ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

A HILL CLIMBING ALGORITHM WITH PRE-PROCESSING AND A COLUMN REPLICATION METHOD FOR VARIATION TOLERANT LOGIC MAPPING PROBLEM

M.Sc. THESIS

Furkan PEKER

Department of Electronics and Communications Engineering

Electronics Engineering Programme

August 2017

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

A HILL CLIMBING ALGORITHM WITH PRE-PROCESSING AND A COLUMN REPLICATION METHOD FOR VARIATION TOLERANT LOGIC MAPPING PROBLEM

M.Sc. THESIS

Furkan PEKER
(504141208)

Department of Electronics and Communications Engineering

Electronics Engineering Programme

Thesis Advisor: Asst. Prof. Mustafa ALTUN

August 2017

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

NANODİZİNLERDE VARYASYON TOLERANSLI MANTIKSAL HARİTALAMA İÇİN ÖN AŞAMALI BİR TEPE TIRMANMA ALGORİTMASI VE SÜTUN KOPYALAMA METHODU

YÜKSEK LİSANS TEZİ

Furkan PEKER
(504141208)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Asst. Prof. Mustafa ALTUN

Ağustos 2017

Furkan PEKER, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504141208 successfully defended the thesis entitled “A HILL CLIMBING ALGORITHM WITH PRE-PROCESSING AND A COLUMN REPLICATION METHOD FOR VARIATION TOLERANT LOGIC MAPPING PROBLEM”, 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 :	**Prof. Müştak Erhan YALÇIN** Istanbul Technical University	.....
	Prof. Sezer Gören UĞURDAĞ Yeditepe University	.....
	... ...	.....
	... ...	.....
	... ...	.....

Date of Submission : **25 August 2017**
Date of Defense : **21 August 2017**

To my dear family,

FOREWORD

Emerging computation methods are needed to be developed for the post Moore's era and nano-crossbar computational arrays are one of the pioneers of these methods. Due to their advantages over manufacturing methods and device density, there are considerably high defect and variations on these structures. Therefore, defect and variation tolerant designs become one of the critical aspects of nano-crossbar computational array designs. In this work, new to literature and successful algorithms and design paradigms presented for nano-crossbar arrays and results discussed. This work is supported by and a part of the EU-H2020-RISE project NANOxCOMP #691178 and the TUBITAK-Career project #113E760.

August 2017

Furkan PEKER
(Mechatronics Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

CMOS	**:** Complementary Metal Oxide Semiconductor
FET	**:** Field Effect Transistor
PLA	**:** Programmable Logic Array
VTLM	**:** Variance Tolerant Logic Mapping
D/VTLM	**:** Defect and Variation Tolerant Logic Mapping
NP	**:** Non-deterministic Polynomial Time
ILP	**:** Integer Linear Programming
COV	**:** Coefficient of Variation
RP	**:** Reference Performance
FM	**:** Function Matrix
VM	**:** Variance Matrix
IMV	**:** Input Mapping Vector
OMV	**:** Output Mapping Vector
CR	**:** Crosspoint Ratio
PM	**:** Performance Matrix
FPM	**:** FET Performance Matrix

SYMBOLS

f	**:** Function
t	**:** Time
n	**:** Column Count
m	**:** Row Count
$\mathbf{V_{th}}$	**:** Threshold Voltage
$\mathbf{R_{onFET}}$	**:** On Resistance of FET
$\mathbf{R_{offFET}}$	**:** Off Resistance of FET
μ	**:** Mean of a Gaussian Random Variable
σ	**:** Standard Deviation of a Gaussian Random Variable

LIST OF TABLES

Page

LIST OF FIGURES

Page

A HILL CLIMBING ALGORITHM WITH PRE-PROCESSING AND A COLUMN REPLICATION METHOD FOR VARIATION TOLERANT LOGIC MAPPING PROBLEM

SUMMARY

Nano-crossbar arrays are generally manufactured with faster and less costly methods than traditional tow-down lithography methods. However, due to very small device sizes, there is less control over these manufacturing processes. Therefore, crossbar arrays suffer from process variations and defects with considerably higher rates than lithography processes. This variations makes performance of any manufactured array unpredictable and worse than expected. As a result, variation and/or defect tolerant logic mapping algorithms needed to be developed for this problem.

In this work, I focus on maximum delay increase due to process variations and focused on optimize this objective. Firstly, a matrix reducing method is proposed to achieve better runtimes for any variation tolerant logic mapping algorithm for the maximum delay optimization. In this method, we find columns which have very low (or no) probability to become the worst case on each performance test and reduce matrix bu deleting these columns. Deleted columns mapped randomly on final mapping. Note that this reducing method is algorithm independent, which means that it can be implemented on any variation tolerant logic mapping search algorithm with same performance criteria.

As a variation tolerant logic mapping scheme, a Hill Climbing row search with column re-ordering (as restart points to search) algorithm is proposed. This method restarts Hill Climbing row search after reaching an unoptimizable point by re-ordering columns with data gathered on row search, until any of the stop criteria reached. Also, a greedy initial column ordering algorithm explained and matrix reducing method is applied to Hill Climbing algorithm as pre-processing phase.

As an extension for our Hill Climbing algorithm on variance tolerance, we modeled variation tolerant logic mapping problem as defect tolerant logic mapping problem by adding defective crosspoints on array matrix and run our algorithm with these defects. Our algorithm successfully tolerate different defect rates for random matrices or benchmark circuits.

Lastly, as a probabilistic approach by process parameters, a worst case route replication method is proposed. This method do not use any search algorithm or need any individual crosspoint characteristic testings, which are time and resource consuming processes. This approach uses area and power overhead as trade-off values, while power-delay product is generally having decreasing characteristics.

We simulated all proposed algorithms and methods on Simulation Results Section. We explained some algorithms from the literature and compared our Hill Climbing with Column re-ordering algorithm with these. Our algorithm achieves performance outputs up to 15% and runtimes up to 150-400 times better than methods on the literature, depending on the array size. As a inspection of matrix reducing method, we applied

this method on different benchmarks and algorithms to compare time optimization results. Up to 72% time improvement is achieved for certain benchmarks and different algorithms by this reducing approach. Our algorithm achieved 15-30% defect tolerance for random arrays with 40% crosspoint ratio and 5-20% defect tolerance for our benchmark set.

We discuss to implement this methods for different nano-crossbar array types like diode or 4-terminal as future work. Also, we aim to improve presented algorithms since both matrix reducing and initial column mapping methods are in their preliminary states in this work.

NANODİZİNLERDE VARYASYON TOLERANSLI MANTIKSAL HARİTALAMA İÇİN ÖN AŞAMALI BİR TEPE TIRMANMA ALGORİTMASI VE SÜTUN KOPYALAMA METHODU

ÖZET

Nano anahtarlamalı dizinler tasarım özellikleri ve boyutları ile gelen yüksek hız ve düşük güç tüketimi özellikleri sayesinde yeni gelişen teknolojiler arasında öne çıkmaktadırlar. Bu özelliklerine ek olarak, genellikle geleneksel litografi yöntemlerine göre daha hızlı ve ucuz yöntemler ile üretilmektedirler. Öte yandan, sahip oldukları küçük boyutlar ve genellikle kendiliğinden kurulma teknolojileri ile gerçeklenmeleri, bu yapıların hatalara ve varyasyonlara duyarlılığını geneleksel litografi yöntemlerine oranla yüksek seviyelere çekmektedir. Yüksek hata ve varyasyon oranları, bu etkenlere karşı tolerans teknikleri geliştirilmesi ihtiyacını doğurmuştur ve nano-anahtarlamalı dizinler için önemli bir tasarım parametresi haline gelmiştir. Literatürde bu tolerans problemi farklı etkenlere göre farklı yöntemler ile incelenmiştir.

Kendiliğinden kurulma yöntemleri olasılıksal doğalarından ve tasarım boyutlarının küçüklüğünden dolayı yüksek hata oranları ile nano-anahtarlamalı dizinleri gerçeklerler. Bu hatalar, nano-anahtarlamalı dizin üzerindeki bir kesişim noktasında oluşturulmak istenilen cihazın (diyot, FET veya 4-uçlu anahtar) veya bu olası cihazların ararındaki tel bağlantılarının işlevselliğini bozmaktadırlar. Bu hatalar, bir cihazın her zaman açık veya kapalı olarak çalışması şeklinde veya bir bağlantı telinin kopması şeklinde olabilir. Dolayısıyla, üretilen bir nano-anahtarlamalı dizinde kullanılacak cihazların seçimi, devrenin işlevselliği açısından kritik anlamda önem taşır.

Hatalı cihazlara ek olarak, devrenin işlevselliğini bozmayan fakat çalışma performansını önemli ölçüde etkileyen varyasyonlar nano-anahtarlamalı dizinlerde mevcuttur. Hatalar, bu varyasyonların aşırılığından kaynaklaran durumlar olarak da tanımlanabilir. Bir nano-telin çapındaki aşırı varyasyon bu telin açık devre olarak çalışmasına sebep olacak seviyelere gelmesine sebep olabilir. Bu durumda söz konusu tel hatalı olarak değerlendirilir ve hata toleransı teknikleri ile kullanımından kaçınılır. Öte yandan, aşırı seviyede olmayan varyasyon değerleri işlevselliği bozmazken, devrenin çalışma performasını önemli ölçüde etkileyebilir. Nanotel boyutlarının sayılı atom çaplarına yaklaşması ile tek bir dopant atom eksikliği bile yarıiletken bir cihazın performans parametrelerini etkileyebilmektedir. Bununla beraber nanotel, oksit kalınlığı vb. tasarım parametrelerinde oluşabilecek boyut varyasyonları da direnç ve kapasite değerlerini önemli ölçüde etkileyebilmektedir. Dolayısıyla, gecikme ve güç tüketimi değerleri nano-dizin üzerinde kullanılan cihazlara göre saçınık bir karaktere sahip olmaktadır. Bu etkenlerin sonucu olarak varyasyon toleransı tekniklerinin geliştirilmesi ihtiyacı doğmuştur. Bu teknikler, mantık fonksiyonunun anahtarlamı nano-dizin üzerine performans parametreleri göz önüne alarak en iyi şekilde yerleştirilmesini amaçlamaktadır. Varyasyon toleranslı mantıksal haritalama adı altında literatürde farklı yöntemler sunulmuştur.

Fonksiyon matrisinde, satır girişleri fonksiyon değişkenlerini, sütun çıkışları ise fonksiyon çarpımlarını ifade etmektedir. Eğer bir fonksiyon girişi, çıkış çarpımında bulunuyorsa bu satır ve sütunları kesişim noktalarında bir anahtarı ifade edecek şekilde 1, aksi durumda ise 0 bulunur. Gerçeklenmek istenilen mantıksal fonksiyona bağlı olarak, nano-anahtarlamalı dizinde bulunan cihazların önemli bir kısmı zaten kullanılmamaktadır. Dizin üzerindeki giriş ve çıkışların sıraları değiştirilerek gerçeklenecek mantıksal fonksiyonun işlevselliği değiştirilmeden, farklı cihazlar kullanılabilir. Mantıksal fonksiyonun değişken girişlerinin ve çarpım çıkışlarının konumlarının nano-anahtarlamalı devre üzerine yerleştirilmesi işlemi mantıksal haritalama olarak adlandırılır. Fonksiyon matrisinin bu değiştirilebilirlik özelliği hata toleransı ve varyasyon toleransı tekniklerinin mantıksal haritalama ile gerçekleştirilebilmesini sağlamaktadır.

Varyasyon toleranslı mantıksal haritalama işlemi için fonksiyon matrisi ile beraber nano-anahtarlamalı dizin matrisinin her bir kesişim noktasının aktif bir cihaz olarak kullanıldığı durumda çıkışta oluşan gecikmeye katkısı bilinmelidir. Her bir kesişim noktasının gecikmeye katkısı ile oluşturulan, nano-anahtarlamalı dizin ile aynı boyuttaki matrise varyasyon matrisi adı verilmektedir. Varyasyon toleranslı mantıksal haritalama işlemi, fonksiyon matrisinin girişlerinin (satırlar) ve çıkışlarının (sütunlar) varyasyon matrisi üzerine en uygun şekilde sıralanması ile gerçekleşir. Varyasyon matrisinin elde edilmesi için üretilen devrenin gecikme testinden geçirilmesi gereklidir, dolayısıyla zaman ve maliyet gerektirir.

Literatürde varyansyon toleranslı mantıksal haritalama yöntemleri genel anlamda iki amaca yönelik tasarlanmışlardır. Bunların ilki, en yüksek gecikmeye sahip hattın gecikme değerinin olabildiğince düşük olmasıdır. İkincisi ise en yüksek gecikmeye sahip hat ile en düşük gecikmeye sahip hat arasındaki gecikme farkının olabildiğince küçük olmasıdır. İkinci amaç gecikmeler arasındaki farkı azaltarak devrenin gecikme varyansını azaltmaya çalışmaktadır fakat bunu yaparken bazı durumlarda yüksek gecikmeye sahip kesişim noktalarını kullanmaktadır. Devrenin en yüksek çalışabileceği hızı en yüksek gecikmeli hat belirlediğinden, önerilen algoritma sadece ilk amaca odaklanarak tüm hesaplama gücünu daha düşük gecikmeye sahip bir haritalama elde etmek için tasarlanmıştır. Ayrıca, kullanım alanı daha geniş olan ve ilgi gören FET tipi nano-anahtarlamalı dizinler için çalışılmıştır.

FET tipi nano-anahtarlamalı dizinlerde her bir sütun mantıksal fonksiyondaki bir çarpımı ifade eder. En yüksek gecikmeli durumda bu hatlardan sadece birisi aktiftir, bu durumda devreden en az akım akacağından gecikme yüksek olacaktır. Söz konusu en yüksek gecikmeli hattın sahip olduğu gecikme, o hat üzerinde aktif olarak kullanılan kesişim noktalarındaki cihazların çıkış gecikmesine olan katkılarının toplamıdır. Fonksiyon matrisinde ‘1’e denk gelen kesişim noktaları FET, ‘0’a denk gelen kesişim noktaları ise kısa devre tel olarak çalışır, dolayısıyla Hadamart çarpımı devrenin haritalama sonrasında sahip olacağı performansı ifade etmekte yeterlidir. Özetle, fonksiyon matrisi ile varyasyon matrisinin Hadamart çarpımının en yüksek toplamlı sütunu, en yüksek gecikmeyi ifade eder. Varyasyon toleranslı mantıksal haritalama işleminde, oluşacak en yüksek gecikmeli sütunun değerinin en düşük değere getirilmesi amaçlanmaktadır.

Bu çalışmada varyasyon toleranslı mantıksal haritalama problemini FET tipi nano-anahtarlamalı dizinlerde çözmek için bir tepe tırmanma algoritması önerilmiştir. Bu algoritma 2 ön aşama, 2 arama olmak üzere 4 tip algoritmanın birlikte çalışması

ile oluşturulmuştur. Ön aşama algoritmaları literatürdeki diğer arama algoritmalarıyla da kullanılabilirler. Bu yapıları ile algoritma-bağımsızdırlar ve ön aşaması oldukları algoritmaların performanslarını iyileştiricidirler. Tepe Tırmanma algoritması, sabit bir sütun sıralaması için satır sıralamalarını arayarak iyileştirilmiş sonuçlar bulmaya çalışır.

Üretim aşamasında pek çok etken beklenen yapıda sapmalara sebep olduğundan, simulasyonlar için her bir kesişim noktasında oluşacak gecikme değeri rastgele bir Normal dağılımlı sayı olarak modellenmiştir. Normal dağılım için beklenen ortalama değer ve standart sapma değerleri üretim tekniğine bağlı olarak belirlenir ve varyasyon matrisi bu dağılımdan alınan rastgele sayılar ile oluşturulur. Simulasyonlarda yüksek ve düşük gecikmeli değerler arasında 10-20 kat fark olacak şekilde ortalama değer ve standart sapma değerleri belirlenerek yüksek varyasyon durumları incelenmeye çalışılmıştır.

Önerilen algoritmanın ilk adımı olarak bir fonksiyon matrisi indirgeme yöntemi önerilmiştir. Her bir fonksiyon matrisi sütununun, her bir varyasyon matrisi sütununda alabileceği değerler rahatlıkla hesaplanabilir. Burada referans olarak en yüksek sayıda '1'e sahip olan sütunun alabilceği en küçük değeri parametre olarak kullanabiliriz, çünkü bu değer verilen fonksiyon ve varyans matrisi çifti için ulaşılabilecek en iyi performans değeridir. Eğer, diğer herhangi bir sütunun alabileceği en yüksek değer, performans değerin altındaysa bu sütun hesaplaması gereksiz bir sütun olarak kabul edilebilir. Söz konusu sütunun en yüksek gecikmeye sahip olması mümkün olmadığından, arama algoritmasının her seferinde bu sütun için işlem yapmasına gerek yoktur. Tüm hesaplaması gereksiz sütunlar bulunduktan sonra fonksiyon matrisinden silinerek indirgenmiş, performans eşdeğeri bir matris elde edilir. İndirgeme işlemi sadece en yüksek gecikmeyi iyileştirmeye çalışırken geçerlidir ve bu amaçla çalışan tüm algoritmaların bir ön aşaması olarak kullanılabilir. Fonksiyon matrisinin sütunlarının sahip oldukları '1' sayıları arasındaki fark arttıkça bu yöntemin iyileştirme oranı da artmaktadır. Bazı matrislerde hiç ingirgeme yapılamazken, bazı bençmarklarda %72'lere kadar sütun indirgenmesi yapılabilmektedir. Bu oran kullanılan algoritmanın sütun işlemi yoğunluğuna göre farklı oranlarda iyileştirme sağlar. %76'lara varan zaman iyileştirmesi bu yöntem ile sağlanmıştır.

İkinci adım olarak, bir sütun sıralaması elde edilerek Tepe Tırmanma algoritmasının başlangıç noktası belirlenir. Bu sıralama için, her bir fonksiyon sütununun, her bir varyans sütununda alabileceği en düşük değerler belirlenir. Bu noktadan sonra, en kritik (en yüksek sayıda '1'e sahip) fonksiyon sütunundan başlanarak, her bir fonksiyon sütunu en düşük değeri alabileceği varyans sütununa atanır. Daha önce kullanılan bir varyans sütunu bir daha kullanılamaz. Bu ilk haritalama konumu, aynı çalışma süresinde rastgele başlangıç noktasına göre %2-3 dolaylarında daha başarılı sonuçlar alınmasını sağlamıştır.

Arama algoritması olarak kullanılan Tepe Tırmanma algoritması, sabit bir sütun sıralaması için satır sıralamalarını değiştirerek alabileceği en iyi sonuca ulaşmaya çalışır. Artık iyileştiremediği bir noktaya ulaştığında ise yeni bir sütun sıralaması bularak aramaya tekrar başlar. Bu yeni sütun bulunması işlemi arama algoritmasının ikinci kısmı olan sütun yeniden haritalama kısmıdır. Tepe Tırmanma algoritması arama sırasında, her performans testinde hangi sütunun en yüksek gecikme değeri verdiğini kaydeder. Sütun yeniden haritalama algoritması çalışacağı zaman bu bilgiyi Tepe Tırmanma algoritmasından alır ve en fazla sayıda en yüksek gecikme değerini vermiş

sütunun yerine fonksiyon matrisinin düşük ‘1’ sayısına sahip sütunlarından biri ile yeri değiştirilir. Sütun yeniden sıralama algoritması çalıştıktan sonra Tepe Tırmanma algoritması tekrar çalışır ve bu döngü durma parametrelerinden birine ulaşılıncaya kadar devam eder. Kullanılan bu method literatürde verilen algoritmalara göre özellikle yüksek boyutlu matrislerde (24x24, 48x48) %15’lere yaklaşan iyileştirme oranlarına sahiptir. Düşük boyutlu matrislerde (6x6) ise benzer iyileştirme oranlarına rağmen 5-20 kat daha hızlı çalışmaktadır.

Varyasyon toleransı için önerilen Tepe Tırmanma algoritmasının ek bir kullanımı olarak, problem tanımını değiştirerek bu algoritmanın hata toleransı işlemlerini de gerçekleştirmesi sağlanmıştır. Bunun için varyasyon matrisine hatalı kesişim noktaları eklenerek bu noktaların gecikme değerleri beklenen gecikme değerlerinin 100 katı olarak verilmiştir. Bu durumda çalışan algoritma çok yüksek gecikmeye sahip kesişim noktalarını tolere etmeye çalışacağından hata toleransı yapacak şekilde çalışmaya başlamış olur. Bu şekilde yapılan simülasyonlarda %40 kesişim noktası yoğunluğuna sahip rastgele matrislerde %15-30 hata oranları tolere edilebilmiştir. Bu çalışmada kullanılan stanadart bençmark seti için %5-20 hata oranları tolere edilebilmiştir.

Son olarak, üretimde oluşacak varyanslara farklı bir yaklaşım olarak, alandan taviz vererek sütun kopyalama yöntemi kullanılmıştır. Gecikmelere sebep olan etkenin ağırlıklı olarak dirençlerden kaynaklı olduğu ve nano-dizinin iç kapasitelerine göre daha yüksek bir çıkış kapasitesi sürüldüğü varsayıldığında, aynı anda iki veya daha fazla sütunun aktif olması daha yüksek akım akışına fırsat vererek gecikmelerin iyileştirilmesini sağlayacaktır. Birbirinin kopyası olarak haritalanmış iki sütun daima aynı anda iletimde olacağından, tek başına bir sütunun sebep olacağı gecikmeden çok daha düşük bir gecikme değeri elde edilebilir. Buradaki problem hangi sütunların ne kadar kopyalanacağıdır. Öyle ki, her bir sütun daha fazla alan kullanımını ve güç tüketimini ifade etmektedir. Bu noktada, üretim yönteminin ortalama değer ve standart sapma parametrelerinin bilinmesi hedef bir performans değeri elde edilmesi için yeterlidir. Her bir fonksiyon sütununun çıkış performansı, o sütunun sahip olduğu ‘1’ sayısına ve üretim yöntemi parametrelerine bağlı olarak bir Normal dağılım olarak ifade edilebilir (Normal dağılımların toplamı). En fazla sayıda ‘1’e sahip fonksiyon matrisi sütununun toplam ortalama değeri, hiç varyans olmasaydı elde edilecek performansa denk geldiğinden bir performans parametresi olarak alınabilir. Öte yandan, bu ortalama değerin $3\times$ standart sapma aşağısı %99.7 ihtimalle alacağı en düşük değeri ifade eder. Bu değer, bir arama algoritması kullanıldığında elde edilebilecek en iyi performansı ifade ettiğinden diğer bir performans parametresi olarak kullanılabilir. Bu yöntemin en büyük avantajı her bir kesişim noktası için gecikme testinin yapılmasına gerek olmaması ve herhangi bir arama algoritması kullanılmamasıdır. Fakat, alan ve güç tüketimi artışı bu yöntemin dezavantajıdır. Bazı matrislerde alan ve güç tüketimi istenilen parametrelere ulaşmak için %100 seviyelerine çıkarken, bazı bençmarklarda sadece %5 alan artışı yeterli olabilmektedir. Fonksiyon matrisi indirgeme yöntemine benzer olarak, sütunlarda bulunan ‘1’ sayısının saçınık olması bu yöntemden daha fazla verim alınmasını sağlamaktadır.

Sonraki çalışmalarda bu algoritmaların farklı performans parametrelerine göre düzenlenerek çok-amaçlı hale getirilmesi sağlanması planlanmaktadır. Ayrıca verilen arama algoritmasında iyileştirmeler yapılması ve yeni yaklaşımsal yöntemler eklenmesi ile daha iyi gecikme performansı sonuçlarına ulaşılması sağlanabilir. Ayrıca

tüm bu yöntemler diyot ve 4-uçlu mantıksal tasarım yaklaşımları için benzer olarak yeniden oluşturulabilir.

1. INTRODUCTION

In the last decade, developments in nanofabrication make it possible to solve longstanding integration and miniaturization problems of CMOS circuits [1]. A switching component, usually as a field-effect transistor (FET), has been efficiently built with crossed carbon nanotubes or silicon nanowires [2]. These developments lead to programmable circuit architectures based on nano-crossbar arrays which operate similarly to conventional programmable logic arrays (PLA's) [3], [4], [5], and [6], passive crossbar arrays [7], and resistive crossbar logic [8]. Two fully operational crossbar implementations as a nanoprocessor and a finite-state machine are shown to be feasible in [9] and [10]. In Figure 1.1 a crosspoint of nano-crossbar array is given. Here, crosspoint can act as a device, regarding to doping type of nanowires or interlayer. FET, diode and 4-terminal devices and their delay models are also explained in Figure 1.1. In our work we focus on FET type nano-crossbar arrays since they are more popular on literature and manufactured logic devices. In Figure 1.2 a general nano-crossbar array structure is given.

Structures of a conventional PLA and a nano-crossbar array are given in Figure 1.3. Each crosspoint of a nano-crossbar can be configured to operate as a switching device or a FET [11]. Reprogrammability of crosspoints provides a PLA type operation on these arrays. A logic function can implemented with properly placed devices on AND/OR planes along with the corresponding input literals, for both conventional PLA and nano-crossbar architectures.

As area and power efficient structures, nano-crossbar arrays are generally realized with self-assembly based bottom-up fabrication methods as opposed to relatively costly traditional top-down lithography techniques [12]. This advantage comes with a price: very high process variations, especially for delay values in logic computations [13], [6] . Motivated by this, we focus on the worst-case delay optimization problem in the presence of high process variations.

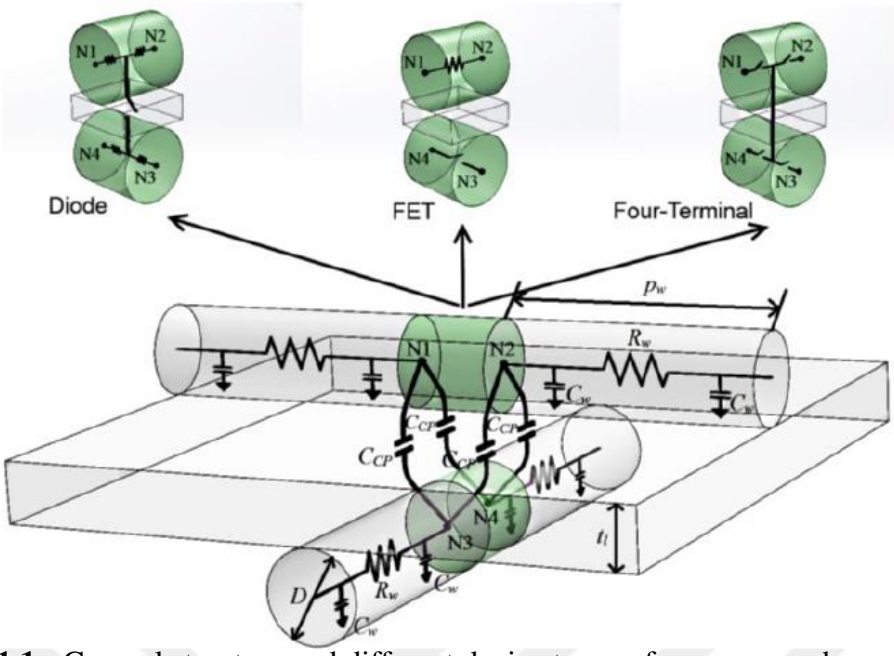


Figure 1.1 : General structure and different device types of a nano-crossbar array and delay model.

Consider a function $f = x_1x_2 + x_2x_3 + x_1x_3$ to be implemented on a 3×3 nano-crossbar array having 3 output lines for 3 products and 3 input lines for 3 literals. Suppose that delay of each crosspoint varies between d and $10d$ where d is a minimum delay time. Here, 6 of 9 crosspoints should be configured as FET's, 2 on each product line, and the rest of them are configured as disconnected lines. Selection of these 6 crosspoints plays an important role for the worst-case delay. There are total of $3! \times 3! = 36$ options to select as the number of orderings of input and output lines, and each selection gives a different delay value between $2d$ or $20d$, since there are 2 crosspoints on each product line. If we have a chance to measure delay contribution of each crosspoint, then by trying these 36 options we can find the best solution. However, with an increase in the number of input lines (n) and output lines (m), the number of options $n! \times m!$ quickly grows beyond practical limits.

In the literature, variation tolerance techniques are generally named under variation tolerant logic mapping problem (VTLM) for different types of nanoarrays. These methods aim to find a best/acceptable logic mapping of a function matrix on a crossbar matrix, considering desired operating performances like delay or power. Different logic mappings mean using different crosspoints/elements on the presumably manufactured crossbar array. Due to variation effects, each array element might have different electrical characteristics. As a result, any different logic mapping

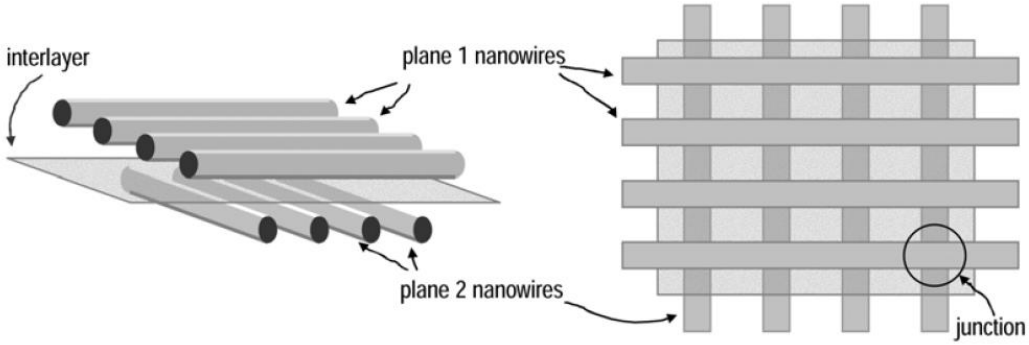


Figure 1.2 : General structure of nano-crossbar arrays.

would have a different operating performance. Here, for a given function matrix, any rows or columns can be switched without changing the functionality. With this interchangeability property, given column number as n and row number as m, there is $n! \times m!$ different mappings for any function matrix on the same sized crossbar array. After size of only 8 x 8 arrays, it is not feasible to use exhaustive search to find the best mapping for desired delay or power performances. This specifications of the variation tolerant logic mapping for $n! \times m!$ arrays make it a $NP-Complete$ problem [14]. By exploiting interchangeability property of the function matrix, different algorithms and approaches developed to find an acceptable row/column mapping for a desired delay or power performances.

In this study, we propose a hill climbing algorithm for the VTLM problem with delay improvement up to 15% and runtime up to 150-400 times better compared to the recent methods in the literature. We comment that the whole crossbar is not necessarily needed for the worst-case delay optimization problem, so our algorithm first performs a reducing operation for the crossbar. This significantly decreases the computational load of the algorithm, up to 72% for our standard benchmark set. Also we explained a greedy column mapping to get a better starting point for hill climbing algorithm.

Since our hill climbing algorithm tries to eliminate crosspoints with higher delays to achieve better worst-case delay results, using crosspoints with relatively high $(\times 100)$ delays as a representation of defects on array allows our algorithm to address both defect and variation tolerance problems. We explain this defect tolerant run mode and results in our work. Note that we can not implement our matrix reducing method for both variation and defect tolerant search since using only one defective crosspoint means an unsuccessful mapping.

As an extension to our work, a worst-case column replication method is given. Here, we use probabilistic data from process specifications and determine columns to be replicated to achieve desired delay outputs. This method is a different approach than generic variaton tolerant logic mapping methods since we do not need any delay data from the manufactured matrix, therefore we do not need any delay testing, which is a costly and time consuming step. Note that we use area overhead while replicating columns, therefore there is a trade-off between circuit area and manufacturing cost, along with the power consumption due to paralleling resistances while achieving lower delays. Also we do not run any recursive or exhaustive search algorithms for this method, which is an another advantage over variation tolerant logic mapping search methods.

1.1 Previous Works

Although defect/fault tolerant logic mapping for nano-crossbars has been long studied, research on variation tolerance is relatively new. First, Gojman and Dehon consider variations on crosspoint transistor parameters to accurately determine the placement of defects as opposed to using randomly assigned defect maps [15]. They propose a post fabrication mapping algorithm (VMATCH) to tolerate defects caused by variations on threshold voltage values of crosspoint FET's. By using independent Gaussian distributions, they determine defects such that when an ON resistance of a crosspoint FET is larger than the OFF resistance, the corresponding crosspoint is defective. As a result, this work can be considered as a transition between variation and defect tolerance methods. However, it does not directly focus on variation tolerant performance optimization.

There are also studies aiming to tolerate both defects and variances; Ghavami et al. propose a greedy algorithm for this purpose [16]. They define the problem as an isomorphism problem between crossbar and function graphs. The algorithm suffers from high runtime values, upto $\times 1000$ larger than those of our algorithm. Also the optimized worst-case delay values are not quite satisfactory, 8-25% larger than the optimal solutions compared to 3-10% for our algorithm. In a later work, weighted sub-graph isomorphism concept is used to define this problem and a greedy algorithm

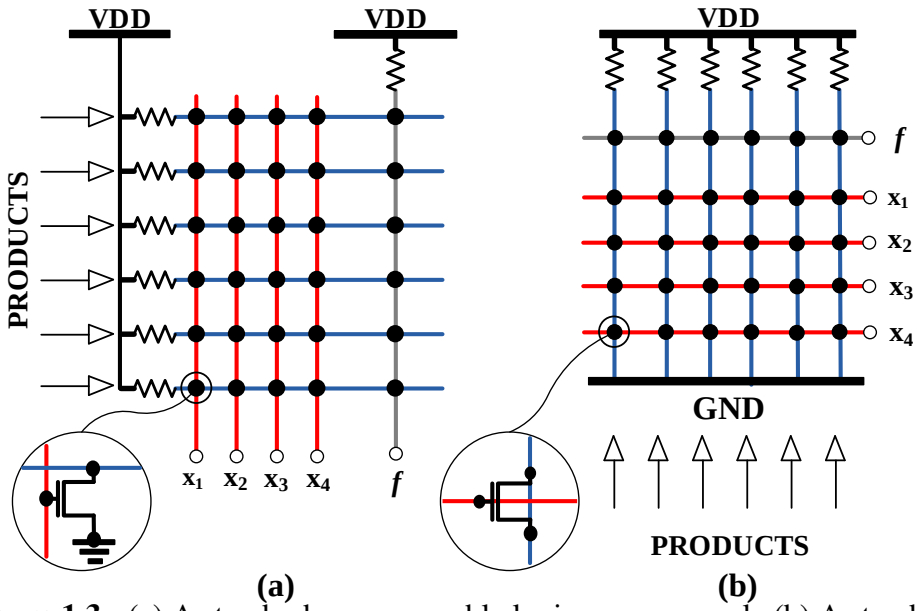


Figure 1.3 : (a) A standard programmable logic array example (b) A standard nanocrossbar array with pseudo-nmos logic. Each crosspoint is a possible nmos device shown in detail view.

is proposed to solve it [17]. While offering satisfactory performance outputs, in our case delays, the algorithm still suffers from high runtimes .

As a complete variation tolerance methodology, Tunc and Tahoori propose a logic mapping algorithm based on simulated annealing [18]. They also explain a delay testing technique on nanocrossbar arrays to obtain delay contributions of crosspoints that is needed for constructing a variation matrix. Although this algorithm offers better runtime and success rate values compared to the above mentioned algorithms, its performance is not satisfactory for relatively large crossbars. Since the algorithm uses randomly selected iterations without progress monitoring, its efficiency is questionable; sufficient results can not be achieved unless relatively high number of trials are reached. On the other hand, our algorithm is designed to make a continuous progress; that is why we call it a *hill climbing algorithm*. Another approach based on linear integer programming is proposed by Zamani et al. [14]. Although satisfactory delay results can be achieved by this systematic method, runtime values are even worse than those of the simulated annealing algorithm. Both simulated annealing and linear integer programming methods are also used to solve defect tolerant logic mapping problem.

Yang et al. propose a different approach for the VTLM problem using a non-dominated sorting genetic algorithm [19]. While finding near Pareto optimal solutions, time

overhead of this algorithm is disadvantageous. In average, runtimes are generally much higher ($\times 30 - 40$) than ours. In order to improve runtimes, Zhong et al. use a greedy re-assignment technique [20], originally proposed in [21]. We also use this re-assignment method with modified parameters for the initial column mapping step of our algorithm that provides us approximately 10% extra time reduction.

Another evolutionary algorithm is proposed by Zhong et al. [22]; it is a bi-level multi-objective optimization algorithm that uses different approaches on row and column mappings defined as lower and upper problem levels. Every individual of an upper level problem is required to be first solved as a lower level problem that puts too much burden on the lower level (row order) algorithm. Therefore the overall algorithm performance mostly depends on the performance of the lower level algorithm, defined as a min–max-weight and min-weight-gap bipartite matching problem being solved by a heuristic variant of the Hungarian method. Upper level (column order) problem is solved by a non-dominated sorting genetic algorithm given in [19]. As a result, this algorithm achieves nearly 70% better delay values compared to the algorithm given in [19]. However, runtimes are still an issue. Our proposed algorithm gives delay values in the same range while having considerably lower runtimes.

In their work, Yuan et. al. [23] proposed a memetic algorithm to solve both defect and variation tolerant logic mapping problems on nano-crossbar arrays. While having high defect tolerance success rates and variation tolerance optimization rates than several algorithm from the literature, like simulated annealing or genetic algorithm, this method have high runtimes, due to its probabilistic nature and running min–max-weight and min-weight-gap bipartite matching problem for each candidate solution. We used this algorithm to compare our results since it is a relatively new and successful algorithm.

There are also studies focusing on adding extra rows or columns for better variance tolerance at the cost of area yield. In their work, Zamani and Tahoori use row redundancies with duplicated input lines [24]. Their approach successfully reduces the critical path delay with an average of 50%. Using area overheads is primary disadvantage of these methods. Amount of redundancy can not be determined after post-fabrication measurements that is a dilemma "before fabrication, the amount of redundancies should be known, but it can be determined after fabrication". Indeed, the

need of crosspoint tests are extensively valid for all of the above mentioned studies including our hill climbing algorithm.

1.1.1 Organization

Organization of this work and our contributions on each section can be summarized as follows.

- We introduce preliminaries for the VTLM problem and define our performance objectives in Section 2.
- We propose our VTLM algorithm in Section 3 with its steps given in the subsections. In these steps, we use a *function matrix reducing*, a *greedy column mapping*, and a *hill climbing* row search with *column reordering* methods.
- We extend our hill climbing algorithm for both defect and variance tolerance (D/VTLM) problem in Section 4.
- We explain our worst-case column replication approach in Section 5.
- Experimental results are given Section 6, and Section 7 concludes this study with insights and future directions.

2. PRELIMINARIES

In variation tolerant logic mapping scheme, we need to know two main factors of the problem. The first one is the Variation Matrix VM and the other one is the Function Matrix FM. The goal of any mapping algorithm is finding the best (or desired) performance by determining the proper row and column orders for FM to be mapped on the VM.

Function Matrix (FM) is a binary matrix which indicates a logic function to be mapped onto nanoarray structure. As a general design topology, horizontal lines are function inputs and outputs while vertical lines are function product lines [11]. If an input parameter is included in a function product, crosspoint of corresponding vertical and horizontal lines are tagged as '1' (device active), otherwise '0' (device passive, short circuit). An example FM and mapping scheme on a nanocrossbar array is given in Figure 2.1.

Function Matrix Parameters: row number m, input mapping vector IMV, column number n, output mapping vector OMV [18] and crosspoint ratio CR for random generated matrices.

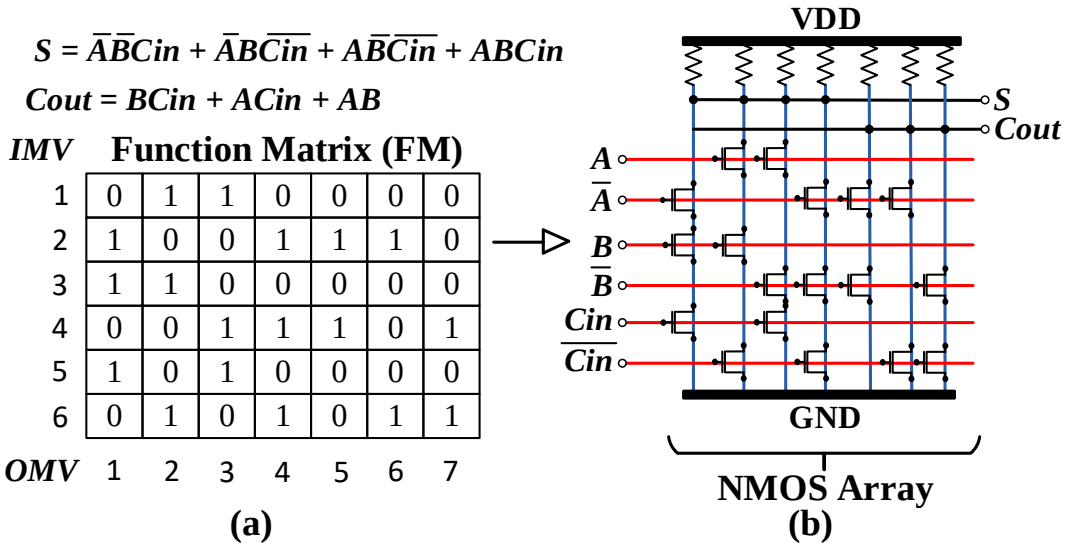
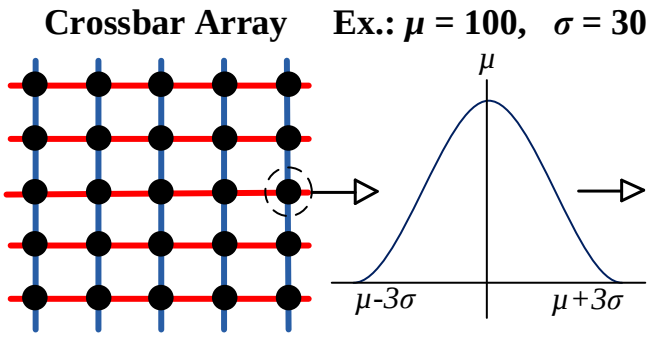


Figure 2.1 : (a) Binary Function Matrix and Output Functions (b) Logic Mapping on a FET type array with pseudo-NMOS logic.



Variance Matrix (VM)

116	59	120	126	96
32	121	119	67	109
108	112	114	11	74
86	96	124	104	95
207	142	108	141	132

Figure 2.2 : Variance Matrice Generation for a 5x5 crossbar array by using Gaussian Distribution.

$$FM(i,j) = \begin{cases} 1, & \text{if input } i \text{ is included in output } j \\ 0, & \text{otherwise} \end{cases}$$

Variance Matrix (VM) consists of switching delay values of each crosspoint. These values are determined by the delay testing of the the nano crossbar [18]. Since these values are affected by various factors (size, dopant atoms, mismatchs [13]), we can consider these values as Gaussian random numbers while simulating any optimization algorithm. An example variance matrix and random delay value generation scheme is given in Figure 2.2.

In our study, we use *Coefficient of Variation (COV)* (σ/μ) as 0.2 and 0.3 to inspect different process variation cases. In calculations we use mean value as 100.

Variation Matrix Parameters: row number m, column number n, expected mean of crosspoint delay μ, standart deviation of crosspoint delay σ.

$$VM(i,j) = \begin{cases} \text{Delay Value of Crosspoint } (j,i) \\ Gaussian(\mu, \sigma^2) \text{ for simulation} \end{cases}$$

Performance Matrix (PM) is the Hadamart Product of FM and VM matrices. This matrix express all used columns and delay values for them. Equation 2.1 express this product.

Since we will focus on FET type arrays on this work, we can define a row array to express all column performances as sum of all rows on a PM, expressed in equation 2.2 as FET Performance Matrix *(FPM)*.

$$i = 1,2,...n, \quad j = 1,2,...m$$

$$(PM)_{j,i} = (FM)_{j,i} \times (VM)_{j,i} \tag{2.1}$$

$$FPM_i = \Sigma_j (PM)_{j,i} \tag{2.2}$$

While inspecting our proposed hill climbing algorithm on defect tolerance simulations, we generate a new VM which has defective crosspoint delays as relatively high $(\times 100)$ from expected value. Generation of this altered VM matrix is given in corresponding section thoroughly.

2.1 Objectives

In the literature, two main objectives are considered while developing algorithms. First; minimizing maximum value of sum of the all columns for Hadamart products of FM and VM matrices. Second; minimizing difference between maximum and minimum sum of the all columns for Hadamart products of matrices [18]. Hadamart product of two matrix means product of same index of each matrix on the result matrix PM, as in equation 2.1. When a crosspoint is activated (mapped as 1), its delay value is added to the corresponding output. Definitions of primary objectives are given in equations 2.3 and 2.5.

$$Objective\ 1 = minimize(maximum(FPM_i)) \tag{2.3}$$

$$Objective\ 2 = maximize(minimum(FPM_i)) \tag{2.4}$$

$$Objective\ 3 = minimize(Objective\ 1\ -\ Objctive\ 2) \tag{2.5}$$

In our work, we focus on $Objective\ 1$ since this performance determines the worst case delay value of given nano crossbar. Also, since we do not using any defective crosspoints, not using a crosspoint with the better performance to have least difference between column outputs $(Objective\ 3)$ is a questionable move for delay optimization. Therefore, we focus any computation power on optimizing worst case delay values of nanocrossbar arrays.

A logic mapping optimization scheme is given in Figure 2.3. Here, we use interchangeability of rows and columns of the function matrix and find a better

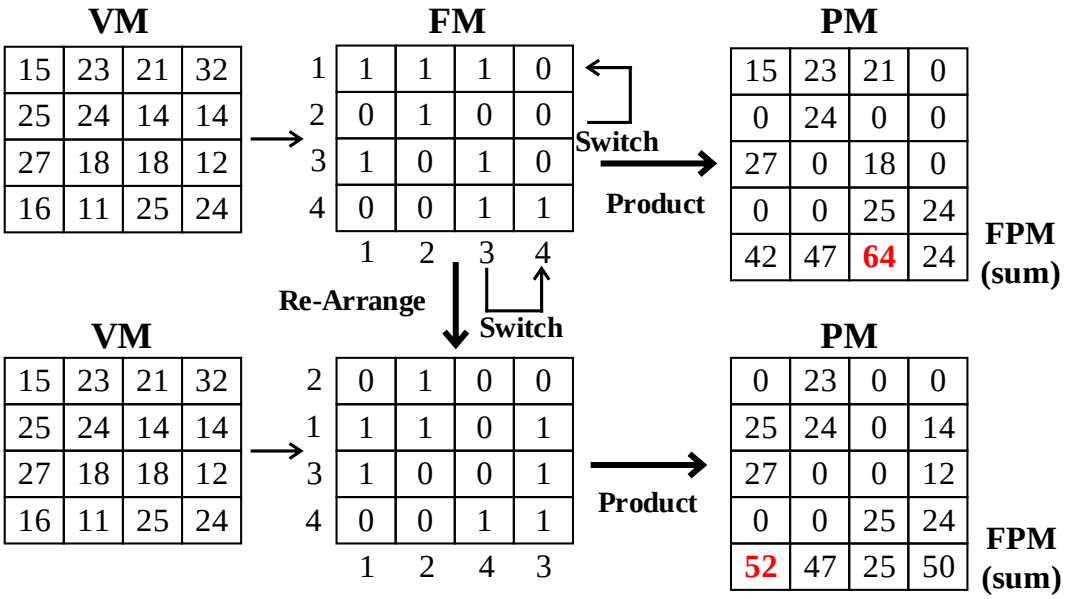


Figure 2.3 : A logic mapping optimization case by switching both rows and columns. Maximum of FPM output is lower in re-arranged Function Matrix FM.

mapping for maximum delay sum value of the all product columns (FPM). In next sections, systematic methods are investigated to achieve optimal mapping results and compared with the methods from the literature.

3. PROPOSED ALGORITHMS ON VARIATION TOLERANCE

In this section we introduce all stages of our Hill Climbing Algorithm. These stages are: function matrix reducing, initial column mapping and hill climbing row search with column re-ordering. In Section 6, we investigate performance of this algorithm thoroughly.

Nature of the logic matching problem gives us two optimization moves, switching rows or columns. Since we calculate $Objective$ 1 by the sum of the PM columns as FPM, changing rows or columns have different effects on the performance output. For a given column map, we can optimize $Objective$ 1 by switching rows precisely and systematically. However, for a given row map, we can not precisely control output performance since each column switch totally changes sums on switched columns, which results as mostly random jumps between performance values. This means, for row search for a fixed column order, we can gradually optimize output performance but can not do vice-versa. As a result of this characteristic of the mapping problem, we define optimization process in two steps as: first the column mapping using matrix data and second the precise tuning by row search. Gradually optimization process is similar to Hill Climbing search paradigm, so we model our row search as this algorithm. As an expansion to this search, unlike random re-starting points used in general Hill Climbing paradigm, we propose a Column Re-ordering algorithm for new starting points by using earlier row search data.

As a pre-processing method for our algorithm, we propose a function matrix (FM) reducing method. With this method, we find FM columns which are unnecessary to compute for our search and reduce this matrix to a performance equivalent form. This approach results as higher computations in unit time for any algorithm.

Later in this section we propose our algorithm to solve the variation tolerant logic mapping problem. First, we apply matrix reducing method to Function Matrix FM and find a smaller performance equivalent matrix. Then, we explain different initial column mapping methods to find a starting point to our Hill Climbing algorithm. After

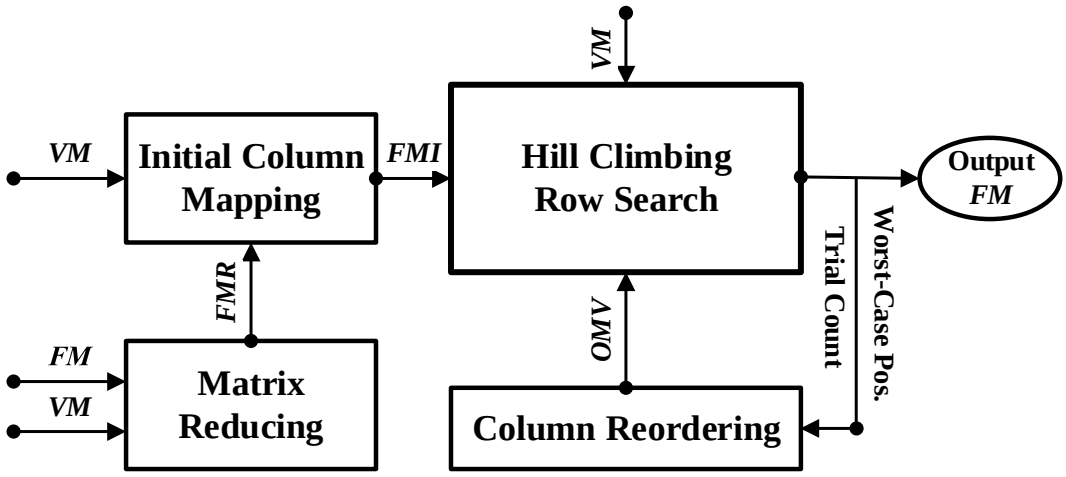


Figure 3.1 : Work-flow of our proposed Hill Climbing with Column Re-Ordering algorithm.

finding a starting point, we explained our Hill Climbing Row Search method with Column Re-ordering. Work flow of the proposed algorithm scheme is given in Figure 3.1.

3.1 Function Matrix Reducing

In the variation tolerant logic mapping problem, performance of the individual logic mappings have to be calculated in order to decide the next move on algorithm or just compare with the other solutions. These calculations can be made more effectively to improve algorithm effectiveness in unit time. Tahori et al. [18] checked only the performances of the switched columns or rows to reduce performance testing times on each individual mapping. However, this method is only suitable when all column performances (FPM) are known in earlier mapping. Moreover, even in this method, some column performances are calculating over and over again with no effect expectations on the performance output $Objective$ 1. Since we only interested in the maximum value of all the column performances, columns with the lower transistor counts are likely to be never become the worst case performance on an array, so there is no need to calculate performance for these columns in any mapping.

An example is given in Figure 3.2. Here, columns reduced for increasing upper transistor count limits. For each case, reduced matrix and standard matrix search performance outputs and timings compared. Until 90% of maximum transistor count, there is very low or no difference between performance outputs. After this limit, there

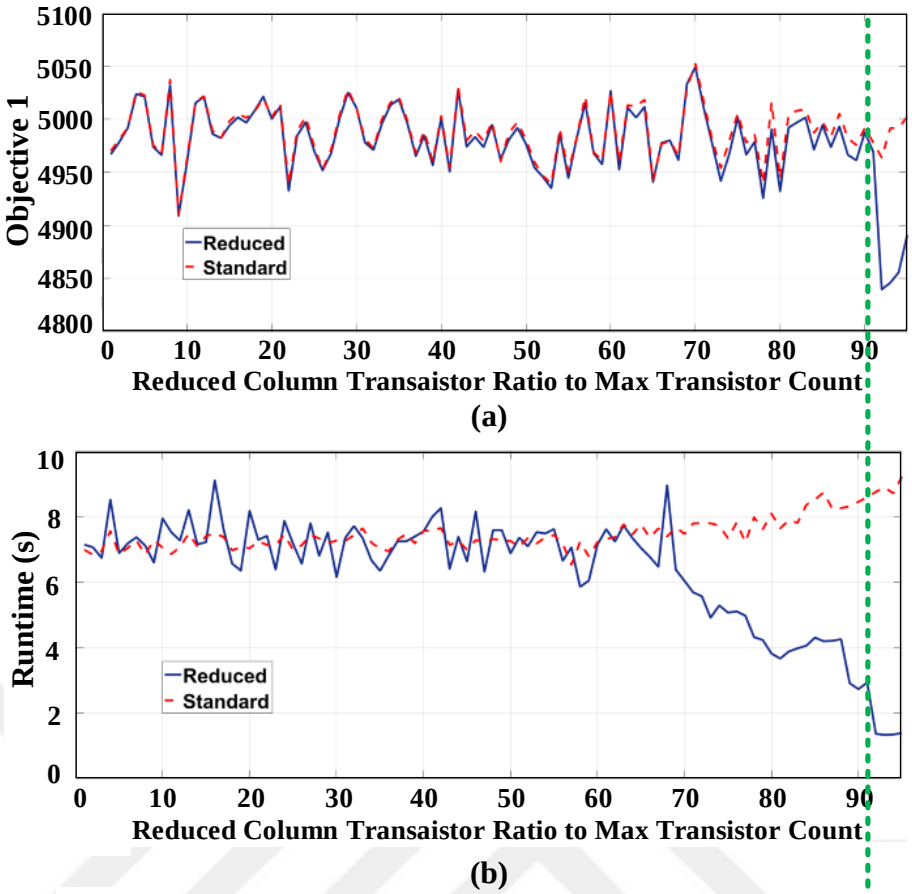


Figure 3.2 : (a) Objective 1 difference between initial and reduced matrices, using algorithm outputs regarding to reduced matrix and (b) Time comparison for standard and reduced matrices as the reducing increases.

is high difference between these values since we do not consider a critical column on reduced search, therefore achieved better results which are not correct. However, reduced matrix search times getting better for each column reduction. So, if we reduce this example from 90% limit, we can have 50% time improvement on our search algorithm without algorithm output correctness. Note that this limit is unique for each FM and VM or process parameters μ and σ.

Here, we propose a preliminary method to find this certain transistor count for a given FM and VM.

3.1.1 FM reducing method

Since we know both FM and VM, we can find all the possible outcomes for each FM column. Possible outcome count is dependent on the transistor count T_i for each column, as given in the equation 3.1 as $CCount$, which is not feasible to calculate for larger arrays. However, we can easily calculate maximum and minimum possible

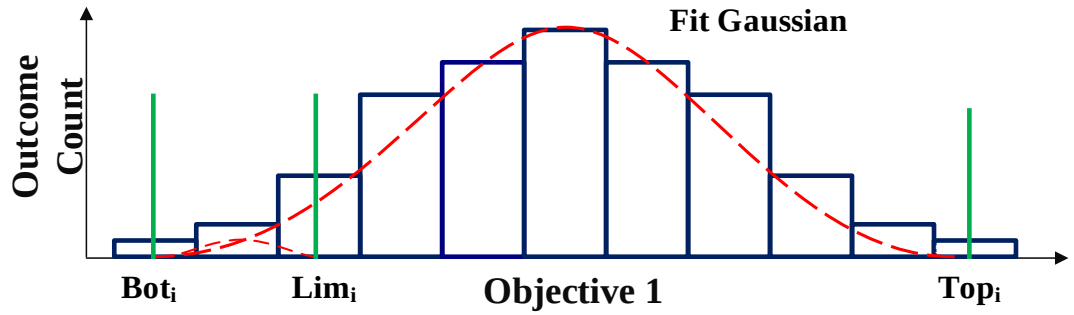


Figure 3.3 : Top_i, Bot_i and Lim_i limit values for ith column of FM, and fitted Gaussian distribution considering histogram of all possible outcomes.

outcomes for each FM column, as given in equations 3.4, 3.5. Due to *Gaussian* model of the VM generation, histogram of all possible outcomes would fit a *Gaussian* distribution. So, we will show these histograms as fitted *Gaussian* curves with upper and lower limits as Top_i and Bot_i in Figure 3.3.

We know that column with the maximum transistor count T_{max} exist in performance calculations. So, if maximum possible outcome of a FM column is lower than the minimum possible outcome of column with the T_{max}, we can consider this column as absolutely redundant to calculate for $Objective$ 1. However, we can find more redundant to calculate columns by considering possible errors with low probability. For this, we find span of the lowest outcomes for T_{max} on each VM column and use maximum of these outcomes as new upper limit as Lim_i. A Reduction example is given in Figure 3.4 and Algorithm in Figure 3.5 explains this process.

Note that this reducing method can be modified for different $Objective$ types. Here, we reduced FM while considering only $Objective$ 1 optimization.

This method can be implement onto any variation tolerant logic mapping algorithm. Considering our implementation for $Objective$ 1, we used different algorithms and benchmarks in the Section 6 and compared timing and correctness results.

$$i, j = 1, 2, ..., n \quad n : column\ count, m : row\ count$$

$$T_i : Transistor\ count\ of\ ith\ column$$

$$CCount_i = \frac{m!}{T_i! \times (m - T_i)!}\ x\ n \tag{3.1}$$

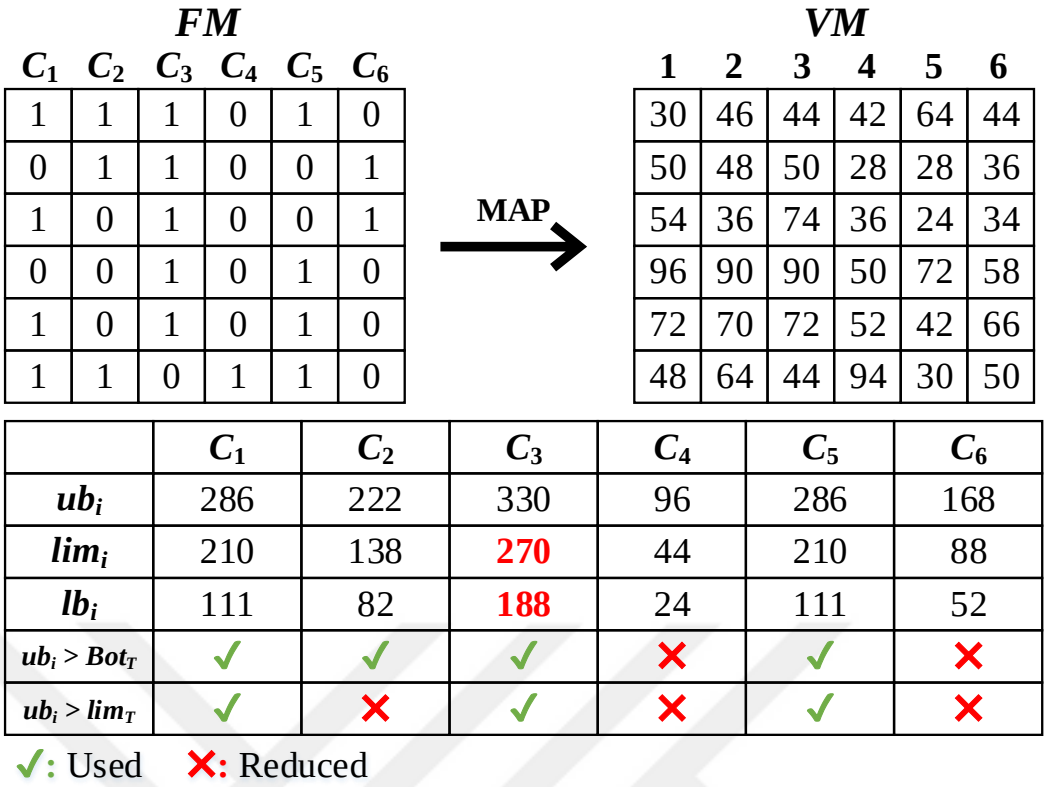


Figure 3.4 : A matrix reducing example. When lb_{max} is used as limit C_4 and C_6 are redundant. When lim_{max} is used as limit C_4, C_6 and C_2 are redundant.

$$VMHL(:,i) = SORT_{HIGHtoLOW}(VM(:,i)) \tag{3.2}$$

$$VMLH(:,i) = SORT_{LOWtoHIGH}(VM(:,i)) \tag{3.3}$$

$$Top_i = Maximum(\sum_{k=1}^{T_i} VMHL(k,j)) \tag{3.4}$$

$$Bot_i = Minimum(\sum_{k=1}^{T_i} VMLH(k,j)) \tag{3.5}$$

3.2 Initial Column Mapping

On the column mapping process, we have $m!$ different row positions for each different column map (OMV order). So, we need to use values of FM and VM to find a column mapping to start searching rows. In [20], a greedy mapping algorithm is given by using total delay values for each VM column. This method places FM columns with the higher transistor count onto VM columns with the lower delay sum values, in

```
1: Input: FM_{m×n}, VM_{m×n} and tolerance ratio t
2: Output: FMR_{m×n}                                   ▷ new Function Matrix
3: FMR ← FM
4: for each column i do
5:     FMr[i] ← Sum of FM[i] column
6: end for
7: T_max ← Maximum Transistor Count of All Columns
8: FMs ← Sort Low to High FMr
9: for each column i do
10:    T_i ← Transistor Count of FMs[i]
11:    for each column j do
12:        PMax[i][j] ← Sum of the highest T_i values on VM[j]
13:        if T_i = T_max then
14:            Pmin[j] ← Sum of the lowest T_max values on VM[j]
15:        end if
16:    end for
17: end for
18: myLimit ← Minimum Value of Pmin
19: k ← c
20: for each column i do
21:    if Maximum of row PMax[i] < myLimit × t then
22:        Reds[end+1] ← FMs[i]
23:    end if
24: end for
25: for each column i do
26:    for each index of Reds j do
27:        if FMr[i] = Reds[j] then
28:            FMR[][i] ← null                          ▷ Column Reducing
29:            break
30:        end if
31:    end for
32: end for
```

Figure 3.5 : Algorithm 1 as Matrix Reducing by Variation Matrix Values (Absolute Redundant).

high-to-low transistor count order. Since most of the times this mapping scheme resuls as a better solution space, it is reasonable to use this algorithm for a starting column mapping point for a search. Bo Yuan et al. [22] used this greedy column mapping on their multi-level genetic algorithm and gathered better results as explained in [20].

As a different approach to this greedy mapping [20], we use possible minimum values for each FM column on each VM column. This mapping algorithm is explained in Figure 3.6 as Algorithm 2. Here, columns with the highest to lowest transistor count mapped to the VM column which gives least possible solution of all available VM columns, calculated by considering each FM column independent. After each individual column mapping, used VM column is not available anymore. A greedy

Input: $FM_{m\times n}$, $VM_{m\times n}$, column count n
Output: $FMI_{m\times n}$
for each column i **do**
    FMr[2][i] $\leftarrow$ sum of ith column of FM
    FMr[1][i] $\leftarrow i$
end for
FMs $\leftarrow$ High to Low sort of FMr Regarding to FMr[2]
for each column i **do**
    $T_i \leftarrow$ FMs[2][i]
    for each column j **do**
        W[i][j] $\leftarrow$ Sum of the lowest T_i values of VM[j]
    end for
end for
for each column k **do**
    [perf][pos] $\leftarrow$ Value and Position Index for the Minimum of kth row of W
    OMV[FMs[1][k]] $\leftarrow$ pos
    pos column of W $\leftarrow$ $null$
end for
for each column i **do**
    ith column of FMI $\leftarrow$ $(OMV[i])th$ column of FM
end for

Figure 3.6 : As Proposed Algorithm 2: Greedy Column Mapping Regarding to the Possible Minimum Outcomes.

mapping example is given in Figure 3.7. Performance comparison of these initial orderings given in Section 6.

3.3 Hill Climbing Row Search

This section gives the main processing part of our algorithm propose. After determining a starting column map, our algorithm starts its search by finding the column with the worst delay value, $Objective$ 1. Then, focuses on the optimizing that worst case column and check performance of whole matrix, until a better performance is achieved. This column optimization is consists of changing higher delay valued '1's rows with the lower delay valued '0's in order to achieve maximum optimization for focused column. After finding a row order with better performance, new performance is compared with the reference performance value. If reference performance value is reached, algorithm stops and saves current column and row mappings. Else, algorithm starts to try optimizing the new worst case column and keeps the earlier worst case column index in a separate array. This column information will be used in column

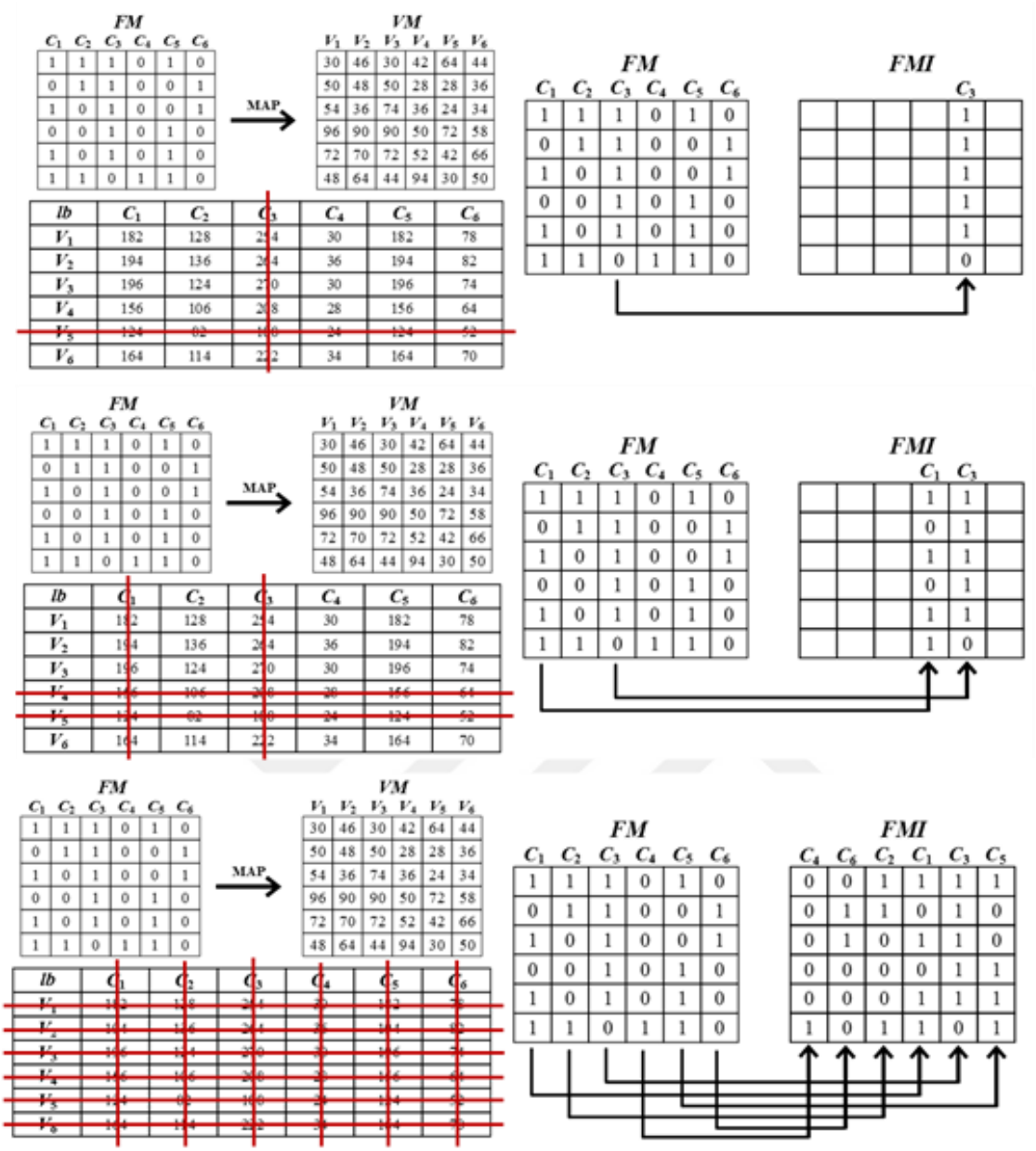


Figure 3.7 : An initial column mapping example for a 6×6 matrix. First, second and last steps are given.

re-mapping process. If there is no row switch to get a better performance from the mapping, algorithm does a column re-mapping and starts over while recording current performance value and input row order *IMV* and output column order *OMV*. This search process is given in Figure 3.8 as Algorithm 3.

3.4 Column Re-Ordering

After reaching an un-optimizable point, algorithm checks which column have the worst case delay value most of the times on row search process. Then this column is switched

```
Input: IMV, OMV, FM, VM, column count n, row count m
Output: D_final, OMV_final, IMV_final
t ← 0
while t < n do
    next ← 0
    D_old ← worst-case delay for IMV, OMV, VM set
    P_old ← D_old column position
    while next ≠ 1 do
        P_ones ← '1' positions on P_old column, high to low
        P_zeros ← '0' positions on P_old column, low to high
        for each index of P_ones i do
            for each index of P_zeros j do
                IMV_cur ← switch i and j on IMV
                D_cur ← worst-case delay for IMV_cur
                P_cur ← D_cur column position
                if D_cur < D_my then
                    D_old ← D_cur
                    P_old ← P_cur
                    IMV ← IMV_cur
                    add P_cur to P_data matrix
                    break for loops
                end if
            end for
        end for
        if D_cur > D_old then
            next ← 1
        end if
    end while
    add D_old to D_data matrix
    add OMV to OMV_data matrix
    add IMV to IMV_data matrix
    t ← t + 1
    if t ≠ n then                                   ▷ Column Reordering
        C_wc ← the most repeated column on P_data
        M ← mean '1' count on FM column sums
        C_sorted ← column low to high order for '1' sums
        OMV ← switch C_wc and C_sorted(mod(t, M))
    end if
end while
D_final ← minimum of D_data
P_f ← position of D_final on D_data
OMV_final ← P_f th index of OMV_data
IMV_final ← P_f th index of IMV_data
```

Figure 3.8 : As Algorithm 3, Hill Climbing Row Search Algorithm with Column Re-Ordering.

with a column with low transistor count. $1/3$ lowest part of the all FM columns is used repeatedly to avoid placing high transistor count to the determined worst case column. This ratio can be adjusted specifically for benchmarks. Column Re-Ordering algorithm is given in Figure 3.8 in Algorithm 3 as a part of our complete hill climbing algorithm.

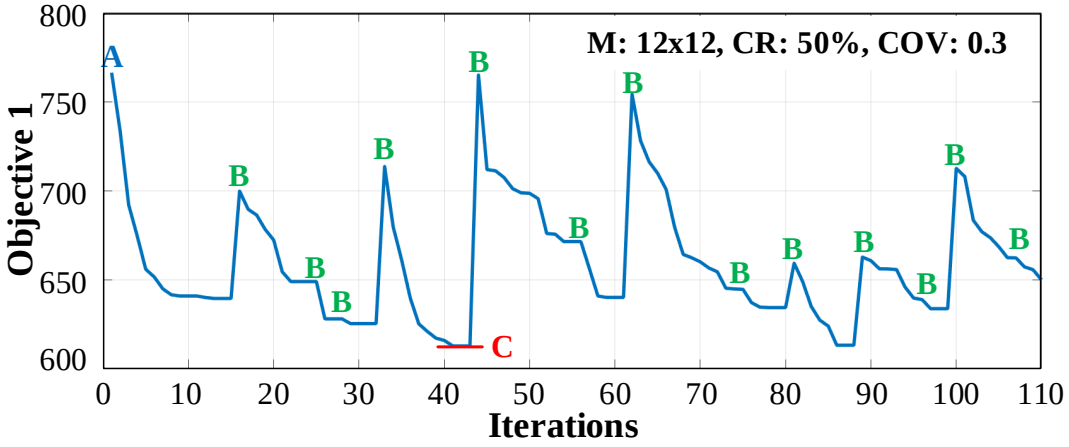


Figure 3.9 : Process of our proposed algorithm. Each spike in the performance value is a column re-mapping process.

3.5 Algorithm Work-Flow

At first, we apply function matrix reducing method to find a smaller performance equivalent matrix. After this, an initial mapping is determined by the proposed greedy column mapping algorithm, which is given in Figure 3.6 as Algorithm 2.

Knowing our starting point, we start Hill Climbing search given in Figure 3.8 as Algorithm 3. After reaching an un-optimizable state, column re-ordering algorithm works with the information gathered by Hill Climbing process if algorithm stop criteria are not met. Then, Hill Climbing works again with new column ordering.

There is one criteria for our algorithm to stop its search. If column re-mapping count is reached column count n, algorithm stops and gives best result from earlier recorded results. Algorithm gives the lowest recorded *Objective 1* value and corresponding *IMV, OMV* orderings as outputs.

Figure 3.9 is an example process of our algorithm for a 12x12 random array. Here point 'A' is initial mapped performance. Each downward slope is Hill Climbing Algorithm on the process. After reaching an un-optimizable point, a column re-mapping is occurred, as in all 'B' points. After all re-mappings done, the lowest un-optimizable point 'C' is selected as the best solution. Note that this is an anytime algorithm. And as an extension, we can implement this method to diode or 4-terminal based nanocrossbar arrays.

Run-times and accuracy rates for our algorithm is given in Section 6.

4. EXTENSION OF PROPOSED HILL CLIMBING ALGORITHM ON DEFECT TOLERANCE

As an extension for our proposed hill climbing algorithm, we implement $VTLM$ problem as defect tolerant variation $(DTLM)$ problem by using infinite delay values for defective crosspoints. Due to algorithm flow, crosspoints with infinite delay are avoided, therefore defects are avoided.

For this work, first we define a new variance matrix (VM) with defects. For this purpose, we use random defect placements on VM as infinite delays. As a supportive matrix, we define a defect matrix (DM) where defective crosspoints marked as '1' and non-defective crosspoints marked as '0'. Figure 4.1 explains this generation process. For our algorithm to work, we use 100μ delays on defective crosspoints to easily inspect algorithm variation toleration flow on defective cases.

Work flow of hill climbing algorithm is given in Figure 4.2. We used different reorderings regarding to columns with the most defective crosspoints, since they are hardest ones to map.

Since column with the most defective points acts as the column with the worst-case delay, algorithm tries to optimize this column, for every different column reordering cases. If algorithm finds a non-defective mapping on a single column ordering, it does not accept any defective mapping since it would have a much higher delay value. Three examples are given in Figure 4.3. Here, when there is no defect algorithm works as standard. When there is defects, there might be successful mappings or not. In the second case, there is 15% defect rate. Here, any mapping above red line $(100 \times \mu)$ is defective anyone below is not defective. After finding a non-defective mapping, algorithm still tries to reach row orderings with the better delay values. In the last case, there is 30% defect rate, therefore algorithm couldn't find any non-defective mappings, since all the solutions are above red line $(100 \times \mu)$. While searching, if algorithm can find a successful mapping, it do not accept any defective mappings and finds only

VM

V_1	V_2	V_3	V_4	V_5	V_6
30	46	30	42	64	44
50	48	50	28	28	36
54	36	74	36	24	34
96	90	90	50	72	58
72	70	72	52	42	66
48	64	44	94	30	50

→

VM

V_1	V_2	V_3	V_4	V_5	V_6
∞	46	30	∞	64	44
50	48	∞	28	∞	36
54	∞	74	36	24	∞
96	∞	90	∞	72	58
∞	70	72	52	∞	66
48	64	∞	94	30	∞

→

DM

D_1	D_2	D_3	D_4	D_5	D_6
1	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	0	1	0	0
1	0	0	0	1	0
0	0	1	0	0	1

Figure 4.1 : Adding defects on to *VM* and generating *DM* from this new *VM*.

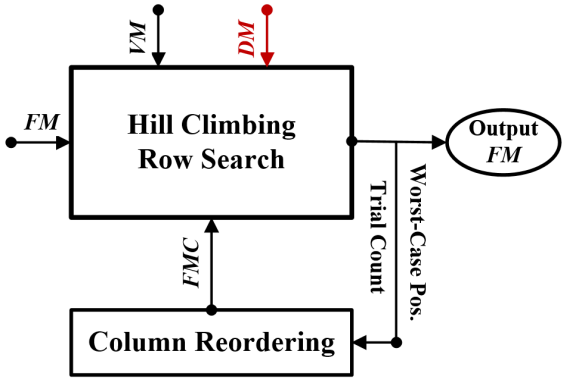


Figure 4.2 : Work-flow of hill climbing algorithm for defect tolerance. A *DM* is added to the algorithm.

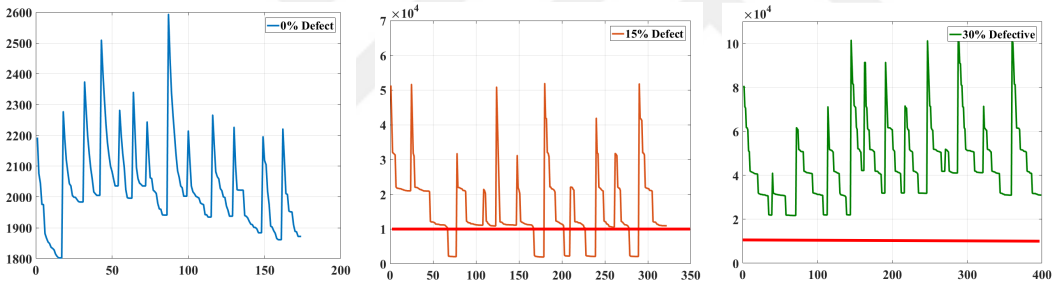


Figure 4.3 : Three examples for hill climbing algorithm with different defect rates. While the first graph has no defects and works as normal, other graphs have different defect rates. If algorithm achieve a successful non-defective mapping, there is delay values below red lines as limits.

mappings with the better delays. Therefore, proposed hill climbing algorithm finds a successful mapping and an optimized delay value.

Here, we use *DM* as a confirmation matrix. For a certain *FM* column and row ordering we check whole matrix sum of Hadamart product of this *FM* and *DM*. If this sum is '0', it means there is no defective crosspoints are used (there is no '1' on '1' matching on these two matrices) and we have a successful mapping. Otherwise it is a defective mapping. We use this confirmation for each different row and column orderings to move on to next iteration on our algorithm.

Simulation results for defect tolerance is given in Section 6.

5. COLUMN REPLICATION APPROACH

In this section of our work, we approach variation tolerance in a different mapping paradigm. Here, we use area overhead on *VM* to use worst case route replication similar to works on literature [25]. One main consideration here is; variation of crosspoint delays are mainly cause of the variation on the resistance values. It is expected to have variations in inner capacitance values of nanoarray but we can neglect these capacitances because of the small values and possible much larger output load capacitances [26].

In FET type nanoarrays, each column is a logic function product line, so we focus on replicating the columns. This approach also means a power overhead related to the replicated columns. However, since we do not need to know exact delay contributions of each crosspoint, there is no need to use delay testing on a fabricated nanoarray [18]. Also, we can not use any variation tolerant logic mapping algorithm in this mapping, since we do not know any crosspoint delay values, which is other main advantage of this approach.

Here, we determine a performance limitation to find columns to be replicated. After that, replication process is explained and lastly area and power overhead calculation methods are given. Results for different sized arrays are given and explained in Section 6.

5.1 Determining a Goal Delay Performance

While determining a performance limit, we can consider the expected maximum delay value if there is no variation. This straightforward calculation is given in equation 5.1. But it is expected to have better values since optimization algorithms focuses to use crosspoints with lower delay values. So, we can estimate an optimized performance goal to have an optimized array equivalent in expense of more area overhead. As a general consideration, worst case value of a Gaussian Distribution is 3 times standard deviation distance from mean on both upper and lower values. This consideration

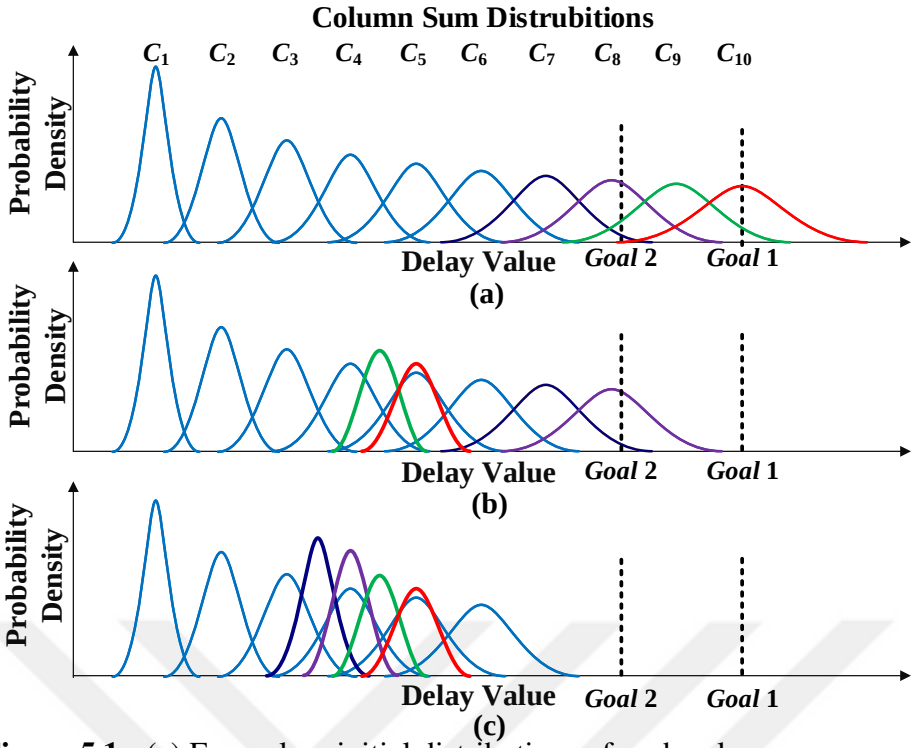


Figure 5.1 : (a) Exemplary initial distributions of each column on an array considering random mapping and Limit positions (b) Replicating columns until worst case values of all columns are under *Goal* 1 (c) Same procedure for *Goal*2.

gives 99.7% accurate outputs. So, we consider optimized performance value as the best value for the worst column as equation 5.2. We use this two goal parameters and different coefficient of variations to compare needed area overheads for different cases.

$$Goal\ 1 = T_{max}\mu \tag{5.1}$$

$$Goal\ 2 = T_{max}\mu - 3\sqrt{T_{max}\sigma^2} \tag{5.2}$$

5.2 Worst Case Route Replication

This approach takes each columns and compares its expected worst case delay with determined *Goal* [25]. If worst case value is more than defined limit value, a replicate of this column is added to the array. After replication, we have 2 columns with same transistor positions and same Gaussian Distribution parameters. Here, to avoid Gaussian products, we focused on the worst possible cases depending on the 3σ distances. So, replication of a column makes its worst case delay value half of its initial worst case value. If only one replication did not make the worst case delay

value better than limit value, another replication could be added. This makes worst case value $1/3$ of its initial worst case value. Replication of any column is continued until getting a better performance than defined limit. Still, maximum replication limit is selected as 5 in our calculations.

5.3 Power Overhead Calculation

Considering each column to be equally likely to be activated, we can calculate average power overhead. Since when a column is replicated, its power consumption is expected to be doubled. We can consider power consumption values for each column by using equations 5.3, 5.4, where K is process constant for actual resistance values and power supply. Here, more transistor count results as higher resistance values, so there would be lower power consumption for higher transistor count.

$$i = 1,2...n \qquad n : column\ number$$

$$T_i : Transistor\ Count\ of\ ith\ Column$$

$$P_i = K/T_i \tag{5.3}$$

$$P_{AVG} = \frac{1}{n}\sum_{i=1}^{n} K/T_i \tag{5.4}$$

Rather than using a performance goal as equation 5.1 and equation 5.2, we can inspect power - delay products of the given arrays regarding to decreasing *Goal* values. Since there is a trade-off between delay and power, we can find a better product point than equation 5.1 and equation 5.2 goals. While using average power, as in equation 5.4, we also use average delay rather than worst case delay determined by the selected goal. This approach would give us more fair data about general operation performance.

Inspection of power - delay product and area overheads for arrays with the different size are given in Section 6.

6. SIMULATION RESULTS

In this section we give simulation results for each proposed method. First, we give algorithms from literature to compare with our proposed algorithm. Second, we simulate our algorithm for different comparison cases on Table 6.2, Table 6.3 and Figure 6.1. Then, we present matrix reducing performance optimization for different algorithms on Table 6.1. Next, we inspect defect tolerance runs of our hill climbing algorithm on Figure 6.2, Figure 6.3, Table 6.4 and Table 6.5. Lastly, we give power and area overheads for column replication method while not using any search algorithm or delay testing on the matrix in Figure 6.4 and Table 6.5. All these simulations are explained in corresponding subsections.

6.1 Algorithms to Compare

For the comparison of our algorithm we used 3 different methods. We will explain these briefly in next subsections.

6.1.1 Random solution generation

It is an important attribute for an algorithm to perform better than random generated solutions. For this comparison, we generated increasing numbers of IMV and OMV permutations, until $Objective$ 1 values saturate.

6.1.2 Simulated annealing

As a application of generally acknowledged method, Simulated Annealing is proposed by Tahoori el al. [18] as a effective and fast method for variance tolerant mapping problem for FET and Diode type nanocrossbar arrays. In our work, we re-coded this approach and achieved similar results as output performance and timings.

6.1.3 Genetic algorithm

In their work Bo Yuan et al. [22] used a multi-objective non-dominated sorting genetic algorithm to achieve optimized results for both $Objective$ 1 and $Objective$ 2. Unlike their work, we applied a standard genetic algorithm scheme to obtain a probabilistic algorithm for comparison with our algorithm for only $Objective$ 1. This algorithm is more simple than genetic algorithm proposals in the literature, but since focuses on only $Objective$ 1, it has relatively low run times for similar population and iteration characteristics.

6.1.4 Memetic Algortihm

As an extended genetic algorithm, we used work of B. Yuan et al. [23]. This algortihm result best delay and defect optimization results from other genetic algorithms or simulated annealing algorithms. Note that this algorithm is specifically generated for both defect and variance tolerant logic mapping problem.

6.2 Simulation for Reduced Function Matrices

As a separate inspection for matrix reduction simulations, we use different benchmarks and algorithm types in Table 6.1. Each algorithm finds a solution for both using reduced matrix and standard matrix and compares $Objective$ 1 performance results along with the timings. Here in our reduction scheme, there is always under %4 percent output error from standard run of the algorithm while time optimization for each algorithm changes considerably by matrix reduction ratio.

Note that these simulations made with upper limit as Lim_T. If we use Bot_T values, these errors would 0% while having smaller time optimization ratios.

In reducing results, some of the arrays can not reduced at all in our benchmark set. However, benchmarks with high transistor count diversity have up to 72% reducing ratios. These reducing ratios correlated with the runtime optimization ratios but each algorithm responses slightly differently to this method. In manner of runtimes, up to 76% optimization achieved by this approach.

Table 6.1 : Output deviation rate between reduced and not reduced matrices and time improvement rates for different benchmarks considering different algorithms over 100 trials each.

Algorithm		**Proposed Hill Climbing**		**Simulated Annealing [18]**		**Genetic [19], [22]**		**Random Generation**	
Benchmark $(R \times C)$	**Reduced Column Ratio**	**Delay Increase Rate**	**Time Decrease Rate**	**Delay Increase Rate**	**Time Decrease Rate**	**Delay Increase Rate**	**Time Decrease Rate**	**Delay Increase Rate**	**Time Decrease Rate**
5ex1 (75×14)	71%	0.2%	64.8%	1.3%	32.26%	1.8%	4.5%	0.9%	60.7%
Inc (34×14)	39%	2.6%	27.2%	0.1%	26.2%	0.5%	8.6%	1.01%	26%
clip (167×18)	19%	0.6%	24.9%	0.3%	15.6%	0.1%	3.5%	0.8%	19.81%
Misex2 (29×40)	72%	0.2%	76.4%	1.1%	26.8%	0.1%	4.48%	1.8%	43.2%
9sym (87×18)	-	-	-	-	-	-	-	-	-
Bw (65×10)	44.3%	3.1%	41.08%	1.4%	19.02%	0.6%	2.9%	0.16%	35.7%
Rd53 (32×10)	4%	1.03%	10.2%	1.2%	1.56%	0.4%	3.05%	1.06%	0.5%
Rd73 (141×14)	24%	0.7%	22.8%	1.12%	17.43%	0.8%	0.5%	0.7%	21.5%
Sao2 (58×18)	48.2%	1.05%	24.9%	1.3%	19.86%	0.3%	2.4%	0.9%	37.6%
Table5 (158×34)	44.2%	1.2%	50.6%	0.4%	34.12%	0.3%	22.6%	0.4%	39.57%

Table 6.2 : Delay optimization rates for the proposed hill climbing algorithm having different initial column mapping techniques over 500 random generated samples.

Benchmark $(R \times C)$	***COV***	**Proposed Greedy**	**Greedy [20]**	**Random**
Random 1 (6×6)	0.2	38.3%	36.1%	36%
$(CR : 40\%)$	0.3	44.4%	41.9%	38.4%
Random 2 (12×12)	0.2	35.3%	32.3%	33.7%
$(CR : 40\%)$	0.3	40.9%	40.2%	38.8%
Random 3 (24×24)	0.2	29.6%	28.6%	28.8%
$(CR : 40\%)$	0.3	35.1%	34.8%	34.3%
Random 4 (48×48)	0.2	26.01%	25.08%	25.4%
$(CR : 40\%)$	0.3	29.25%	28.8%	28.05%

6.3 Proposed Algorithm Simulations and Comparisons

First, we simulate our proposed Hill Climbing algorithm considering different starting column mappings. Random, greedy [20] and our proposed greedy regarding to minimum possible values are used as different cases. We use different array sizes and coefficient of variation values. Table 6.2 gives comparison for these different initial column mapping performances for our hill climbing algorithm. Our algorithm gives approx. 2-3% better optimization rates for the same runtimes.

As a second inspection, we analyzed timings and the delay optimization rates for different sized random arrays with 40% *CR* in Figure 6.1. Here, both increasing row and column counts decreases delay optimization rates. Here, decreasing column count has more positive effect on delay optimization rate than decreasing row count. Therefore, arrays with the lower column counts expected to have better delay optimization results.

We also make detailed comparisons of our algorithm with three different algorithms in the literature in Table 6.3. In terms of the worst-case delays, represented by "Delay Opt. Rate", our algorithm gives the best results in almost all cases. In terms the runtime, our algorithm gives the best results in all of the cases. Another point is that our algorithm does a better job for randomly generated ones than standard benchmarks. As we previously explain, our algorithm is much more tolerable to the row increase than the column increase. This feature is not compatible with the standard benchmark

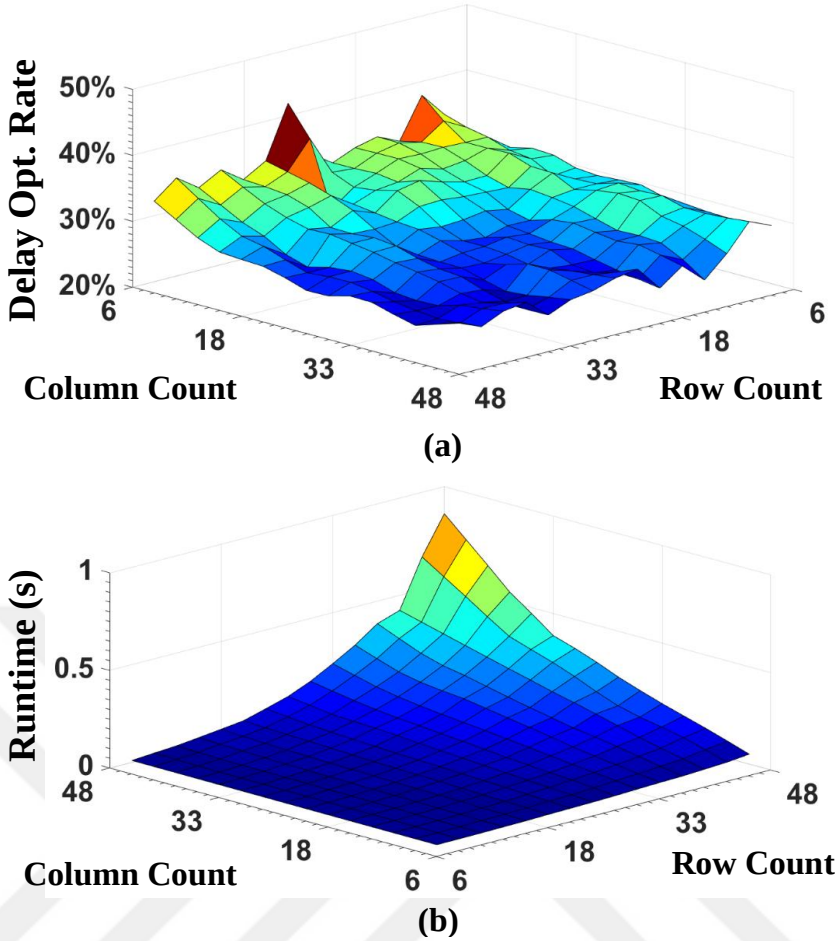


Figure 6.1 : (a) Delay optimization rates of proposed mapping algorithm (b) average maximum runtime of proposed algorithm for different sized arrays. All values are average of 100 samples.

circuits generally having less products (rows) than literals (columns). However, for randomly generated benchmarks having same number of rows and columns, our algorithm's performance is quite satisfactory. Note that we run memetic algorithm from [23] maximum 20 seconds since our proposed algorithm has significantly lower runtimes. Note that we did not apply our matrix reducing and initial column mapping methods for our hill climbing algorithm on Table 6.3 to have a fair comparison.

6.4 Defect Tolerance Simulations

To test our hill climbing algorithm on defect tolerance, we used different defect rates on VMs to find successful mapping ratios over 100 cases for different random matrices and benchmarks.

On Figure 6.2, different random matrix cases inspected. On the first graph we used 24×24 arrays with different crosspoint rates (CR). Here, having higher crosspoint rate

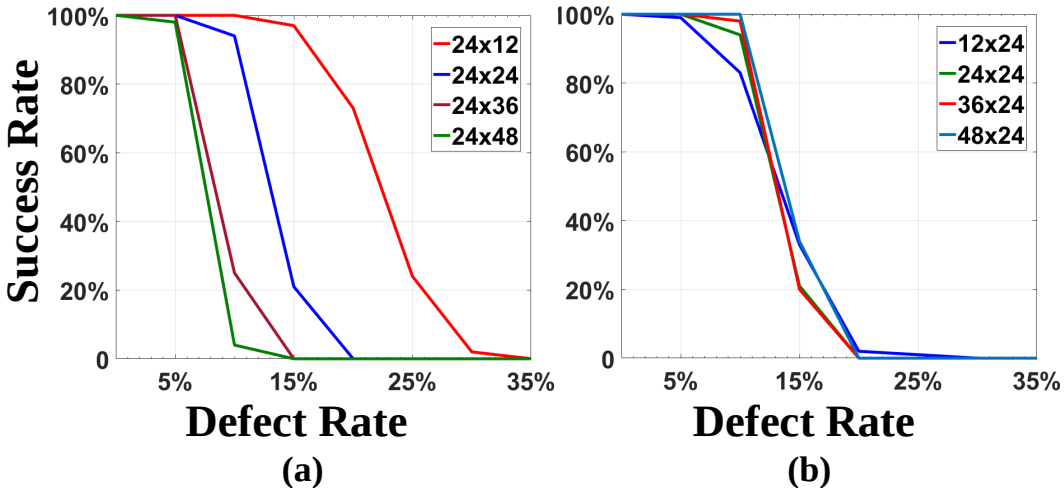


Figure 6.2 : Defect tolerance results and comparisons for different type of random arrays.

results as lower defect tolerance as expected. On the second graph we used *CR* as 40%, row count as 24 and inspected cases with different column count. Here, increasing column count decreases defect tolerance since we need to correctly map more columns for the same row size. On the third graph we used constant column count as 24 and inspected different row sizes. Here, since we use a constant *CR* as 40%, finding correct mapping has the same difficulty on each case. Therefore, increasing column count is more critical than increasing row count for the random cases with constant *CR*. We generally achieved 10% - 30% defect tolerance on these cases.

As our next defect tolerance simulation we used our standard benchmark set. On Figure 6.3 comparison of different benchmarks are given. We achieved to tolerate 5% to 10% defect rates for these benchmarks. Rather than random matrices, benchmark types, therefore function types directly affects defect toleration rates on these matrices.

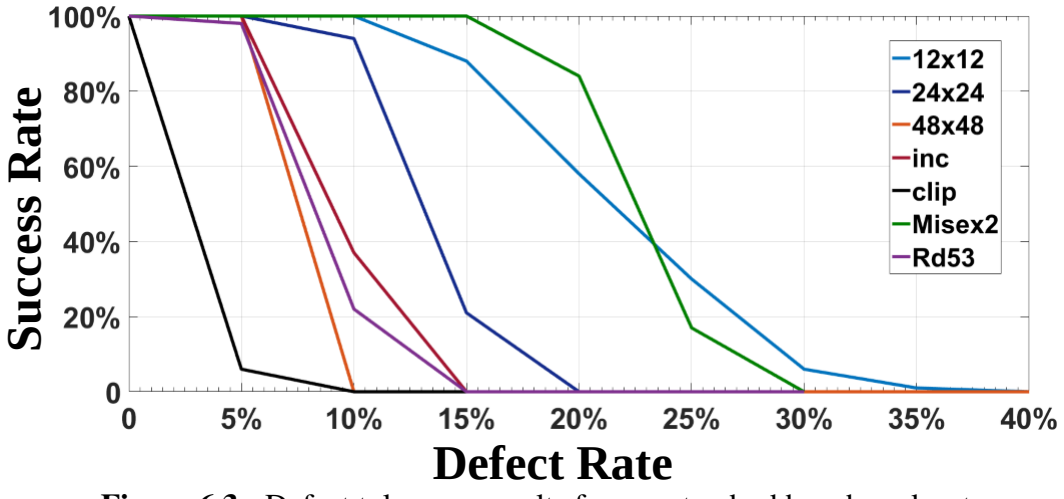


Figure 6.3 : Defect tolerance results for our standard benchmark set.

Table 6.3 : Delay optimization rate and runtime comparisons for our proposed hill climbing algorithm; $COV = 0.2$. Bold values/elements represent the best results.

Benchmark $(C \times R)$	**Hill Climbing**		**Memetic Alg. [23]**		**Simulated Ann. [18]**		**Genetic Alg. [19], [22]**	
	Delay Opt. Rate	Runtime (s)	Delay Opt. Rate	Runtime (s)	Delay Opt. Rate	Runtime (s)	Delay Opt. Rate	Runtime (s)
5ex1 (75×14)	**25.7%**	**0.11**	-	>20	15.3%	0.185	22%	0.69
inc (34×14)	**20.8%**	**0.0096**	20.6%	5.4	15%	0.67	14.5%	0.76
clip (167×18)	**19.01%**	**0.09**	-	>20	9.3%	0.18	11.2%	0.12
Misex2 (29×40)	**24.5%**	**0.094**	15.2%	1.4	13%	0.97	11.1%	0.54
9sym (87×18)	**12.5%**	**0.072**	-	>20	8.3%	0.11	8.8%	0.78
Bw (65×10)	**20.8%**	**0.0078**	**-**	>20	13.7%	0.15	12.6%	0.74
Rd53 (32×10)	19.2%	**0.0086**	**23.1%**	7.1	11.6%	0.12	10.1%	0.34
Rd73 (141×14)	**13.86%**	**0.038**	-	>20	7.8%	0.27	8.1%	0.82
Sao2 (58×18)	**17.94%**	**0.1**	**-**	>20	8.9%	0.16	9%	0.78
Table5 (158×34)	**16.1%**	**0.55**	-	>20	7.5%	0.59	8.3%	1.85
6×6 $(CR = 40\%)$	**37.9%**	**0.0007**	20.3%	1.02	36.7%	0.04	10.9%	0.53
12×12 $(CR = 40\%)$	**32.7%**	**0.0025**	20.6%	1.03	27.6%	0.4	19%	0.61
24×24 $(CR = 40\%)$	**31.7%**	**0.015**	16.1%	1.2	18.4%	0.68	11.5%	0.78
48×48 $(CR = 40\%)$	**24.8%**	**0.37**	10.8%	4.24	12.8%	1.45	15.3%	1.14

Table 6.4 : Comparison of $D/VTLM$ results for proposed hill climbing algorithm for 5% defect rate. Bold values/elements represent the best results.

Defect Rate: 5%	**Hill Climbing Alg.**			**Memetic Algorithm [23]**			**Simulated Annealing Alg. [18]**			**Genetic Algorithm [19], [22]**		
Benchmark $(C \times R)$	Succ. Rate	Delay Opt. Rate	Run time	Succ. Rate	Delay Opt. Rate	Run time	Succ. Rate	Delay Opt. Rate	Run time	Succ. Rate	Delay Opt. Rate	Run time
5ex1 (75×14)	**96%**	16.6%	**0.059**	-	-	>20	0%	-	0.16	5%	**21.2%**	0.75
inc (34×14)	**100%**	16.4%	**0.032**	**98%**	**18.9%**	1.7	53%	5.5%	1.25	43%	14.3%	0.6
clip (167×18)	**2%**	**8.6%**	**0.29**	-	-	>20	0%	-	0.67	0%	-	1.07
Misex2 (29×40)	**100%**	**24%**	**0.2**	**100%**	15.3%	1.22	23%	4.8%	0.33	79%	12%	0.82
9sym (87×18)	**25%**	**5.1%**	**0.146**	-	-	>20	0%	-	0.39	0%	-	0.77
Bw (65×10)	**85%**	**14%**	**0.027**	-	-	>20	38%	4.2%	0.29	12%	12.4%	0.6
Rd53 (32×10)	96%	12.2%	**0.014**	**100%**	**21.7%**	2.95	56%	5.5%	0.22	43%	10.4%	0.55
Rd73 (141×14)	**1%**	**6.6%**	**0.14**	-	-	>20	0%	-	0.5	0%	-	0.81
Sao2 (58×18)	**77%**	**11.1%**	**0.084**	-	-	>20	0%	-	0.32	1%	8.3%	0.69
Table5 (158×34)	0%	-	3.01	-	-	>20	0%	-	0.99	0%	-	1.61
6×6 $(CR = 40\%)$	**100%**	**21%**	**0.0027**	**100%**	17.9%	1.03	**100%**	19.3%	0.02	99%	14.3%	0.52
12×12 $(CR = 40\%)$	**100%**	**18.61%**	**0.006**	**100%**	18.2%	1.05	**100%**	14.7%	0.3	93%	16.8%	0.59
24×24 $(CR = 40\%)$	**100%**	**17.6%**	**0.09**	**100%**	15.07 %	1.15	6%	6.05%	0.42	38%	13.09%	0.76
48×48 $(CR = 40\%)$	**100%**	**18.9%**	**1.8**	**100%**	10.2%	2.24	0%	-	1.25	0%	-	1.13

Table 6.5 : Comparison of $D/VTLM$ results for proposed hill climbing algorithm for 10% defect rate. Bold values/elements represent the best results.

Defect Rate: 10%	**Hill Climbing Alg.**			**Memetic Algorithm [23]**			**Simulated Annealing Alg. [18]**			**Genetic Algorithm [19], [22]**		
Benchmark $(C \times R)$	Succ. Rate	Delay Opt. Rate	Run time	Succ. Rate	Delay Opt. Rate	Run time	Succ. Rate	Delay Opt. Rate	Run time	Succ. Rate	Delay Opt. Rate	Run time
5ex1 (75×14)	**2%**	**15.7%**	**0.056**	-	-	>20	0%	-	0.17	0%	**-**	0.68
inc (34×14)	26%	10.8%	**0.03**	**82%**	**17.3%**	1.51	0%	-	1.29	1%	14.1%	0.608
clip (167×18)	0%	**-**	0.32	-	-	>20	0%	-	0.69	0%	-	1.14
Misex2 (29×40)	**100%**	**22.1%**	**0.23**	**100%**	12.9%	1.11	0%	-	0.35	2%	7.1%	0.827
9sym (87×18)	0%	**-**	0.142	-	-	>20	0%	-	0.41	0%	-	0.76
Bw (65×10)	**14%**	**12.1%**	**0.02**	-	-	>20	0%	-	0.3	0%	-	0.6
Rd53 (32×10)	18%	8.4%	**0.014**	**98%**	**20.6%**	2.11	0%	-	0.22	0%	-	0.56
Rd73 (141×14)	0%	**-**	0.14	-	-	>20	0%	-	0.51	0%	-	0.8
Sao2 (58×18)	**1%**	**3.4%**	**0.09**	-	-	>20	0%	-	0.34	0%	-	0.69
Table5 (158×34)	0%	-	3.03	-	-	>20	0%	-	1.02	0%	-	1.81
6×6 $(CR = 40\%)$	**100%**	14%	**0.0017**	**100%**	**18.2%**	1.04	95%	13.7%	0.15	97%	17.5%	0.52
12×12 $(CR = 40\%)$	**100%**	14.4%	**0.008**	**100%**	**19.06%**	1.02	**94%**	13.6%	0.32	79%	14.2%	0.59
24×24 $(CR = 40\%)$	97%	**17.14%**	**0.091**	**100%**	14.7%	1.13	0%	-	0.44	0%	-	0.67
48×48 $(CR = 40\%)$	0%	**-**	2.03	0%	-	2.2	0%	-	1.36	0%	-	1.13

6.5 Column Replication Simulations

Random arrays with Crosspoint Rate (CR) as 40% are used with different sizes and area overheads to reach performance values under determined limits are given in Figure 6.4. For different limits, 2 different conditions inspected. Here, increasing array size decreases area overhead for both *Limit* 1 and *Limit* 2. For the *Limit* 2 we generally needed area overhead between 70% - 90%. For the *Limit* 1 these ratios drop between 60% - 17%. Lower limits have higher area overhead values, as expected. Note that increasing column number drastically increases area overhead for *Limit* 2 overheads.

On Table 6.6 different benchmark outputs for the same conditions compared along with the power overheads and expected worst case delay optimization. Here, high transistor count diversity makes high worst case delay optimization with little area overhead (ex. Sao2). However, exact opposite state gives not significant worst case delay optimization values considered needed area overhead (ex. Rd53).

Examples of power - delay products changes as the limit decreases given in Figure 6.5 for different sized matrices. Until reaching large array sizes, this product always decreasing. Since these Function Matrices are random with a certain crosspoint rate *CR* and each crosspoint assignment is a bernoulli trial, column transistor counts would have a gaussian distribution in high sample sizes. As a result, decreasing limit values needs to replicate more columns, since mid-range column count is higher than high and low extremes. High sample sizes makes larger arrays prone to this situation, so power

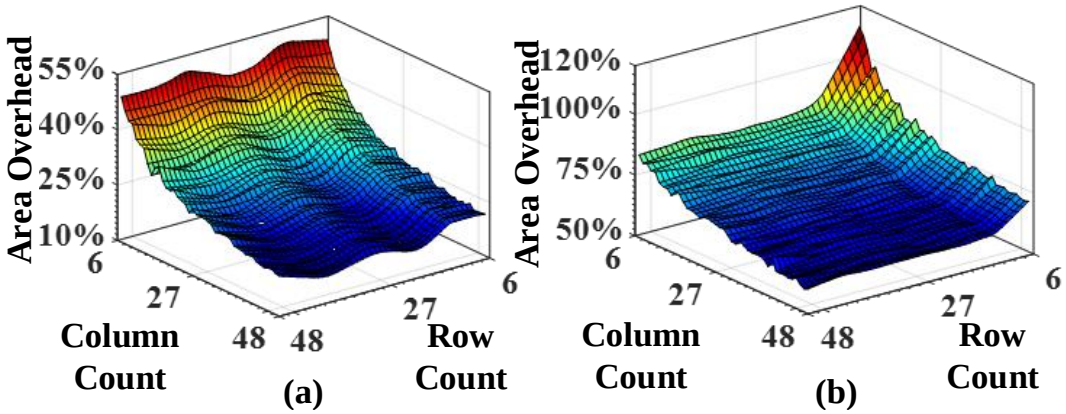


Figure 6.4 : Area overheads for different sized arrays. Each graph uses a different goal value. (a) COV = 0.3, *Goal* 2 (b) COV = 0.3, *Goal* 1

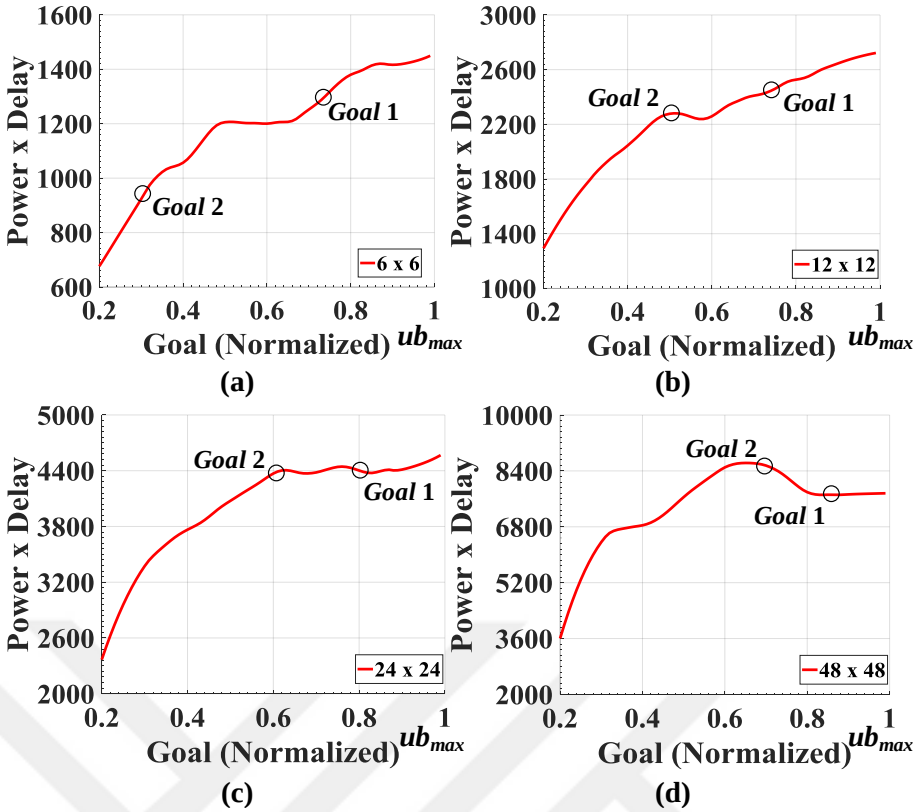


Figure 6.5 : Power - Delay Product values compared to *Goal* 1 and *Goal* 2 values for (a) 6×6 (b) 12×12 (c) 24×24 (d) 48×48 random matrices with %40 crosspoint ratio.

- delay product is increasing up to some point. Benchmarks had decreasing products most of the times in our trials.

Table 6.6 : Area, power overheads and worst case delay optimization ratios given for different benchmarks and random arrays for column replication approach. Each case is mean of 1000 different FM for random matrices.

Limit type	**Limit 2**						**Limit 1**					
Coeff. of Variation	**0.3**			**0.2**			**0.3**			**0.2**		
Benchmark	**Area**	**Power**	**Delay**	**Area**	**Power**	**Delay**	**Area**	**Power**	**Delay**	**Area**	**Power**	**Delay**
5ex1 (75x14)	64%	55%	-29%	42%	34%	-20%	42%	34%	-14%	21%	16%	-10%
İnc (34x14)	35%	18%	-31%	28%	13%	-22%	21%	10%	-16%	14%	6%	-11%
clip (167x18)	16%	10%	-20%	16%	10%	-13%	11%	6%	-9%	5%	3%	-6.6%
Misex2 (29x40)	7.5%	1.2%	-35%	5%	0.7%	-24%	5%	0.7%	-18%	5%	0.7%	-12%
9sym (87x18)	100%	100%	-27%	100%	100%	-19%	100%	100%	-13%	100%	100%	-9%
Bw (65x10)	30%	16%	-24%	10%	4.7%	-17%	10%	4%	-12%	10%	4.7%	-8%
Rd53 (32x10)	100%	100%	-35%	70%	64%	-25%	50%	43%	-18%	50%	43%	-13%
Rd73 (141x14)	50%	43%	-19%	50%	43%	-13%	35%	30%	-10%	28%	23%	-6%
Sao2 (58x18)	5.5%	1%	-21%	5.5%	1%	-15%	5.5%	1%	-10%	5.5%	1%	-7.5%
Table5 (158x34)	2.9%	1%	-15%	2.9%	1%	-11%	2.9%	1%	-8%	2.9%	1%	-5%
6x6 (CR:40%)	140%	117%	-60%	82%	66%	-44%	55%	40%	-30%	52%	37%	-22%
12x12 (CR:40%)	83%	73%	-46%	64%	52%	-34%	46%	35%	-23%	27%	19%	-17%
24x24 (CR:40%)	75%	68%	-36%	47%	40%	-25%	33%	26%	-18%	21%	16%	-12%
48x48 (CR:40%)	68%	63%	-27%	38%	33%	-20%	20%	17%	-13%	13%	10%	-9%

7. CONCLUSION AND DISCUSSION

Due to high process variations on nano-crossbar array manufacturing with self assembly methods, defect and variance tolerant logic mapping techniques needed to be developed. Previous works mostly focused on the probabilistic methods to achieve a logic mapping with optimized objectives. In this work, we presented a Hill Climbing algorithm as an alternative successful approach to variation tolerant logic mapping problem. We focused on worst case delay optimization since this value determines the bottleneck speed of the manufactured nano-crossbar array. Our method achieved better runtime (upto 40 times) and accuracy results (upto %40) especially for the larger arrays with variations. As a future design aim for this algorithm, we focus on different objective improvements and different nano-crossbar array switch types like diode or 4-terminal. Also, Hill Climbing search paradigm can be improved with new heuristics for column re-ordering process.

As an algorithm independent pre-processing method, we reduced function matrices to their performance equivalent for $Objective$ 1 optimization. We applied this method as the first step on our Hill Climbing algorithm. Reduced matrices can be used in any variation tolerant logic mapping process for designed objective. Depending on the algorithm and function characteristics, matrix reducing improves runtime dramatically for most cases (upto %85 time improvement). As a future work, we aim to implement matrix reducing method for different objectives and nano-crossbar array switch types like diode or 4-terminal.

Since defect and variation tolerance are both design problems for nano-crossbar arrays, we extended our proposed hill climbing algorithm for defect toleration. Since our algorithm tries to avoid crosspoints with the high delay values, we used very high delay values for defective crosspoints. Due to this nature of the algorithm, defective crosspoints can be avoided. We find %10-30 defect tolerance for random matrices with %40 crosspoint ratio and %5-10 defect tolerance for our benchmark set.

As an extension to our work, we explained column replication approach for achieving desired performances with known process parameters without using delay testing or any search algorithm. This approach trades extra area and power consumption for lower worst case delay values. As a future work, implementation of this method on different nano-crossbar array types can be achieved. Also, we can supplement this approach with extensive mapping methods to optimize different objectives. Note that current replication method is mostly a preliminary approach to achieve a mapping without variation array.

REFERENCES

[1] **Zhang, A.**, **Zheng, G. and Lieber, C.M.**, (2016). Nanoelectronics, Circuits and Nanoprocessors, Nanowires, Springer, pp.103–142.

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

[3] **Hamoudi, H.** (2014). Crossbar nanoarchitectonics of the crosslinked self-assembled monolayer, *Nanoscale research letters*, *9*(1), 287.

[4] **Chen, Y.**, **Jung, G.Y.**, **Ohlberg, D.A.**, **Li, X.**, **Stewart, D.R.**, **Jeppesen, J.O.**, **Nielsen, K.A.**, **Stoddart, J.F. and Williams, R.S.** (2003). Nanoscale molecular-switch crossbar circuits, *Nanotechnology*, *14*(4), 462.

[5] **Snider, G.**, **Kuekes, P. and Williams, R.S.** (2004). CMOS-like logic in defective, nanoscale crossbars, *Nanotechnology*, *15*(8), 881.

[6] **DeHon, A. and Gojman, B.** (2011). Crystals and snowflakes: building computation from nanowire crossbars, *Computer*, *44*(2), 37–45.

[7] **Linn, E.**, **Rosezin, R.**, **Tappertzhofen, S.**, **Böttger, U. and Waser, R.** (2012). Beyond von Neumann—logic operations in passive crossbar arrays alongside memory operations, *Nanotechnology*, *23*(30), 305205.

[8] **Rosezin, R.**, **Linn, E.**, **Kugeler, C.**, **Bruchhaus, R. and Waser, R.** (2011). Crossbar logic using bipolar and complementary resistive switches, *IEEE Electron Device Letters*, *32*(6), 710–712.

[9] **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.

[10] **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.

[11] **Snider, G.**, **Kuekes, P.**, **Hogg, T. and Williams, R.S.** (2005). Nanoelectronic architectures, *Applied Physics A*, *80*(6), 1183–1195.

[12] **Cui, Z.**, (2017). Applications of Nanofabrication Technologies, Nanofabrication, Springer, pp.401–426.

[13] **Lu, W. and Lieber, C.M.** (2007). Nanoelectronics from the bottom up, *Nature materials*, *6*(11), 841–850.

[14] **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.

[15] **Gojman, B. and DeHon, A.** (2009). VMATCH: Using logical variation to counteract physical variation in bottom-up, nanoscale systems, *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, IEEE, pp.78–87.

[16] **Ghavami, B.**, **Tajary, A.**, **Raji, M. and Pedram, H.** (2010). Defect and variation issues on design mapping of reconfigurable nanoscale crossbars, *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, IEEE, pp.173–178.

[17] **Ghavami, B.** (2016). Joint defect-and variation-aware logic mapping of multi-outputs crossbar-based nanoarchitectures, *Journal of Computational Electronics*, *15*(3), 959–967.

[18] **Tunc, C. and Tahoori, M.B.** (2010). Variation tolerant logic mapping for crossbar array nano architectures, *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, IEEE Press, pp.855–860.

[19] **Yang, Y.**, **Yuan, B. and Li, B.** (2011). Defect and variation tolerance logic mapping for crossbar nanoarchitectures as a multi-objective problem, *Information Science and Technology (ICIST), 2011 International Conference on*, IEEE, pp.1139–1142.

[20] **Zhong, F.**, **Yuan, B. and Li, B.** (2014). Hybridization of NSGA-II with greedy re-assignment for variation tolerant logic mapping on nano-scale crossbar architectures, *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ACM, pp.97–98.

[21] **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, *IEEE Transactions on Evolutionary Computation*, *18*(6), 846–859.

[22] **Zhong, F.**, **Yuan, B. and Li, B.** (2016). A hybrid evolutionary algorithm for multiobjective variation tolerant logic mapping on nanoscale crossbar architectures, *Applied Soft Computing*, *38*, 955–966.

[23] **Yuan, B.**, **Li, B.**, **Chen, H. and Yao, X.** (2016). Defect-and Variation-Tolerant Logic Mapping in Nanocrossbar Using Bipartite Matching and Memetic Algorithm, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *24*(9), 2813–2826.

[24] **Zamani, M. and Tahoori, M.B.** (2011). Variation-aware logic mapping for crossbar nano-architectures, *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, IEEE Press, pp.317–322.

[25] **Sapatnekar, S.S.**, (2011). Statistical Design of Integrated Circuits, Low-Power Variation-Tolerant Design in Nanometer Silicon, Springer, pp.109–149.

[26] **Morgul, M.C.**, **Peker, F. and Altun, M.** (2016). Power-Delay-Area Performance Modeling and Analysis for Nano-Crossbar Arrays, *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*, IEEE, pp.437–442.

CURRICULUM VITAE

Name Surname: Furkan Peker

Place and Date of Birth: Trabzon 1991

E-Mail: pekerf@itu.edu.tr

EDUCATION:

- **B.Sc.:** 2014, Yıldız Technical University, Mechanical Engineering Faculty, Department of Mechatronics Engineering
- **M.Sc.:** 2017, İstanbul Technical University, Electrical Engineering Faculty, Department of Electronics and Communications Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2-link ball placing industrial robot arm design and PID control (2013) as Mechatronic Design course project.
- Realization and optimization of a multi-objective pathfinder algorithm (2013) as Artificial Intelligence course project.
- Internship at FİGES A.Ş. (MATLAB Distrubitor and Military Project contractor in Turkey, 2014)
- TUBITAK 2241/A (2209/A) support for the undergraduate final thesis of "Design and realizing of Ball and Plate system with PID control and Image Processing" (2014).
- Graduated as Honored Student from Yıldız Technical University (2014).
- Research Assistant at İstanbul Technical University (2015 - cont.).
- Researcher at TUBITAK Career Project #113E760 "Synthesis and Optimization of Switching Nanoarrays". (2015-2017)
- Researcher at EU-H2020-RISE Project "Synthesis and Performance Optimization of a Switching Nano-Crossbar Computer (NANOxCOMP)" (2015- cont.).
- Exchange Researcher of EU-H2020-RISE Project "Synthesis and Performance Optimization of a Switching Nano-Crossbar Computer (NANOxCOMP)" at IROC Tecnologies (Jan-May 2016, Grenoble/France).

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- Morgul, M.C., Peker, F. and Altun, M. (2016). Power-Delay-Area Performance Modeling and Analysis for Nano-Crossbar Arrays, VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on, IEEE, pp.437–442.