{R,F} = \{1,2,2\}$ where R stands for *row* and F stands for *function*.

Column Index Set: A set of all column indices of a matrix for a chosen value of 0 or 1. In Figure 3.1, columns represented by x_1, x_2, x_3, and x_4 have column indices of 2, 2, 1, and 2, respectively, for a chosen value of 1. So its set of column indices is $I_{C,F} = \{2,2,1,2\}$ where C stands for *column* and F stands for *function*.

Row/Column Permutation: In order to find a valid mapping, defective switches of a crossbar matrix which are denoted as 0 (stuck-open) or 1 (stuck-closed) must be

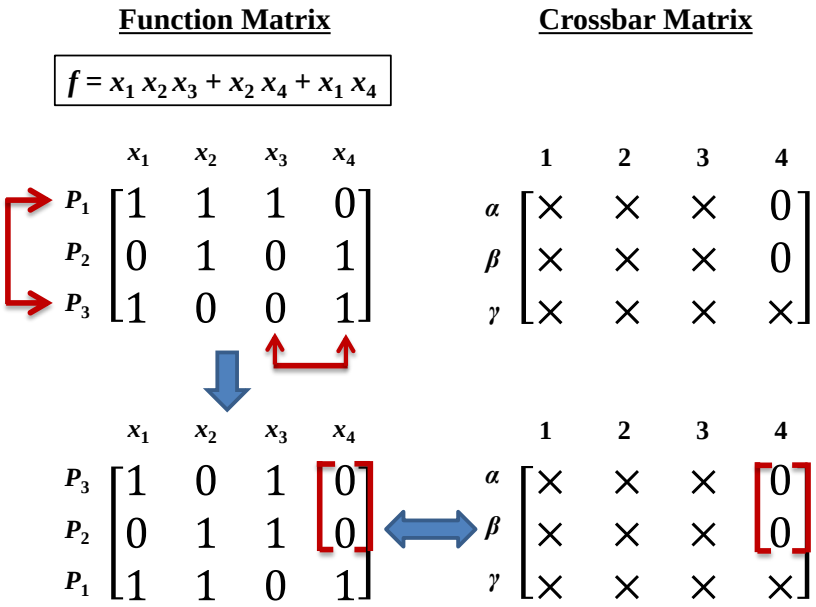


Figure 3.1 : Row and column permutations of the function matrix to obtain a valid mapping.

matched with 0s (unused) and 1s (used) in a function matrix. Here, an important property is that row and column permutations in the function matrix do not alter the implemented function.

This is an important reconfigurability feature for fault tolerance as illustrated in Figure 3.1.

Row/Column Matching with Multiplication: In order to match two rows from function and crossbar matrices, element-by-element multiplication is used. Functional switches in the crossbar matrix can be matched with either 1s or 0s in the function matrix. By representing functional switches with 0s and defective switches with 1s, matching with element-by-element multiplying of the rows can be achieved. If the obtained row is same as the crossbar row then there is a matching, all defective switches tolerated; otherwise there is no matching. Figure 3.2 illustrates an example for a valid matching between the first rows of the matrices.

3.2 Previous Approach

In the previous work [28] of the author, two heuristics are employed to find a valid matching between logic and crossbar matrix: element-by-element matrix multiplication and double index set comparison.

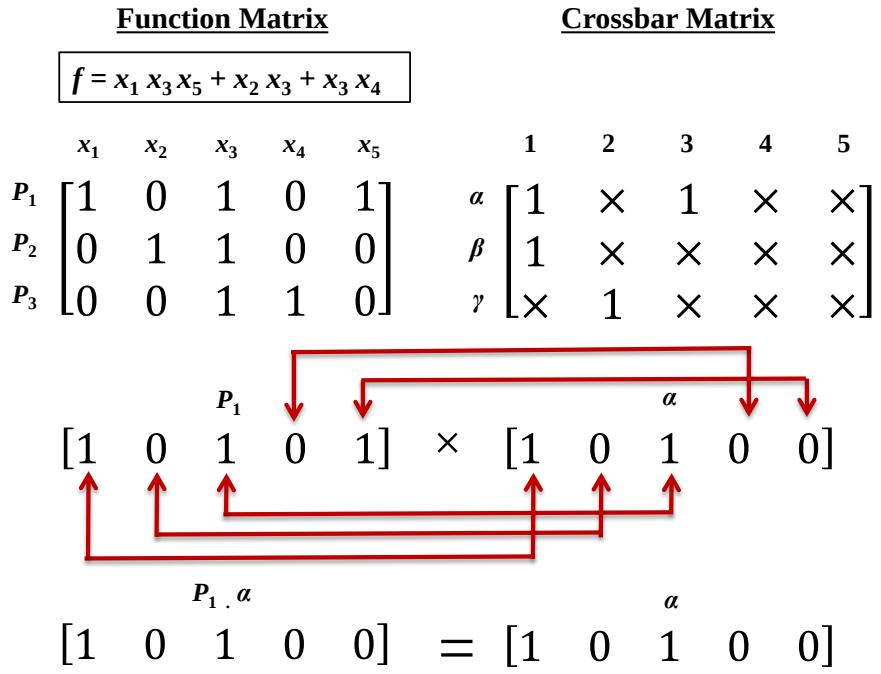


Figure 3.2 : Element-by-element multiplication of the rows represented by P_1 and α; there is a matching.

First method is similar to matching with multiplication and only difference is using multi-dimensional arrays instead of rows. Second method produces an invariant for a matrix which is constant for all the permutation of the matrix. By comparing the two double index set, it is possible to conclude if there is a matching. In short, previous approach adopts multi-dimensional arrays as a core element of matching problem.

In this study [29], one-dimensional arrays are used, rows of a matrix, as matching elements. The reason for preference of this method over the previous one is that for large-scale target functions although multi-dimensionality reduces the computational load and provides more information, probability of mismatches increases.

During extensive experimental study for different benchmark circuits with varying sizes, it is found that multi-dimensional approach require more reconfiguration of given target function than one-dimensional backtracking process. Since the reconfiguration affects more elements due to the multi-dimensional arrays, the advantages diminish compared with the increase in the probability of mismatch occurrence.

3.3 Proposed Algorithm

Permanent defects are determined before mapping the logic function on a crossbar so the positions of defective switches are known beforehand which is expressed as a

Input:	Function and crossbar (defective) matrices
Output:	"YES" **i**f the matrices are matched; "NO" otherwise
Step 1	**Sorting**: Sort function and crossbar matrices according to the row and column index sets.
Step 2	**Matching**: Starting from the top row in the function matrix, perform matching with multiplication by advancing search from the top row to the bottom row of the crossbar matrix. If all of the function rows are matched then return "YES".
Step 3	**Backtracking**: If no matching is found for a function row then search previously matched crossbar rows from top to bottom. If a matching is found then repeat Step 2 by excluding the already matched rows.
Step 4	**Repeating**: If no matching is found then repeat Step 2 (and Step 3) for 3000 times by randomly applying a pairwise crossbar column permutation. If a matching cannot be found under 3000 trials, then return "NO".

Figure 3.3 : Outline of the proposed algorithm.

defect map in the similar works in literature. Since defects are known in advance, it is possible to manipulate logic function according to the crossbar.

The verbal outline of algorithm is shown in Figure 3.3. Detailed explanation of steps will be given.

It should be noted that the algorithm chooses the row or column with the minimum size as a constant part of the search process. In explanation of the algorithm steps, an example with column permutation staying the same is demonstrated. If the size of the row would be smaller, row permutation would stay the same. If the logic function has N x M size, determination of which dimension would take the constant permutation is chosen with the result of min{N,M}.

The purpose of choosing minimum dimension is decreasing the probability of mismatches. In section 5, the aspect ratio relation of algorithm runtime will be presented in detail.

In addition, since the algorithm is symmetric, it is not important whether search is advancing through rows or columns provided that smaller dimension stays the same.

1. Step: *Sort function and crossbar matrices according to the row and column index sets.*

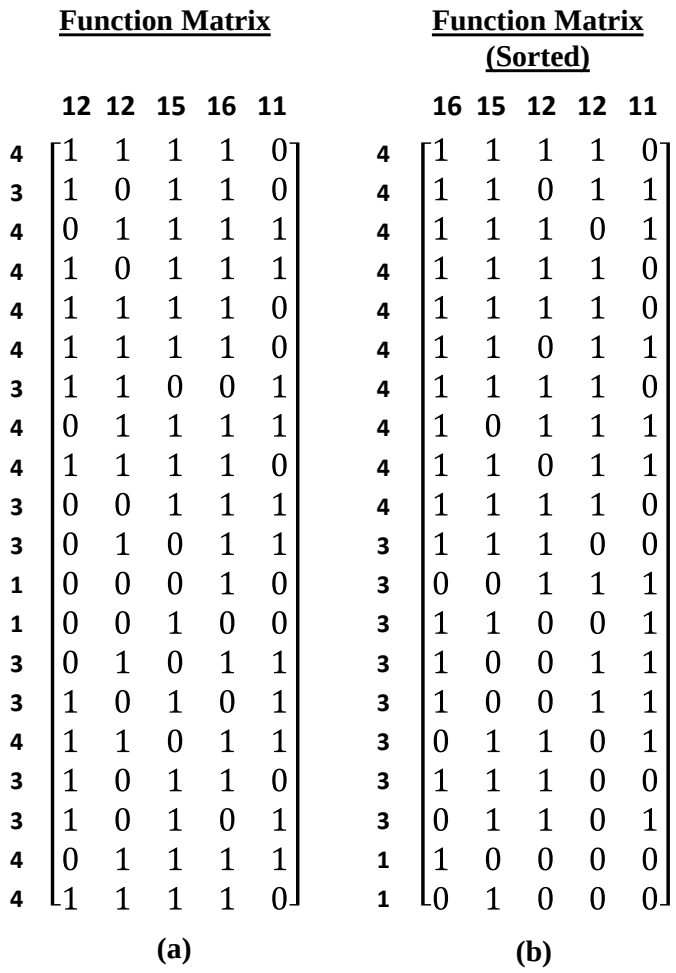


Figure 3.4 : In the presence of stuck-closed defects, (a) function matrix and (b) its sorted form.

Firstly, row and column index sets for logic and crossbar matrix are found. The most defective row and columns are determined with the information provided by them. After that, rows and columns are aligned in order to improve the probability of a valid matching.

It is shown in [14] [30] that, beginning with a constant permutation for one dimension and advancing through another reduces the number of operations for finding a valid mapping. The advantage of method is working with sorted matrices which decreases the possibility of an unfavorable initial appointment. Sorting process is shown in Figure 3.4.

2. Step: *Starting from the top row in the function matrix, perform matching with multiplication by advancing search from the top row to the bottom row of the crossbar matrix. If all of the function rows are matched then return "YES".*

Since function matrix and crossbar matrix is sorted considering the rows and columns which have the most matchable elements and the rows and columns which have the most defective elements respectively, probability of finding a valid matching increases drastically.

3. Step: *If no matching is found for a function row then search previously matched crossbar rows from top to bottom. If a matching is found then repeat Step 2 by excluding the already matched rows.* (Backtracking).

In Figure 3.5, every row until P_{14} is matched with a row in crossbar matrix. However, P_{14} cannot be matched with any of the unmatched rows (denoted with 0s), so previously matched rows should be searched again. The matching for P_{14} is the 4th row of crossbar matrix in Figure 3.5. So previously the 2nd row of function matrix was matched to the this row and it should be included in search again.

Distinction of backtracking process is that, the 2nd row which is added to the search is checked with the unmatched rows of crossbar matrix. This method prevents the proposed algorithm to obtain a recursive character which expands the computational load. If algorithm would check all the rows for 2nd row, a matching might be found with an already matched row and it should be included in search again and the recursion might cumulate drastically.

In case backtracking is not able to find a valid matching with unmatched rows of crossbar matrix, column permutation is changed and search begins again which is the 4. step of the algorithm.

4. Step: *If no matching is found then repeat Step 2 (and Step 3) for 3000 times by randomly applying a pairwise crossbar column permutation. If a matching cannot be found under 3000 trials, then return "NO".*

As mentioned before, sorted matrices are used to find a valid matching. Nevertheless, in some cases no matching can be found even though backtracking search. When such a case occurs, column permutation is changed in order to find a valid mapping. Although, for most cases column permutation is not necessary. The number of column permutations for example benchmark circuits in the section 5 will be given.

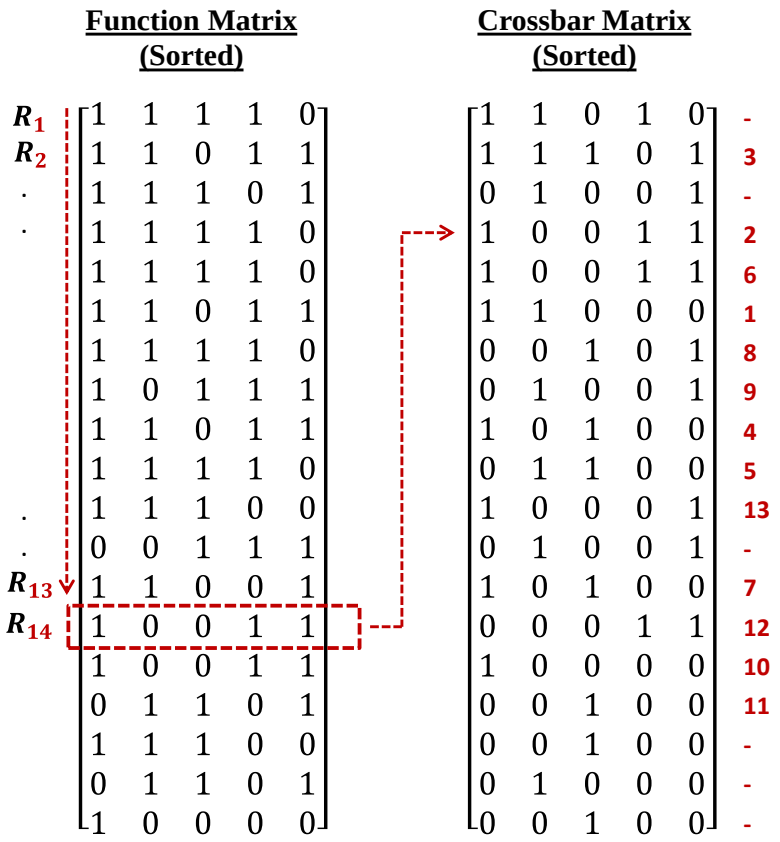


Figure 3.5 : 0s show unmatched rows and the numbers show which row from the function matrix is matched with the corresponding crossbar matrix row. P_{14} cannot be matched with any of the unmatched rows.

Another point needs to be addressed is the reason behind the choice of 3000 trial number.In order to maintain 95% succes rate it is essential to consider different permutation due to the size of solution space.

A pseudo code of the proposed heuristic algorithm is depicted in Figure 3.6 below. Parameters *row_pattern* and *column_pattern* indicate row and column permutations of a function matrix, respectively. Establishing correct row and column patterns yields a valid mapping of a target function into a defective crossbar.

3.4 Performance Evaluation

The algorithm uses a constant permutation for one dimension (column) and advancing through the other one (row) that reduces the number of operations for finding a valid mapping [14] [30].

Instead of using conventional two dimensional matchings of matrices, the algorithm performs considerably faster one dimensional matrix row matchings. Motivation is that

the main problem of mapping target functions has many different solutions. Therefore probable information lost in one dimensional check can be easily compensated; backtracking and repeating is also for this purpose.

An important factor is the relation between logic inclusion ratio (IR) and fault rate. For a constant IR between 30% and 40%, a typical range for standard benchmark functions, the number of mapping solutions, so the performance of the algorithm, dramatically decreases with an increase in the fault rate especially beyond 25%.

For fault rates below 20%, the algorithm works satisfactorily in terms of both run time and accuracy with surpassing related algorithms in the literature. The algorithm's performance is also justified with a complexity analysis as follows.

Considering a function/crossbar matrix with a size of $N \times M$ where $N \geq M$. The number of initial operations for every row checking is M for multiplication plus M for comparison, so in total of $2M$. Additionally, each function row is matched with N crossbar rows, so $2M \cdot N$ operations are needed. In case of backtracking, another N rows need to be checked that results in $2M \cdot [N+N]$ operations. For all of the function rows, there are $N \cdot [2M \cdot [N+N]]$ operations. In the worst-case scenario, 3000 trials are executed so the number of operations become $3000N \cdot [2 \cdot M \cdot [N+N]]$. As a result the algorithm works in $O(M \cdot N^2)$ time in the worst-case scenario.

Algorithm Heuristic Algorithm

```
1: Input: Function Matrix (FM) and Crossbar Matrix (CM)
   Size of NxM
2: Output: row_pattern and column_pattern
3:
4: function INDEX_SORT(M)
5:     Sort I_{R,M} descending
6:     Sort I_{C,M} descending
7:     row_pattern ← I_{R,M}
8:     column_pattern ← I_{C,M}
9:     return M
10: end function
11:
12: INDEX_SORT(FM)
13: INDEX_SORT(CM)
14: matched_rows = 0;
15: for t in 3000 do
16:     if t > 1 then
17:         change column_pattern
18:     end if
19:     for k in N do
20:         F_k ← kth row of FM
21:         for j in N and not in matched_rows do
22:             C_j ← jth row of CM
23:             if F_k .* C_j = C_j then
24:                 matched_rows(k) = j
25:                 break
26:             end if
27:         end for
28:         if no matching then          ▷ Backtracking process
29:             for j in matched_rows do
30:                 C_j ← jth row of CM
31:                 if F_k .* C_j = C_j then
32:                     matched_rows(k) = j
33:                     break
34:                 end if
35:             end for
36:         end if
37:         if matching found then ▷ matched_rows changed
38:             F_m ← previously matched row of FM
39:             for j in N and not in matched_rows do
40:                 C_j ← jth row of CM
41:                 if F_m .* C_j = C_j then
42:                     matched_rows(m) = j     ▷ Rematching
   process
43:                     break
44:                 end if
45:             end for
46:         end if
47:         if no matching found for F_m then
48:             break                ▷ column_pattern changes
49:         end if
50:     end for
51: end for
```

Figure 3.6 : Pseudocode of proposed algorithm

4. TRANSIENT FAULT TOLERANCE

Transient faults occur after the production and mapping of nano-arrays. Related to the time domain, their tolerance can not be achieved by applying the same technique used for permanent faults that is based on fault identification followed by reconfiguration. Transient fault tolerance is purely based on redundancy. For nano-crossbar arrays, redundancy is correlated with the logic inclusion ratio (IR) as well as the used sum-of-product representations of target functions.

Similar to permanent faults, stuck-open and stuck-closed transient faults are considered. It is supposed that target functions are implemented in irredundant sum-of-products (ISOP) forms to minimize the number of used switches for cost optimization in fabrication. It is also supposed that target functions are implemented using optimal size nano-crossbars. Using these assumptions, analyse fault tolerance performance of nano-crossbar arrays by considering the specifics of target functions are analyzed. Figure 4.1 shows an example. A given target function f in ISOP form is implemented with an optimal size fault free crossbar shown in Figure 4.1 (a). When a stuck-open fault occurs on a used switch (denoted with 1s) as shown in Figure 4.1 (b), the corresponding literal is erased from the target function and the corresponding matrix element becomes 0. In this example, since the new function f' is not equal to the original function f, the fault cannot be tolerated. When a stuck-closed fault occurs on an unused switch (denoted with 0s) as shown in Figure 4.1 (c), the corresponding literal is added to the target function and the corresponding matrix element becomes 1. Here, the new function f'' is equal to f, so the fault is tolerated.

4.1 Stuck-Open Faults

Stuck-open faults are tolerated iff they occur on unused switches. Faults on used switches change the implemented functions; since ISOP forms of target functions consisting of prime implicants is used, removing any literal from them results in a new function. Fault tolerance performance FT_{so} of an $N \times M$ crossbar can be directly

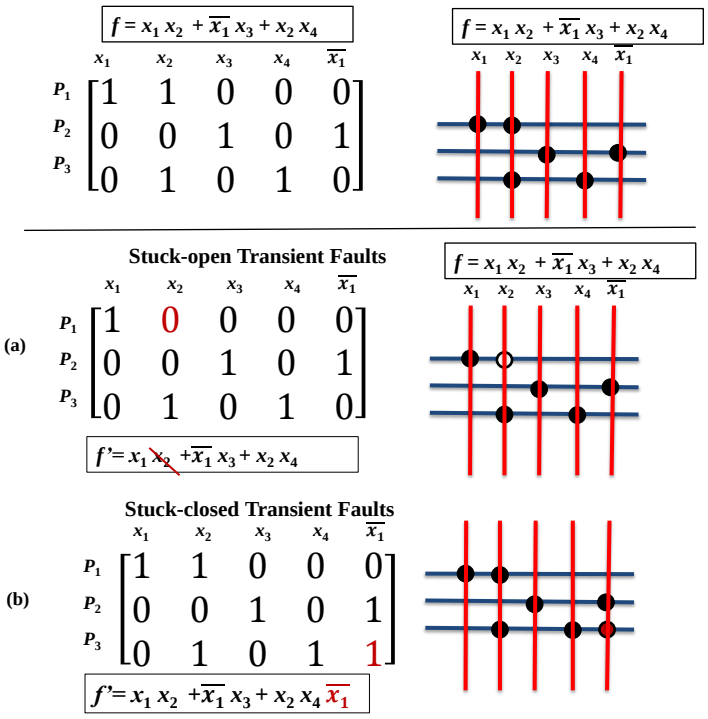


Figure 4.1 : Implementations in the presence of (a) no faults (b) stuck-open faults, and (c) stuck-closed faults.

calculated by using

$$FT_{so} = (1 - p_{so})^{N \cdot M \cdot IR} \tag{4.1}$$

where p_{so} is an independent stuck-open fault probability of each switch and IR is the logic inclusion ratio.

4.2 Stuck-Closed Faults

It is shown that along with all stuck-closed faults occurring on used switches, faults on unused switches can also be tolerated. This is illustrated in Figure 4.2 with a brief summary of our tolerance analysis method. All possible positions of tolerable faults on unused switches in the crossbar are determined. These positions, represented by added 1s in red in Figure 4.2, are determined by decreasing the number of rows that the faults are seen. First, tolerable fault positions in single rows (products) are determined. For the example in Figure 4.2 among 5 rows, representing 5 products of the target function, 3 of them have the positions. Therefore there are 3 matrices showing tolerable fault positions. Analyzing the first matrix at the upper-left corner, it is concluded that a stuck-closed fault in the first row at the right end of the crossbar can be tolerated; $f_{t_1} = x_1 x_2 \overline{x_3} + x_2 x_3 + \overline{x_3}\, x_4 + x_4 x_5 + \overline{x_1}\, x_5\, \overline{x_2} = f$. The same is valid for second and third

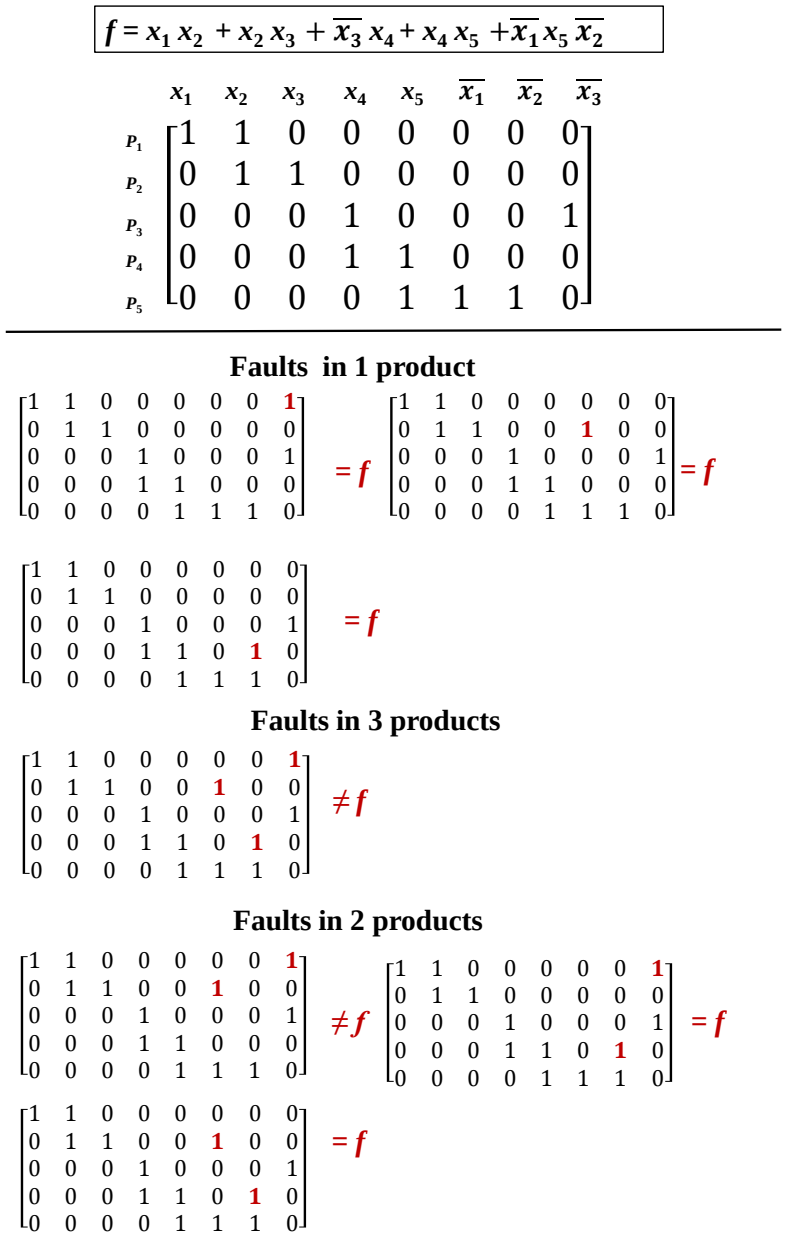


Figure 4.2 : Tolerable faults and faults cannot be tolerated

matrix as well. Next, it is determined tolerable fault positions seen in three product. There is no solution for faults in three products, so proceeding to the next step and checking faults in two products are exercised. In order to find all possible positions of tolerable faults, logic equivalences of Boolean expressions are exploited. Consider a given target function $f = P_1 + ... + P_m$ in ISOP form. Stuck-closed faults on unused switches add literals to the corresponding products that results in a new function. If this function is equal to f, tolerance is achieved. Our main purpose is finding all f_t's, corresponding tolerable fault positions in crossbars, that is equal to our beginning

function f. Two examples of f_t's from Figure 4.2 are $f_{t_1} = x_1x_2\ \overline{x_3} + x_1x_2 + \overline{x_3}\ x_4 + x_4x_5 + \overline{x_1}\ x_5\ \overline{x_2}$ and $f_{t_2} = x_1x_2 + x_1x_2\ \overline{x_1} + \overline{x_3}\ x_4 + x_4x_5 + \overline{x_1}\ x_5\ \overline{x_2}$.

Added products of literals, shown in red, are named as P_{t_i}'s; i represents the corresponding product number. As an example, f_{t_1} has $P_{t_3} = \overline{x_3}$; f_{t_2} has $P_{t_2} = \overline{x_1}$. A general form of f_t's can be represented as

$$f_{t_{\{i,..,k\}}} = P_1 + ... + P_iP_{t_i} + ... + P_kP_{t_k} + ... + P_m \tag{4.2}$$

where the subscript of f, $\{i,..,k\}$ set shows which products have added literals corresponding to faults.

Our method of finding all $f_{t_{\{i,..,k\}}}$'s that equal to f proceeds as follows. It is started with finding $f_{t_{\{i\}}}$'s, $1 \leq i \leq m$ only one product of f changed. Assuming found $P_{t_i}, ..., P_{t_t}$, it is checked $f_{t_{\{i,..,k\}}}$ (faulty logic function containing all fault products) for equivalence. If two function is equal to each other, the rest of fault products is not necessary to be checked. If two function is not equal to each other, then advancement is carried out through decrementing the number of fault products and equivalence is checked again.

Following theorem proves why it is not necessary to check all faulty logic functions if $f_{t_{\{i,..,k\}}} = f$.

Theorem 1: If $f_{t_{\{i,..,k\}}} = f$, then for $\forall x \subset \{i,..,k\} f_{t_x} = f$.

Given theorem provides that when $f_{t_{\{i,..,k\}}} = f$, then all combination of $f_{t_i}s$ also satisfy our tolerance condition and it is not necessary to check the rest of the faulty logic functions. If only fault products ($P_{t_i}s$) can be found that satisfy $f_{t_i} = f$ condition, it can determined that the rest of faulty logic functions.

In order to Determining Fault Products ($P_{t_i}s$) Found in $f_{t_{\{i\}}}s$ firstly, the values which makes $f \neq f_{t_{\{i\}}}$ are established. It can be seen from Table 4.1, when $P_i = 1$ and $P_{t_i} = 0$ equivalence is not determined. Considering this case, if P_{t_i} is chosen according to the following necessary and sufficient condition, equivalence is satisfied.

Theorem 2: P_{t_i} is the negation of single literal products or AND of them found in $f(P_i = 1)_{t_{\{i\}}}$ if and only if $f = f_{t_{\{i\}}}$.

If P_{t_i}'s are chosen according to given theorem, equivalence is satisfied. Proofs of the theorems will be given in Section 4.4.

Table 4.1 : Equivalence of f and $f_{t_{\{i\}}}$

P_i-P_{t_i}	f	$f_{t_{\{i\}}}$	**Equivalence**
1 - 1	$f = ... + 1 + ...$	$f_{t_{\{i\}}} = ... + 1.1 + ...$	$f = f_{t_{\{i\}}}$
0 - 1	$f = ... + 0 + ...$	$f_{t_{\{i\}}} = ... + 0.1 + ...$	$f = f_{t_{\{i\}}}$
0 - 0	$f = ... + 0 + ...$	$f_{t_{\{i\}}} = ... + 0.0 + ...$	$f = f_{t_{\{i\}}}$
1 - 0	$f = ... + 1 + ...$	$f_{t_{\{i\}}} = ... + 1.0 + ...$	**undetermined**

An example of given method is as follows:

Given an $f = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + \overline{x_1}x_4 + \overline{x_2}.\overline{x_4} + \overline{x_3}x_5 + x_6\overline{x_5}$, literal set of f, LS = { $x_1, x_2, x_3, x_4, x_5, x_6, \overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \overline{x_5}$.}.

1.step: Faults only occur in one product are found.

When P_1 = 1, evaluation of the function as follows, $f(P_1 = 1)_{t_{\{1\}}} = x_4 + x_6\overline{x_5}$. Negation of single literal products must be a member of LS, so the value of $P_{t_1} = \overline{x_4}$.

When $P_2 = 1$, evaluation of the function as follows, $f(P_2 = 1)_{t_{\{2\}}} = \overline{x_4} + x_6\overline{x_5}$, so the value of $P_{t_2} = x_4$

when $P_3 = 1$, evaluation of the function as follows $f(P_3 = 1)_{t_{\{3\}}} = x_5 + x_6\overline{x_5}$, so the value of $P_{t_3} = \overline{x_5}$

when $P_4 = 1$, evaluation of the function as follows, $f(P_4 = 1)_{t_{\{4\}}} = x_2x_3 + \overline{x_3}x_5 + x_6\overline{x_5}$, so P_{t_4} cannot have any value due to the no single literals.

When $P_5 = 1$, evaluation of the function as follows, $f(P_5 = 1)_{t_{\{5\}}} = x_1x_3 + \overline{x_3}x_5 + x_6\overline{x_5}$, so P_{t_5} cannot have any value due to the no single literals.

when $P_6 = 1$, evaluation of the function as follows, $f(P_6 = 1)_{t_{\{6\}}} = x_1x_2 + \overline{x_1}x_4 + \overline{x_2}\,\overline{x_4}$, so P_{t_6} cannot have any value due to the no single literals.

When $P_7 = 1$, evaluation of the function as follows, $f(P_7 = 1)_{t_{\{7\}}} = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + \overline{x_1}x_4 + \overline{x_2}\,\overline{x_4}$, so P_{t_7} cannot have any value due to the no single literals

2.Step: $f_{t_{\{1,2,3\}}}$ will be checked which has fault products P_{t_1}, P_{t_2} and P_{t_3}. As mentioned before, if $f = f_{t_{\{1,2,3\}}}$ then all combination of $P_{t_i}s$ also satisfy equivalence. For $f = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + \overline{x_1}x_4 + \overline{x_2}.\overline{x_4} + \overline{x_3}x_5 + x_6\overline{x_5}$ and $f_{t_{\{1,2,3\}}} = \overline{x_1}x_2x_3\overline{x_4} + x_1\overline{x_2}x_3x_4 + x_1x_2\overline{x_3}.\overline{x_5} + \overline{x_1}x_4 + \overline{x_2}.\overline{x_4} + \overline{x_3}x_5 + x_6\overline{x_5}$, f and $f_{t_{\{1,2,3\}}}$ is equal to each other, so every combination of $P_{t_i}s$ as follows also equal to f.

$f_{t_{\{1,2,3\}}} = f_{t_{\{1,2\}}} = f_{t_{\{1,3\}}} = f_{t_{\{2,3\}}} = f_{t_{\{1\}}} = f_{t_{\{2\}}} = f_{t_{\{3\}}} = f$ show the positions of tolerable stuck-closed faults.

As a general rule fault tolerance with a p fault rate can be stated as follows:

$$FT_{sc} = \binom{n}{1}(1-p)^{Z-LS_1} \cdot p^{LS_1} + \ldots + \binom{n}{n}(1-p)^{Z-LS_n} \cdot p^{LS_n} \tag{4.3}$$

n is number of fault products occur in one product, Z is number of zeros found in function matrix and LS_i is sum of literal number found in fault products

4.3 Failure Analysis of Benchmark Functions

In this section, results for transient faults tolerance performance of benchmark circuits are presented. Table 4.2 shows the results of benchmark functions with respect to fault rates and types. Performance values of benchmark functions are found with given formulas in section 4. For stuck-open faults since it is not possible to tolerate any fault occurs on a used switch, performance is directly related to number of switches used in function. For stuck-closed faults, tolerable functions are obtained with the method in section 4.

Stuck-open faults yield better results than stuck-closed faults. Reason behind that, logic inclusion ratio of benchmark function is generally less than 50% which means more possible position for stuck-closed faults. Also, since the number of tolerable stuck-closed cases found with our theorem are not high enough to balance logic inclusion ratio in favor of performance.

Table 4.2 : Performance of Benchmark Functions for Transient Faults with 5% Fault Rate

Circuit Name	**Stuck-open**	**Stuck-closed**	
		Expected Perf.	**Actual Perf.**
B12 1	23%	16%	21%
B12 6	19%	14%	16%
B12 7	19%	14%	19%
C17 0	73%	73%	77%
Dc1 2	54%	44%	53%
Dc1 6	73%	63%	66%
Misex1 7	48%	32%	35%

4.4 Theorem Proofs

In this section the mentioned theorems used in finding tolerable equivalent logic functions will be proven.

*Theorem 1:*If $f_{t_{\{i,..,k\}}} = f$, then for $\forall x \subset \{i,..,k\} f_{t_x} = f$.

*Proof :*Without loss of generality, lets prove this for $f_{t_{\{i,..k,l\}}}$ and $f_{t_{\{i,..,k\}}}$

$f = P_1 + \ldots + P_m$

$f_{t_{\{i,..,k\}}} = ... + P_i P_{t_i} + \ldots + P_k P_{t_k} + ... + P_l + ... + P_m$

$f_{t_{\{i,..k,l\}}} = ... + P_i P_{t_i} + \ldots + P_k P_{t_k} + ... + P_l P_{t_l} + ... + P_m$

Assuming when $f_{t_{\{i,..k,l\}}} = f$ and $f_{t_{\{i,..,k\}}} \neq f$.

If $f_{t_{\{i,..k,l\}}} = f$ for $P_i = 1$ evaluating the function $f_{t_{\{i,..k,l\}}} = f = 1$ and from our assumption $f_{t_{\{i,..,k\}}} \neq f$ and $f_{t_{\{i,..,k\}}} = 0$.

$f(P_i = 1) = \ldots + 1 + ...P_m^{'} = 1$

$f(P_i = 1)_{t_{\{i,..,k\}}} = ...1P_{t_i} + \ldots + P_k^{'} P_{t_k}^{'} \ldots + P_l^{'} + ... + P_m^{'} = 0$

$f(P_i = 1)_{t_{\{i,..,k,l\}}} = ...1P_{t_i} + \ldots + P_k^{'} P_{t_k}^{'} \ldots + P_l^{'} P_{t_l}^{'} + ... + P_m^{'} = 1$

If $f(P_i = 1)_{t_{\{i,..,k\}}} = 0$, then P_{t_i}, $P_k^{'} P_{t_k}^{'}$, $P_l^{'}$ and other products must be 0. If these products are used in $f(P_i = 1)_{t_{\{i,..,k,l\}}}$ it also becomes 0 which contradicts with our assumption $f(P_1 = 1)_{t_{\{i,..,k,l\}}} = f$. This contradiction can be shown for other products as well.

Theorem 2: P_{t_i} is the negation of single literal products or AND of them found in $f(P_i = 1)_{t_{\{i\}}}$, if and only if $f = f_{t_{\{i\}}}$.

Proof : The theorem provides a necessary and sufficient condition.

Sufficiency: If $P_{t_i} = \overline{x_k}$, then $\overline{x_k}$ literal is a single product of $f(P_i = 1)_{t_{\{i\}}}$ which can be shown as $f(P_i = 1)_{t_{\{i\}}} = ... + x_k + P_m^{'}$. When $P_i = 1$ and $P_{t_i} = 0$, literal of faulty product $P_{t_i} = \overline{x_k} = 0$ and $x_k = 1$. Evaluating $f(P_i = 1)_{t_{\{i\}}}$ with given values $f(P_i = 1)_{t_{\{i\}}} = ... + 1 + P_m^{'} = 1$. Our condition for fault tolerance met.

If $P_{t_i} = \overline{x_k}.\overline{x_l}$, then x_k and x_l are single literal products of $f(P_i = 1)_{t_{\{i\}}}$ which can be shown as $f(P_i = 1)_{t_{\{i\}}} = ... + x_k + x_l + P_m^{'}$. When $P_i = 1$ and $P_{t_i} = 0$, literals of faulty product $P_{t_i} = \overline{x_k}.\overline{x_l} = 0$ and negation of P_{t_i} is $x_k + x_l = 1$. Evaluating $f(P_i = 1)_{t_{\{i\}}}$ with given values $f(P_i = 1)_{t_{\{i\}}} = ... + 1 + P_m^{'} = 1$. Our condition for fault tolerance met.

Necessity: If $f_{t_{\{i\}}} = f$, then P_{t_i} is negation of single literal products or AND of them found in $f(P_i = 1)_{t_{\{i\}}}$.

Lets assume P_{t_i} is not negation of single literal products or AND of them found in $f(P_i = 1)_{t_{\{i\}}}$.

If $f = f_{t_{\{i\}}}$, when $P_i = 1$ and $P_{t_i} = 0$, then $f(P_i = 1)_{t_{\{i\}}}$ must be 1. However since P_{t_i} is not the negation of single literal products or AND of them found in $f(P_i = 1)_{t_{\{i\}}}$, so its literals must be a member of products which has two or more literals. $f = f_{t_{\{i\}}}$ condition cannot be met when other literals takes the value of 0 and $f \neq f_{t_{\{i\}}}$.

5. EXPERIMENTAL RESULTS

Standard benchmark circuits (Lgsynth93) are used to measure defect tolerance performances of nano-crossbars. Defect probability/rate of 15% ,which is estimated limit in [27], is considered for each crosspoint independently.

Simulations are conducted in MATLAB without using any parallel computation. Crossbars with random defects are produced with MATLAB's predetermined matrix generator according to the given defect rate. Uniform distribution is adopted for defect occurrence or positions presented in defective nano-arrays

5.1 Algorithm Runtime and Success Rate

As stated before, optimal crossbars are used which means size of the function and crossbar matrix is the same. Since crossbar based nano-arrays is an emerging technology, production is more time consuming and expensive for larger scales. That's why, optimal size crossbars are used because it is important to realize a logic function with a crossbar size as small as possible.

Furthermore, results with 1.5 times larger row and column size also are given which is a common practice in related literature [15] [19] [17] due to the increased solution space which reduces the computational load of the algorithms. As opposed to optimal crossbars, possibility of finding a valid mapping with greater size increase drastically. This can be explained with an example.

Firstly, assuming a 6x6 size logic function is to be mapped on a 6x6 defective crossbar. Row and column permutation of logic function is 6!.6! = 518,400. Secondly, for the same logic function a 9x9 size (1.5 times bigger) defective crossbar is used to be mapped. Calculation of the all possible permutations of row and column is as such $\frac{9!}{(9-6)!} \cdot \frac{9!}{(9-6)!} = 3,657,830,400$. It can be seen from the immense difference between two results, it is highly likely to find a valid mapping with second approach. Therefore, 95% success rate is maintained for the runtime of benchmark function results.

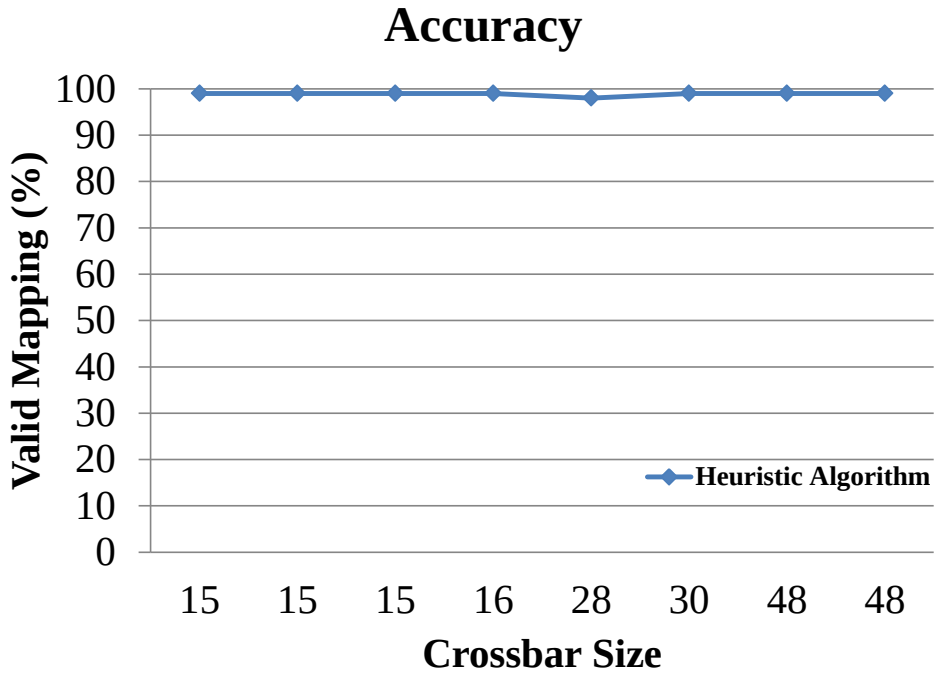


Figure 5.1 : Algorithm accuracy for optimal size crossbars

To obtain defect tolerance values, a sample size of around 600 is used for the accuracy of the runtime results. Runtime fluctuation of results stabilize at this sample size. All experiments run on a 1.70-GHz Intel Core i5 CPU (only single core used) with 4.00 GB memory. Comparison of valid mapping results with exhaustive search is exercised to establish accuracy. Since it is intractable to implement exhaustive search with crosssbar size larger than 7x7, only results pertaining to this limit are presented in Figure 5.1. At the end, a 99% accuracy is obtained.

5.1.1 Runtime

Table 5.1 shows the runtime and success rate of proposed algorithm for benchmark circuits with 15% defect rate. As can be seen from the table, increasing the crossbar size effects the runtime of algorithm immensely. Also runtime and success rate results with using bigger size crossbar is given and showed that the algorithm works very fast for benchmark circuits table5 and t481 with great size.

In Table 5.2 and 5.3, runtime comparison of memetic algorithm and proposed heuristic algorithm (HA) is given . As can be seen from the result, the runtime of the algorithm does not fluctuate unless the size of logic function is altered. All the benchmark functions have the same 40% logic inclusion ratio as well.

Table 5.1 : Runtime (s) and Succes Rate (%) of Proposed Algorithm for Optimal and 1.5 time bigger crossbar size

Benchmark	Size	Optimal Size		1.5 Greater Size	
		Psucc	Avg	Psucc	Runtime
5xp1	75 x 14	100%	0.013	100%	0.003
inc	34 x 14	96%	0.18	100%	0.001
clip	167 x 18	100%	0.03	100%	0.01
misex2	50 x 29	100%	0.006	100%	0.002
9sym	87 x 18	100%	0.008	100%	0.005
bw	65 x 10	100%	0.01	100%	0.002
rd53	32 x 10	98%	0.007	100%	0.001
rd73	141 x 14	100%	0.035	100%	0.01
9sao	58 x 20	-	-	100%	0.003
table5	158 x 34	-	-	100%	0.02
t481	481 x 32	-	-	100%	0.2

Another factor which has a strong effect on the runtime is ratio of row over column size which is the aspect ratio of matrix. As stated before, the row or column with the smaller size is chosen as a constant permutation. This is preferred in order to decrease the probability of mismatches. However, when the size of the element with constant permutation is increased, algorithm runtime do not increase linearly.

5.1.2 Success rate

A 95% success rate is maintained for benchmark results for crossbars with 1.5 times greater size which means proposed algorithm should be able to find a valid mapping at least 95% of the given sample size. Explanation of the drastic raise in the number of solution space due to using crossbars with greater size than the logic function is given previously. Because of this, it is reasonable to expect to find a valid mapping 95% of the time.

Another point needs to be elaborated is the number of trials which is column permutation. As it will be explained in section 5.2, initial sorting of matrices is sufficient to find a valid mapping. However, in case of optimal crossbar further column permutation necessary in order to find a valid mapping 95% of the sample size. When the experiment is conducted with the trial numbers for optimum performance and

Table 5.2 : Runtime Comparsion of Memetic Algrotihm and HA for 16 x 16 Benchmarks with 1.5 Bigger Size

No	MA/FA		HA	
	Psucc	**Avg**	**Psucc**	**Runtime**
1	100%	0.013	100%	0.002
2	100%	0.002	100%	0.001
3	100%	0.002	100%	0.001
4	100%	0.004	100%	0.001
5	100%	0.01	100%	0.001
6	100%	0.003	100%	0.001
7	100%	0.003	100%	0.001
8	100%	0.031	100%	0.001
9	100%	0.046	100%	0.001
10	100%	0.003	100%	0.001
11	100%	0.025	100%	0.001
12	100%	0.007	100%	0.001
13	100%	0.004	100%	0.001
14	100%	0.007	100%	0.001
15	100%	0.002	100%	0.001
16	100%	0.003	100%	0.001
17	100%	0.01	100%	0.001
18	100%	0.003	100%	0.001
19	100%	0.002	100%	0.001
20	100%	0.002	100%	0.001

success rate for the algorithm, it is concluded that 3000 trials is the most sensible limit.

5.2 Effectiveness of Sorting

In the 3th step of algorithm, it is stated that if there is no matching between sorted logic and crossbar matrix, column permutation needs to be changed. 3000 trials (number of column permutations) is chosen for maintaining 95% success rate.

Simulation results showed that for optimal size crossbars, when the size of benchmark circuit increase so does the number of permutation performed to find a valid mapping.

Table 5.3 : Runtime Comparsion of Memetic Algrotihm and HA for 24 x 24 Benchmarks with 1.5 Bigger Size

No	MA/FA		HA	
	Psucc	**Avg**	**Psucc**	**Runtime**
1	100%	1.487	100%	0.002
2	100%	0.018	100%	0.001
3	100%	0.266	100%	0.002
4	100%	0.238	100%	0.002
5	100%	0.904	100%	0.001
6	100%	0.217	100%	0.002
7	100%	2.357	100%	0.001
8	100%	1.277	100%	0.001
9	100%	2.268	100%	0.002
10	100%	0.588	100%	0.002
11	100%	0.74	100%	0.002
12	100%	0.179	100%	0.002
13	100%	0.914	100%	0.002
14	100%	0.211	100%	0.001
15	100%	0.087	100%	0.001
16	100%	0.268	100%	0.002
17	100%	0.173	100%	0.001
18	100%	0.611	100%	0.002
19	100%	4.199	100%	0.002
20	100%	5.776	100%	0.001

As for the crossbars with 1.5 times bigger size, in average no column permutation is performed to find a valid mapping which means first sorting is sufficient.

The number of permutations for each sample for benchmark circuits simulation is demonstrated in Figure 5.2 for 50 samples. Results presented in Figure 5.2 is for optimal crossbars. For relatively smaller size benchmark circuits, cumulation of column permutations between 0 and 10 indicates the effectiveness of sorting even for optimal size crossbar. However, the increase in the size of benchmark circuit, column permutation increases for each sample. In Figure 5.2, clip is an example of the mentioned case.

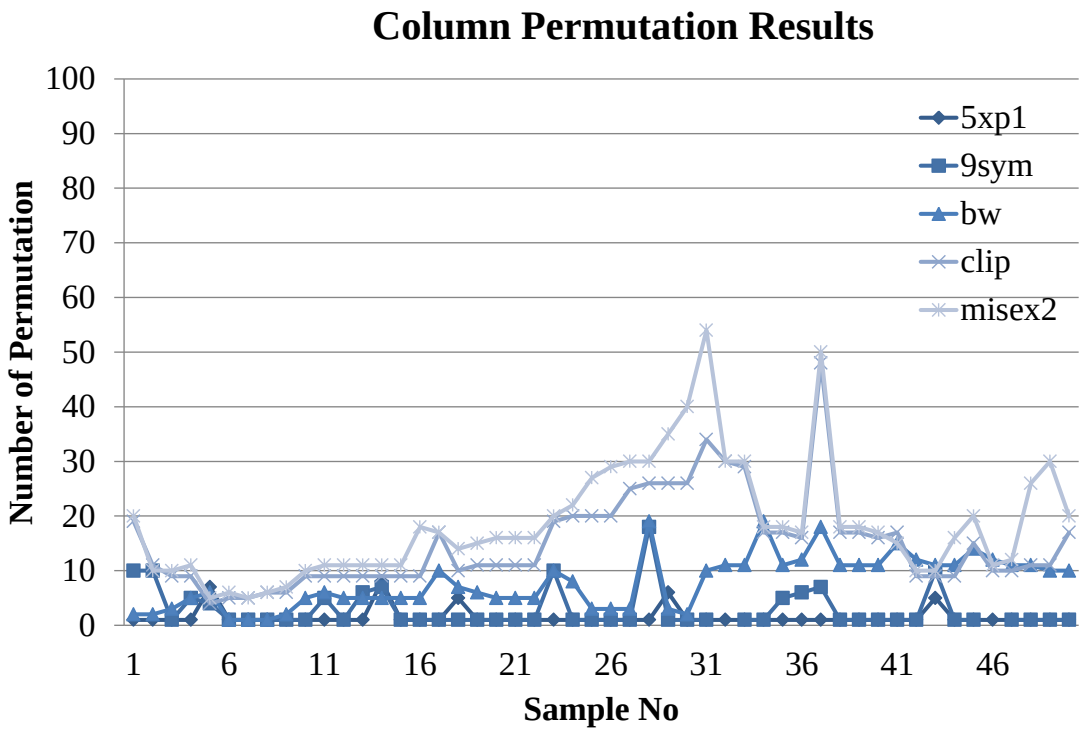


Figure 5.2 : Number of permutation to find a valid mapping for each sample

In addition to the size and column permutation relationship, it should be emphasized that, in case of crossbars with 1.5 times bigger size, initial sorting is sufficient for finding a valid matching. In the simulation results, only 1% of the samples required column permutation for valid mapping.

5.3 Runtime and Aspect Ratio Relationship

In section 3 the method of choosing which dimension of the logic matrix stays the same is shown, which is the one with the minimum size, as the backtracking search advances. The reason behind that is to decrease the possibility of mismatches. We first explain this with using probability and then present the experimental results.

This can be explained with a quantitative example. Firstly, given a logic matrix with 40% logic inclusion ratio and a 36 x 10 size, according to the method of choice, column permutation stays constant so every row needs to be matched has 10 elements. 40% logic inclusion ratio means there are 4 element that can be matched with defective elements in crossbar matrix. If the defect rate is 15%, then there are approximately 2 defective elements in every row of crossbar matrix which has the same size 36 x 10. In short, out of 10 elements there are 2 defective elements in crossbar matrix

and 4 elements in logic matrix which can be matched with defective elements. The number of different positions for mentioned elements can be calculated with $C(10,4)$ and $C(10,2)$ which are 210 and 45. Secondly, given a logic matrix with 40% logic inclusion ratio and 36 x 20 size. according to the the method of choice, column permutation stays constant so every row needs to be matched has 20 elements. 40% logic inclusion ratio means there are 8 element that can be matched with defective elements in crossbar matrix. If the defect rate is 15%, then there are approximately 3 defective elements in every row of crossbar matrix which has the same size 36 x 20. Using the same calculations, the number of different positions for mentioned elements can be calculated with $C(20,8)$ and $C(20,3)$ which are 125,970 and 1,140.

In the fist example there are only 45 different position which means there is $\frac{45}{210} = 21\%$ chance a matching occurs between two rows. As for the second example, there are 1,140 different position which means there is only $\frac{1,140}{125,970} = 0.009\%$ chance a matching occurs between two rows. The drastic increase in the positions of defective elements is the cause of mismatches.

In Figure 5.3, runtime change of proposed algorithm is given for logic function while row dimension stays the same column dimension increases in the increment of 6. For the sake of fairness, it is specified tha logic inclusion ratios as 40% for the all functions. In Figure 5.3, while runtime of algorithm is 0.002s for the crossbar with size of 48 x 18, runtime result for the crossbar with size of 48 x 42 is close to 3s. It is clear from the presented results, runtime expands non-linearly.

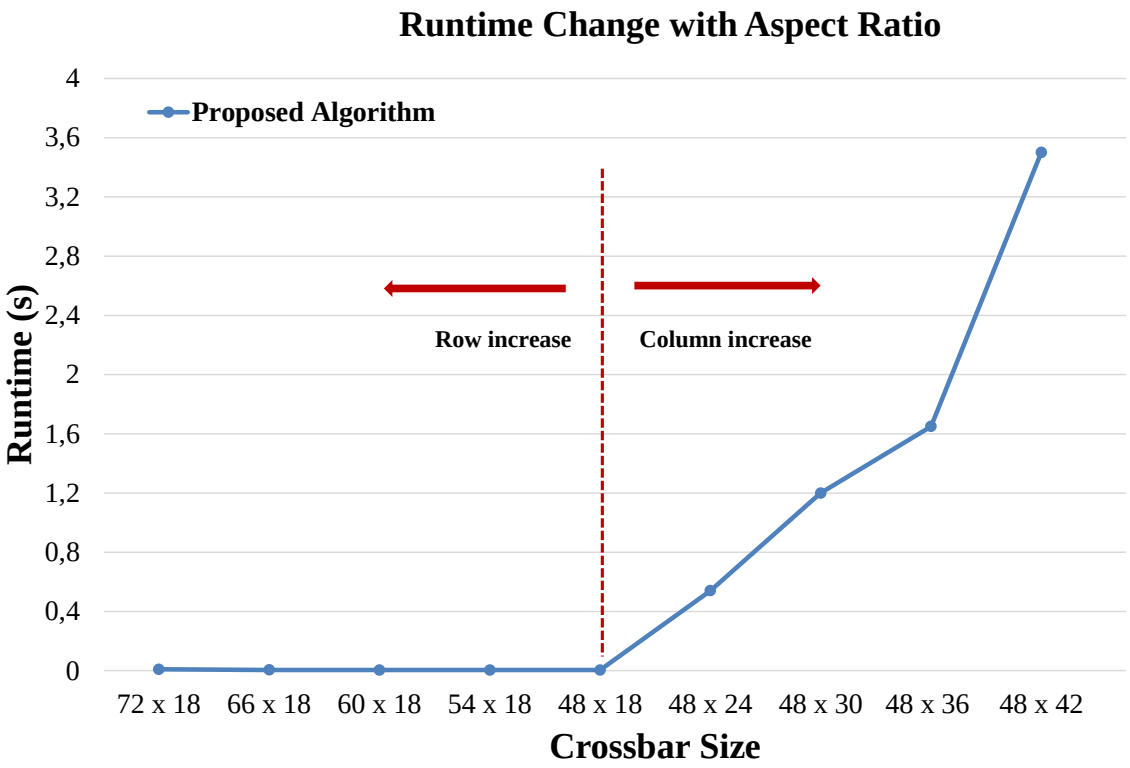


Figure 5.3 : Runtime increases non-linearly when the constant dimension increase with the same logic inclusion ratio

6. CONCLUSION

In this study, a novel heuristic algorithm is proposed to evaluate defect tolerance performances of nano-crossbar based switching arrays with permanent defects and exercised as a defect-tolerant logic mapping for defective nano-crossbars. The algorithm uses indexing and mapping techniques that eliminates considerable amount of crossbar mapping permutations which is the main headache for related studies in the literature. Runtime and success rate comparison of industrial benchmark circuits oriented algorithm results are given for different algorithms.

In addition, a model for transient fault performance of logic functions is given with formally finding tolerable equivalent functions. Failure analysis depicted that transient fault tolerance of nano-crossbar alters from expected results and yields better results for stuck-closed fault types. It is also shown that, mapping design can be calibrated according to inherent defect types of crossbar. Considering the defect/fault types, logic function can be manipulated in accordance with proposed method.

REFERENCES

[1] **Goldstein, S.C. and Budiu, M.** (2001). Nanofabrics: Spatial computing using molecular electronics, *ACM SIGARCH Computer Architecture News*, *29*(2), 178–191.

[2] **Strukov, D.B. and Likharev, K.K.** (2012). Reconfigurable nano-crossbar architectures, *Nanoelectronics and Information Technology*, 543–562.

[3] **Snider, G.S. and Williams, R.S.** (2007). Nano/CMOS architectures using a field-programmable nanowire interconnect, *Nanotechnology*, *18*(3), 035204.

[4] **Keyes, R.** (2000). Miniaturization of electronics and its limits, *IBM Journal of Research and Development*, *44*(1/2), 84.

[5] **Huang, Y.**, **Duan, X.**, **Cui, Y.**, **Lauhon, L.J.**, **Kim, K.H. and Lieber, C.M.** (2001). Logic gates and computation from assembled nanowire building blocks, *Science*, *294*(5545), 1313–1317.

[6] **Yao, J.**, **Yan, H.**, **Das, S.**, **Klemic, J.F.**, **Ellenbogen, J.C. and Lieber, C.M.** (2014). Nanowire nanocomputer as a finite-state machine, *Proceedings of the National Academy of Sciences*, *111*(7), 2431–2435.

[7] **Heath, J.R.**, **Kuekes, P.J.**, **Snider, G.S. and Williams, R.S.** (1998). A defect-tolerant computer architecture: Opportunities for nanotechnology, *Science*, *280*(5370), 1716–1721.

[8] **Ziegler, M.M. and Stan, M.R.** (2003). CMOS/nano co-design for crossbar-based molecular electronic systems, *Nanotechnology, IEEE Transactions on*, *2*(4), 217–230.

[9] **Shrestha, A.M.S.**, **Tayu, S. and Ueno, S.** (2009). Orthogonal Ray Graphs and Nano-PLA Design., *ISCAS*, pp.2930–2933.

[10] **Tahoori, M.B.** (2005). A mapping algorithm for defect-tolerance of reconfigurable nano-architectures, *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, IEEE, pp.668–672.

[11] **Al-Yamani, A.**, **Ramsundar, S.**, **Pradhan, D.K.** *et al.* (2007). A defect tolerance scheme for nanotechnology circuits, *Circuits and Systems I: Regular Papers, IEEE Transactions on*, *54*(11), 2402–2409.

[12] **Yuan, B. and Li, B.** (2014). A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar, *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *10*(3), 25.

[13] **Rao, W.**, **Orailoglu, A. and Karri, R.** (2006). Topology aware mapping of logic functions onto nanowire-based crossbar architectures, *Proceedings of the 43rd annual Design Automation Conference*, ACM, pp.723–726.

[14] **DeHon, A. and Naeimi, H.** (2005). Seven strategies for tolerating highly defective fabrication, *Design & Test of Computers, IEEE*, *22*(4), 306–315.

[15] **Gören, S.**, **Ugurdag, H.F. and Palaz, O.** (2011). Defect-aware nanocrossbar logic mapping through matrix canonization using two-dimensional radix sort, *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *7*(3), 12.

[16] **Yang, J.S. and Datta, R.** (2011). Efficient function mapping in nanoscale crossbar architecture, *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on*, IEEE, pp.190–196.

[17] **Zamani, M.**, **Mirzaei, H. and Tahoori, M.B.** (2013). ILP formulations for variation/defect-tolerant logic mapping on crossbar nano-architectures, *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *9*(3), 21.

[18] **Yuan, B.**, **Li, B.**, **Weise, T. and Yao, X.** (2014). A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nanoarchitectures, *Evolutionary Computation, IEEE Transactions on*, *18*(6), 846–859.

[19] **Su, Y. and Rao, W.** (2014). An integrated framework toward defect-tolerant logic implementation onto nanocrossbars, *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *33*(1), 64–75.

[20] **Likharev, K.K. and Strukov, D.B.**, (2005). CMOL: Devices, circuits, and architectures, Introducing Molecular Electronics, Springer, pp.447–477.

[21] **Yan, H.**, **Choe, H.S.**, **Nam, S.**, **Hu, Y.**, **Das, S.**, **Klemic, J.F.**, **Ellenbogen, J.C. and Lieber, C.M.** (2011). Programmable nanowire circuits for nanoprocessors, *Nature*, *470*(7333), 240–244.

[22] **Gholipour, M. and Masoumi, N.** (2013). Design investigation of nanoelectronic circuits using crossbar-based nanoarchitectures, *Microelectronics Journal*, *44*(3), 190–200.

[23] **DeHon, A. and Gojman, B.** (2011). Crystals and snowflakes: building computation from nanowire crossbars, *Computer*, (2), 37–45.

[24] **Lu, W. and Lieber, C.M.** (2006). Semiconductor nanowires, *Journal of Physics D: Applied Physics*, *39*(21), R387.

[25] **Pesetski, A.A.**, **Zhang, H.**, **Adam, J.D.**, **Przybysz, J.**, **Murduck, J.**, **Goldstein, N. and Baumgardner, J.**, (2010), Carbon nanotube field effect transistor, uS Patent 7,714,386.

[26] **DeHon, A. and Wilson, M.J.** (2004). Nanowire-based sublithographic programmable logic arrays, *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, ACM, pp.123–132.

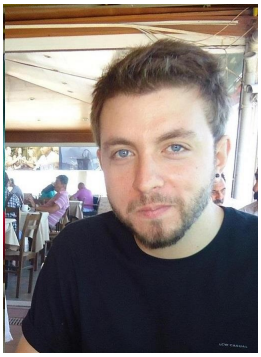
[27] **Haselman, M. and Hauck, S.** (2010). The future of integrated circuits: A survey of nanoelectronics, *Proceedings of the IEEE*, *98*(1), 11–38.

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

[29] **Tunali, O. and Altun, M.** Permanent and Transient Fault Tolerance for Reconfigurable Nano-Crossbar Arrays, *Transaction on Computer Aided Design*, *Under Review*.

[30] **Simsir, M.O.**, **Cadambi, S.**, **Ivančić, F.**, **Roetteler, M. and Jha, N.K.** (2009). A hybrid nano-CMOS architecture for defect and fault tolerance, *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *5*(3), 14.

CURRICULUM VITAE



Name Surname: Onur Tunalı

Place and Date of Birth: Tekirdağ, 08.11.1987

Adress:

E-Mail: onur_tunali@yahoo.com

B.Sc.: Mathematics

M.Sc.: Nanoscience and Nanoengineering

Professional Experience and Rewards:

List of Publications and Patents:

PUBLICATIONS/PRESENTATIONS ON THE THESIS

- **Tunali O.** , Altun M., 2015: Defect tolerance in diode, FET, and four-terminal switch based nano-crossbar arrays. *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, July, 2015 Boston, United States of America.