FRET 37
FRONTIERS IN ELECTRONIC TESTING

Mohammad Tehranipoor

Emerging Nanotechnologies

Test, Defect Tolerance, and Reliability

Emerging Nanotechnologies

Frontiers in Electronic Testing

Series Editor: Vishwani Agrawal
Auburn University
Auburn, Alabama

Emerging Nanotechnologies: Test, Defect Tolerance, and Reliability
Mohammad Tehranipoor (Ed.)
Volume 37, ISBN 978-0-387-74746-0, 2008

Oscillation-Based Test in Mixed-Signal Circuits
G. Huertas Sánchez, D. Vázquez Garcia de la Vega, A. Rueda Rueda, and J.L. Huertas Díaz
Volume 36, ISBN 978-1-4020-5314-6, 2006

The Core Test Wrapper Handbook: Rationale and Application of IEEE Std. 1500™
Francisco da Silva, Teresa McLaurin, and Tom Waayers
Volume 35, ISBN 978-0-387-30751-0, 2006

Defect-Oriented Testing for Nano-Metric CMOS VLSI Circuits, Second Edition
Manoj Sachdev and José Pineda de Gyvez
Volume 34, ISBN 978-0-387-46546-3, 2007

Digital Timing Measurements: From Scopes and Probes to Timing and Jitter
Wolfgang Maichen
Volume 33, ISBN 978-0-387-31418-1, 2006

Fault-Tolerance Techniques for SRAM-Based FPGAs
Fernanda Lima Kastensmidt, Luigi Carro, and Ricardo Reis
Volume 32, ISBN 978-0-387-31068-8, 2006

Data Mining and Diagnosing IC Fails
Leendert M. Huisman
Volume 31, ISBN 978-0-387-24993-3, 2005

Fault Diagnosis of Analog Integrated Circuits
Prithviraj Kabisatpathy, Alok Barua, and Satyabroto Sinha
Volume 30, ISBN 978-0-387-25742-6, 2005

Introduction to Advanced System-on-Chip Test Design and Optimization
Erik Larsson
Volume 29, ISBN 978-1-4020-3207-3, 2005

Embedded Processor-Based Self-Test
Dimitris Gizopoulos, A. Paschalis, and Yervant Zorian
Volume 28, ISBN 978-1-4020-2785-7, 2004

Advances in Electronic Testing: Challenges and Methodologies
Dimitris Gizopoulos (Ed.)
Volume 27, ISBN 978-0-387-29408-7, 2006

Testing Static Random Access Memories: Defects, Fault Models and Test Patterns
Said Hamdioui
Volume 26, ISBN 978-1-4020-7752-4, 2004

Verification by Error Modeling: Using Testing Techniques in Hardware Verification
Katarzyna Radecka and Zeljko Zilic
Volume 25, ISBN 978-1-4020-7652-7, 2004

Elements of STIL: Principles and Applications of IEEE Std. 1450
Gregory A. Maston, Tony R. Taylor, and Julie N. Villar
Volume 24, ISBN 978-1-4020-7637-4, 2003

Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation
Alfredo Benso and P. Prinetto (Eds.)
Volume 23, ISBN 978-1-4020-7589-6, 2003

(Continued after index)

Mohammad Tehranipoor
Editor

Emerging Nanotechnologies

Test, Defect Tolerance, and Reliability

Editor

Mohammad Tehranipoor

Department of Electrical and Computer
Engineering
University of Connecticut
Storrs, CT 06269
USA

Series Editor

Vishwani Agrawal

Department of Electrical and Computer
Engineering
Auburn University
Auburn, AL 36849
USA

ISBN 978-0-387-74746-0 e-ISBN 978-0-387-74747-7

Library of Congress Control Number: 2007933699

© 2008 Springer Science+Business Media, LLC

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper.

9 8 7 6 5 4 3 2 1

springer.com

Preface

The foundations of nanotechnology have emerged over many decades of research in various fields. Over the years, computer circuits have been becoming smaller and chemicals have been getting more complex. Biochemists have learned more about how to study and control the molecular basis of organisms. Mechanical engineering has been getting more precise which resulted in, for instance, emerging nanoelectro-mechanical systems (NEMS). Computer engineering have been getting a great deal of knowledge on how to design circuits with defective components. Today, in the young field of nanotechnology, scientists and engineers of various fields are taking control of atoms and molecules individually, manipulating them and putting them to use with an extraordinary degree of precision which was considered impossible many years ago. Word of the promise of nanotechnology is spreading quickly, and the air is thick with news of nanotech breakthroughs especially over the last few years. Public awareness of nanotech is clearly on the rise, too, partly because references to it are becoming more common in popular culture and everyday life.

The wires and switches inside computer chips have been getting steadily smaller for decades. They have already crossed the 100-nm threshold, sufficient to be considered nanotechnology by the National Nanotechnology Initiative (NNI) definition. As they continue to shrink, quantum effects will become increasingly important, and future designs will stop working if not carefully taken into consideration. Researchers in academia and industry are working on various technologies, but among those there are few nanoscale technologies that could potentially take over in near future. One is molecular electronics: the use of single molecules (or sometimes, small clusters of molecules) to build wires and switches. Another is quantum dots: instead of letting electrons flow through wires, the electrons are tethered in place and only shift back and forth. This shift causes nearby electrons to shift also, which is useful for signaling and computation. Finally, carbon nanotube (CNT) based interconnects and transistors; CNTs have shown promising electrical behavior compared to copper used in Complementary Metal Oxide Semiconductor (CMOS) technology.

Technology Scaling Challenges and Effects

As functional density and operating frequency increase, the number of interconnects and length of interconnects are expected to increase as well. Over the years, the number of metal layers has incrementally increased from the original one. Using six to ten metal layers in industry is a common practice nowadays. Increasing number of metal layers in turn increases the number of vias where it is proven that vias are the main sources of defects. The situation will grow worse since the number of metal layers will even further increase going up to 12 within the next few years.

The material of the layers used in fabrication processes has also undergone a major change from aluminum to copper. Using copper provides a better scalability compared to aluminum. As technology scales and more transistors are integrated on a chip, the interconnects become longer. For high-speed nanometer technology designs the interconnect delay dominates gate delay. It is predicted that in the near future the longest path in the design will be the critical one not the paths with more gates. In nanometer technology era, crosstalk will be a major contributor to interconnect delay. To keep the resistance of the wires low as technology scales, the interconnects are becoming narrower and taller. This results in large cross-coupling capacitances which are now dominating substrate capacitances.

To reduce the power and minimize the negative impact of hot career, which causes reliability issues overtime, the power supply is reduced. However, the transistor voltage threshold is not scaling proportionally which results in increase in the circuit sensitivity and reduction in noise margin. The scaling also increases the leakage current. In 65 nm technology, the static power consumption contributes to 50% of total power consumption while it is expected to further increase in 45 and 32 nm technologies. Negative bias temperature instability (NBTI) is considered a growing threat to device reliability in sub-100 nm technologies as well.

Technology scaling also poses many challenging design and test issues. The power and speed are two important parameters in today's designs. The low power supply has increased circuits sensitivity to noise caused by IR-drop, crosstalk, and process variations. The voltage threshold does not scale proportionally resulting in reduced noise margin. The wavelength of the light used for imaging the geometries is longer than the geometry desired for printing. For example, a designer uses an almost 200 nm light source for a 130 nm gate length. The circuit speed will be limited by quantum effects along with high power and temperature in future designs. Temperature variation can significantly affect circuit performance. The process variation increases clock skew resulting higher switching activity, hence higher temperature and power supply noise. The continuous decrease in transistor feature size has been pushing the CMOS process to its physical limits caused by ultra-thin gate oxides, short channel effects, doping fluctuations, and the unavailability of lithography in nanoscale range. To be able to continue the size/speed improvement trends

according to Moore's Law, research investments are growing on a wide range of emerging devices and technologies.

Emerging Technologies

This book covers various technologies that have been suggested by researchers over the last decades such as chemically assembled electronic nanotechnology, Quantum-dot Cellular Automata (QCA), nanowires (NWs), and carbon nanotubes (CNTs). Each of these technologies offers various advantages and disadvantages. Some suffer from high power, some work in very low temperature and some other need indeterministic bottom-up assembly. These emerging technologies are not considered as a direct replacement for CMOS technology and may require a completely new architecture to achieve their functionality.

Molecular logic devices are based on electron transport properties through a single molecule. Two terminal molecular devices currently being explored consists of thousands of molecules operating in parallel as digital switches or analog diodes. In both cases, a voltage applied to a molecular layer (group of molecules in parallel) results in reconfiguration of the molecular components. This creates a nano-switch where the reconfiguration capability provide us with the opportunity for computing. A near term opportunity of molecular electronics is in integration of molecular devices with sub-50 nm CMOS components to form a hybrid system. A full-molecular system is considered a potential long-term opportunity. In addition to two terminal switches, few other molecular components emerged over the past few years, e.g. bistable switch, molecular NEMS, and spin-based molecular devices.

Carbon nanotube is a subset of molecular electronic materials. It is a cylinder formed from an atomic sheet of carbon atoms. The carbon atoms are bounded together into an array of hexagons which forms a planar sheet. This sheet is rolled up to form a tube. Carbon nanotubes can have diameters up to 15 nm and lengths up to few microns. The diameter and the way the sheet is rolled up determine whether the carbon nanotube has metal or semiconductor properties. The semiconductor tube can be doped n-type and p-type, making it possible to create n–p junction. Carbon nanotubes have shown strong current capability which makes it interesting to IC designers to replace copper with carbon nanotube, however, the integration will be expensive.

A novel mechanism for transmitting and processing information has been extensively investigated in theoretical work on quantum-dot cellular automata (QCA). This work assumes arrays of cells built from quantum dots, on a molecular scale, from individual redox centers. The charges move within the cells in response to external electric fields. It is fascinating that based on such scheme, there is no need to let charges flow through the cells. Wires, AND/OR gates, clocked QCA cells, QCA memory cell, and a shift register are the components that have been successfully demonstrated. Today, standard solid state Quantum-dot Cellular Automata cell design considers the distance between quantum dots to be about 20 nm, and a distance between cells of about 60 nm. Compared to CMOS technology, QCA is expected to present less variability at this scale.

Digital microfluidics is an alternative technology for lab-on-a-chip systems based upon micromanipulation of discrete droplets. Microfluidic processing is performed on unit-sized packets of fluid which are transported, stored, mixed, reacted, or analyzed in a discrete manner using a standard set of basic instructions. Recent advances in microfluidics technology have led to the design and implementation of miniaturized devices for various biochemical applications. These microsystems have shown promises to revolutionize biosensing, clinical diagnostics, and drug discovery. Such applications can benefit from the small size of biochips compared to conventional laboratory methods.

Developments in these nanoscale technologies provide the hope that current trend of integration of electronic devices can be continued. Due to their small feature sizes and self-assembly-based fabrication methods, nanoscale devices present many challenges in the area of testing, defect tolerance, and reliability. As nanoscale fabrication technologies evolve over the next few years, testing and reliability are expected to emerge as major roadblocks to system integration.

Most of the suggested technologies offer very high defect density (up to 10%). Increasing defect density decreases yield and with such a high defect density in nano-devices the manufacturing cost can be prohibitively high and discarding a defective nano-chip will no longer be possible. As a result, to achieve high reliability, nanoscale devices must be thoroughly tested, diagnosed and the location of defects must be found. Novel defect tolerance methods and architectures must be developed to deal with such high defect densities. For example, architectures similar to field programmable gate array (FPGA) have been suggested to use crossbars built from nanowires/nanotubes. Such crossbars can be programmed and the defects can be avoided if the location of defects is known. Similarly, in other nano-devices and architectures, a reliable system can be created using defective devices.

This book is divided into five sections. Section 1 includes five chapters that discuss different aspects of test and defect tolerance for crossbar-based nanoscale devices. The reconfiguration feature of the proposed nano-architectures provides an ability to test these devices and avoid the defective ones. Section 2 contains four chapters focusing on test, defect tolerance and reliability for QCA circuits.

There are two chapters in Sect. 3 which present methods for testing and diagnosis of realistic defects in digital microfluidic biochips. Due to the underlying mixed-technology and mixed-energy domains, biochips exhibit unique failure mechanisms and defects. Finally, Sect. 4 contains three chapters dealing with reliability of CMOS scale devices, developing nanoscale processors and future molecular electronics-based circuits.

Mohammad Tehranipoor July 2007
University of Connecticut

Contents

Section 1: Test and Defect Tolerance for Crossbar-Based Architectures
M. Tehranipoor 1

Chapter 1: Defect-Tolerant Logic with Nanoscale Crossbar Circuits
T. Hogg and G. Snider 5

Chapter 2: Built-in Self-Test and Defect Tolerance in Molecular Electronics-Based Nanofabrics
Z. Wang and K. Chakrabarty 33

Chapter 3: Test and Defect Tolerance for Reconfigurable Nanoscale Devices
M. Tehranipoor and R. Rad 63

Chapter 4: A Built-In Self-Test and Diagnosis Strategy for Chemically-Assembled Electronic Nanotechnology
J.G. Brown and R.D. (Shawn) Blanton 95

Chapter 5: Defect Tolerance in Crossbar Array Nano-Architectures
M.B. Tahoori 121

Section 2: Test and Defect Tolerance for QCA Circuits
M. Tehranipoor 153

Chapter 6: Reversible and Testable Circuits for Molecular QCA Design
X. Ma, J. Huang, C. Metra, and F. Lombardi 157

Chapter 7: Cellular Array-Based Delay-Insensitive Asynchronous Circuits Design and Test for Nanocomputing Systems
J. Di and P.K. Lala 203

Chapter 8: QCA Circuits for Robust Coplanar Crossing
S. Bhanja, M. Ottavi, S. Pontarelli, and F. Lombardi 227

Chapter 9: Reliability and Defect Tolerance in Metallic Quantum-Dot Cellular Automata
M. Liu and C.S. Lent 251

Section 3: Testing Microfluidic Biochips
M. Tehranipoor 265

Chapter 10: Test Planning and Test Resource Optimization for Droplet-Based Microfluidic Systems
F. Su, S. Ozev, and K. Chakrabarty 267

Chapter 11: Testing and Diagnosis of Realistic Defects in Digital Microfluidic Biochips
F. Su, W. Hwang, A. Mukherjee, and K. Chakrabarty 287

Section 4: Reliability for Nanotechnology Devices
M. Tehranipoor 313

Chapter 12: Designing Nanoscale Logic Circuits Based on Principles of Markov Random Fields
K. Nepal, R.I. Bahar, J. Mundy, W.R. Patterson, and A. Zaslavsky 315

Chapter 13: Towards Nanoelectronics Processor Architectures
W. Rao, A. Orailoglu, and R. Karri 339

Chapter 14: Design and Analysis of Fault-Tolerant Molecular Computing Systems
D. Bhaduri, S.K. Shukla, H. Quinn, P. Graham, and M. Gokhale 373

Index 399

List of Contributors

R. Iris Bahar
Brown University
Division of Engineering
Providence, RI 02912

Debayan Bhaduri
Virginia Polytechnic Institute
and State University
dbhaduri@vt.edu

Sanjukta Bhanja
Department of Electrical Engineering
University of South Florida
Tampa, (FL) 33620, USA
bhanja@eng.usf.edu

R.D. (Shawn) Blanton
Carnegie Mellon University

Jason G. Brown
Carnegie Mellon University

Krishnendu Chakrabarty
Duke University

Jia Di
University of Arkansas

Maya Gokhale
Los Alamos National Laboratory

Paul Graham
Los Alamos National Laboratory

Tad Hogg
HP Labs, 1501 Page Mill Road
Palo Alto, CA

Jing Huang
Department of Electrical
and Computer Engineering
Northeastern University
Boston, MA 02115
hjing@ece.neu.edu

William Hwang
Oxford University

Ramesh Karri
Polytechnic University

Parag K. Lala
Texas A&M University at Texarkana

Craig S. Lent
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
lent@nd.edu

Mo Liu
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
mliu1@nd.edu

Fabrizio Lombardi
Department of Electrical
and Computer Engineering
Northeastern University
Boston, (MA) 02115, USA
lombardi@ece.neu.edu

X. Ma
Department of Electrical
and Computer Engineering
Northeastern University
Boston, MA 02115
xma@ece.neu.edu

Cecillia Metra
Department of Electrical Engineering
University of Bologna
Bologna, Italy
cmetra@deis.unibo.it

Arindam Mukherjee
University of North Carolina
at Charlotte

J. Mundy
Brown University
Division of Engineering
Providence, RI 02912

Kundan Nepal
Bucknell University
Department of Electrical Engineering
Lewisburg, PA 17837

Alex Orailoglu
University of California, San Diego

Marco Ottavi
Department of Electrical
and Computer Engineering
Northeastern University
Boston, (MA) 02115, USA
mottavi@ece.neu.edu

Sule Ozev
Duke University

W. R. Patterson
Brown University
Division of Engineering
Providence, RI 02912

Salvatore Pontarelli
Dipartimento di Ingegneria
Elettronica
Università di Roma "Tor Vergata"
Rome, 00133, Italy
pontarelli@ing.uniroma2.it

Heather Quinn
Los Alamos National Laboratory

Reza Rad
University of Maryland
Baltimore County
reza2@umbc.edu

Wenjing Rao
University of California, San Diego

Sandeep K. Shukla
Virginia Polytechnic Institute
and State University
shukla@vt.edu

Greg Snider
HP Labs, 1501 Page Mill Road
Palo Alto, CA

Fei Su
Intel Corporation

Mehdi B. Tahoori
Northeastern University
Boston, MA

Mohammad Tehranipoor
University of Connecticut
tehrani@engr.uconn.edu

Zhanglei Wang
Cisco Systems, Inc.

A. Zaslavsky
Brown University
Division of Engineering
Providence, RI 02912

Section 1: Test and Defect Tolerance for Crossbar-Based Architectures

M. Tehranipoor

As complementary metal oxide semiconductor (CMOS) devices are scaled down into the nanometer regime, new challenges at both the device and system levels are arising. New devices and structures are being researched within the device community including regular and reconfigurable nano-crossbar arrays. Crossbar architectures are one approach to molecular electronic circuits for memory and logic applications. However, currently feasible manufacturing technologies for molecular electronics introduce numerous defects so insisting on defect-free crossbars would give unacceptably low yields. Conventional test and defect tolerance methods employed for CMOS reconfigurable devices such as FPGA are not applicable to emerging nanoscale devices due mainly to the high defect rates in nanotechnology. This section contains five chapters that present novel methods of test and defect tolerance for such high defect density nano-devices. The proposed methods try to alleviate problems such as (1) defect identification, localization and isolation, (2) defect map generation and defect avoidance, (3) test under very high defect rates, and (4) design flow under high defect rates condition.

The first chapter, entitled “Defect-Tolerant Logic with Nanoscale Crossbar Circuits”, argues that increasing the area of the crossbar provides enough redundancy to implement circuits in spite of the defects. The authors identify reliability thresholds in the ability of defective crossbars to implement boolean logic. These thresholds vary among different implementations of the same logical formula, allowing molecular circuit designers to trade-off reliability, circuit area, crossbar geometry and the computational complexity of locating functional components. These choices are illustrated in this chapter for binary adders. For instance, one adder implementation yields functioning circuits 90% of the time with 30% defective crossbar junctions using an area only 1.8 times larger than the minimum required for a defect-free crossbar. The authors also describe an algorithm for locating a combination of functional junctions that can implement an adder circuit in a defective crossbar.

Chapter 2, entitled “Built-In Self-Test and Defect Tolerance in Molecular Electronics-Based Nanofabrics”, presents a method to test nanoblocks

and switchblocks in a nano-architecture and identify the location of defects. The authors in this chapter propose a built-in self-test (BIST) procedure for nanofabrics implemented using chemically assembled electronic nanotechnology. Several fault detection configurations are presented to target stuck-at faults, shorts, opens, and connection faults in nanoblocks and switchblocks. The detectability of multiple faults in blocks within the nanofabric is also considered. The authors also present an adaptive recovery procedure through which defect-free nanoblocks and switchblocks in the nanofabric-under-test can be identified. The proposed BIST, recovery, and defect tolerance procedures, in this chapter, are based on the reconfiguration of the nanofabric to achieve complete fault coverage for different types of faults. It is shown that a large fraction of defect-free blocks can be recovered using a small number of BIST configurations. The authors also present simple bounds on the recovery that can be achieved for a given defect density. Simulation results are presented for various nanofabric sizes, different defect densities, and for random and clustered defects.

The third chapter entitled "Test and Defect Tolerance for Reconfigurable Nanoscale Devices" presents a solution to dealing with issues such as storing large defect map size and per chip placement and routing. In this chapter, the authors present a new test method in addition to novel defect avoidance methods for reconfigurable nanoscale crossbar-based devices. The proposed defect tolerance methods are independent on defect map and avoid per chip placement and routing. The test procedure proposed in this chapter is a built-in self-test method that tests the function implemented on a logic block instead of testing the block itself. The method avoids generation of large defect map and speeds up the configuration process. Probabilistic analyses are presented to show efficiency of the methods in avoiding defects in such high density devices. Two simulation programs are developed and several experiments are performed on MCNC benchmarks to evaluate the proposed methods in terms of yield and timing requirements.

Chapter 4, entitled "A Built-in Self-test and Diagnosis Strategy for Chemically Assembled Electronic Nanotechnology", illustrates that the highly defective nanosclae circuits will require a completely new approach to manufacturing computational devices. In order to achieve any level of significant yield, it will no longer be possible to discard a device once a defect is found. Instead, a method of using defective chips must be devised. A testing strategy is developed for chemically assembled electronic nanotechnology (CAEN) that takes advantage of reconfigurability to achieve 100% fault coverage and nearly 100% diagnostic accuracy.

Finally, the fifth chapter in this section entitled "Defect Tolerance in Crossbar Array Nano-Architectures" presents an application-independent defect-tolerant design flow to minimize customized post-fabrication design efforts to be performed per chip. In this flow, higher level design steps are not needed to be aware of the existence and the location of defects in the chip. Only a final mapping step is required to be defect-aware. Application independence of

this flow minimizes the amount of per chip design steps, making it appropriate for high volume production. The manufacturing yield of molecular crossbars is analyzed under different defect distribution models. The authors report on the size of the minimum crossbar to be fabricated such that a defect-free crossbar of the desirable size can be found with a guaranteed manufacturing yield.

Chapter 1: Defect-Tolerant Logic with Nanoscale Crossbar Circuits

T. Hogg and G. Snider

1 Introduction

Molecular electronics offers the possibility of significantly denser circuits than current lithography-based manufacturing. Achieving this potential requires circuit designs exploiting the capabilities of molecular electronics while compensating for limitations of current fabrication approaches, particularly defects. Creating such circuits in spite of fabrication defects requires economic trade-offs. For instance, accepting lower yields or improving fabrication could reduce defect rates, but increase production cost. Algorithmic configuration strategies for defect-tolerant systems [28], discussed in this chapter, provide higher defect tolerance, but add to manufacturing cost with the additional testing and analysis required. Evaluating this strategy requires determining what defect rates are tolerable *at all*, i.e., is there some level of defects beyond which constructing circuits is not practical? If we can accommodate defects, how much area overhead is required and how is it affected by choice of circuit geometry? This chapter is an empirical exploration of these questions.

We consider a particular type of molecular circuit, the crossbar described in Sect. 2. For nanoscale crossbars, the main type of defect is that introduced during manufacture (so-called "static defects") rather than during operation. This is reasonable for plausible fabrication technologies, which involve high temperatures during manufacture, and hence a relative ease of introducing defects, but low temperature during operation, with much less chance of creating new defects. In this situation, an appropriate systems architecture consists of a compiler to arrange for desired circuit behaviors by only using correctly functioning components of a given crossbar circuit, as determined from a testing phase after manufacture [15]. This approach of avoiding known defects gives a defect-tolerant system architecture. It contrasts with methods dealing with faults that may appear during the operation of the device, perhaps intermittently, e.g., using majority votes from replicated hardware [32].

This leads to the central question addressed in this chapter: given a defect rate and a certain size crossbar, how likely is it we can find a way to implement a particular logical formula in the crossbar? Determining whether such a circuit exists, and if so, finding one, is a combinatorial search problem. Thus a related question is the computational difficulty for the compiler to identify an implementation, or conclude no implementation is possible. For a given desired circuit and crossbar size, decreasing the defect rate will generally require more difficult and costly manufacturing. On the other hand, increasing the allowable defect rate will make it less likely the desired circuit can be implemented and can also result in longer runtimes for the compiler to identify a way to implement the circuit while avoiding the defects.

Furthermore, a logical formula can be written in various logically equivalent forms, e.g.,

$$(a \text{ OR } b) \text{ AND } c$$
$$(a \text{ AND } c) \text{ OR } (b \text{ AND } c)$$

are logically equivalent. These *rewrites* can involve different numbers of terms, and hence require different crossbar areas and shapes to implement. They can also differ in their likelihood of being implementable on crossbars with defects.

After describing the molecular crossbar hardware and the defect model evaluated in this chapter, we consider a simple example of implementing AND logic gates. We then turn to a more interesting circuit: the binary adder. We first describe two approaches to implementing adders using crossbars, and then show their feasibilities in the face of defects. We thus show how crossbar architectures can implement logic circuits, even with numerous manufacturing defects.

2 Crossbar Architecture

The crossbar architecture is a general approach for molecular circuits [2–4, 6, 14, 19, 21, 25, 30]. A molecular crossbar consists of two parallel planes of molecular wire arrays separated by a thin layer of a chemical species (called the "interlayer") with particular electrochemical properties (Fig. 1). Each plane consists of a number of parallel molecular wires (also called "nanowires"), with each wire in a plane being of the same type. The wires in one plane cross the wires in the other plane at a right angle. The region where two perpendicular wires cross is called a junction or crosspoint. Depending on the nature of the interlayer and nanowires, each junction may be configured to implement an electronic device, such as a resistor, diode or field effect transistor [22], or may be left unconfigured so the two crossing wires do not interact electrically. We consider crossbars whose junctions can only be configured as either resistors or diodes, since those are easier to fabricate with current technology than configurable transistors.

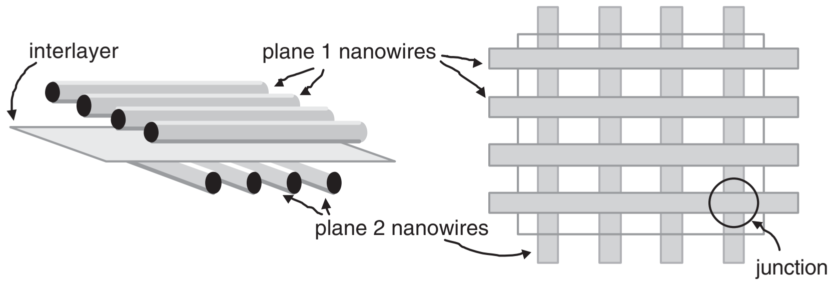


Fig. 1. Schematic view of a molecular crossbar from two different perspectives. Each junction may be independently configured to behave as an electronic device

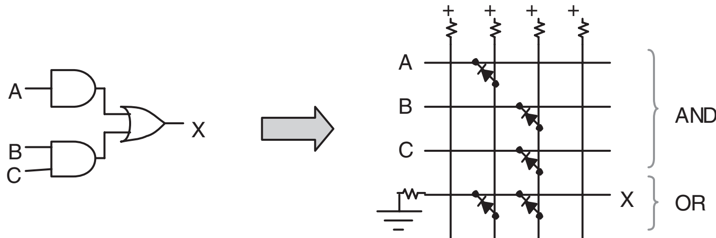


Fig. 2. Implementing the AND/OR function $X = A + BC$ with a diode crossbar and resistors

The crossbar structure is an attractive architecture for molecular electronics since it is relatively simple and inexpensive to fabricate using either chemical self-assembly or nanoimprint lithography.

By suitable selection of the type of connections at each crosspoint (e.g., no connection, or a diode in one direction or the other), crossbars can be set to evaluate any logical formula expressed as a combination of AND and OR operations. Figure 2 shows one example. To see this, consider the output wire, labeled "X". It is connected to ground through a resistor, and via diode junctions to the second and third vertical wires. If both vertical wires are at low voltage ("off"), then the output wire X will also be at low voltage due to its connection to ground. On the other hand, if either of the connected vertical wires is at high voltage ("on"), the diode connection from the high voltage vertical wire(s) will give a high voltage to the output wire (since, by design, the diode resistance in the forward direction is much smaller than the resistor connecting the output wire to ground). If only one of the vertical wires is on, the high resistance of the diode junction in the reverse direction ensures that the output wire remains at high voltage. Thus this combination of resistors and diode connections makes the output X equal to the logical-OR of the inputs on the two vertical wires. Similarly, the connections from the inputs A, B and C implement logical-ANDs. Typical voltage drop across a forward-biased diode is about 0.6 V, and resistors are about 100 kΩ.

The crossbar of Fig. 2 connects each column, through a *pullup* resistor, to a positive voltage source. With the diode directions shown here, each column implements the logical-AND of its inputs (the horizontal wires). Each output row, connected to ground through a *pulldown* resistor, implements the logical-OR of the columns connected to it through diode junctions. Although this is not the only way to configure crossbar circuits, it provides a simple functional form in which each output is the logical-OR of a number of terms, each of which is the logical-AND of some inputs.

Diode/resistor logic cannot implement logical inversion (i.e., a NOT gate). However, by presenting the circuit with two wires for each input (i.e., one wire representing the true input value, the other representing its complement), the crossbars can produce internal signals in both the original and complemented forms. Combining these signals using just AND and OR operations then allows evaluating any logical formula. The complemented inputs to the crossbar are readily produced by the external circuit, fabricated using conventional technology, to which the crossbar is connected for input and output. Thus by doubling the number of wires and presenting all primary inputs in both true and complemented forms, the diode crossbar architecture can implement any logical formula just using combinations of AND and OR operations, as illustrated in Fig. 3.

Inputs and outputs of a nanoscale circuit ultimately need to be connected to the external, sub-micron world. One might use an approach similar to Likharev's architecture [31] to accomplish this, by having the external wires from the crossbar to spread out to a spacing large enough to match lithographically. In such an architecture, generally only a small fraction of the crossbar junctions (e.g., about 1%) are used. Thus, such an increased spacing reduces the high density benefit of molecular electronics. Nevertheless, such connection limitations allow tolerating high defect rates [31].

Alternatively, the high density of molecular electronics can be maintained by using nanoscale latches [24] at the inputs for driving the logic functions,

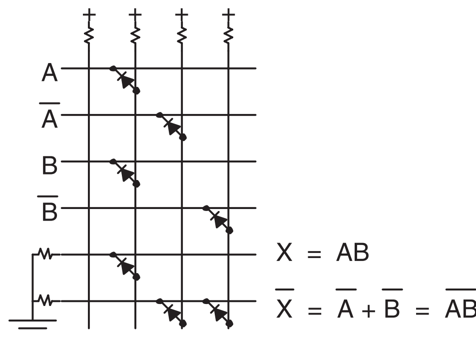


Fig. 3. DeMorgan's theorem allows generating a given logical AND/OR function along with its complement. This generally requires all input signals to be presented in both original and complemented forms

and at the outputs for signal regeneration. We consider this approach in our study, to examine defect tolerance of circuits fully exploiting the density improvement of molecular electronics, though with less tolerance for defects than sparser circuits. In this case, inputs can be provided via demultiplexer circuits [7, 8, 16, 20]. While demultiplexers require significant additional area for the small circuits discussed in this chapter, Rent's Rule states that the number of external connections scales approximately with the square root of the number of components. So for large circuits of practical interest, the vast majority of inputs and outputs on a single logic crossbar will be to other nanocrossbars. Such larger circuits face the same issues of defect-tolerant design as discussed in this chapter.

3 Model of Crossbar Defects

We restrict attention to defects leading to inoperative connections (i.e., with no ability to activate them to make diode connections), rather than defects that break or short out a wire, or prevent routing the output of one gate to the input of another. Moreover, for a given defect rate p, we assume errors occur independently, i.e., without any spatial correlation among defect locations.

Many researchers have explored the problem of finding such defects in crossbars with high defect rates [1, 27, 33].

In this scenario, we can test the circuits to determine which crosspoints are defective, and then use the remaining ones to implement the circuit. That is, a compiler uses the required logic formula and a table of defects to find a way to implement the formula.

Our focus is on crossbars whose configurable junction devices (e.g., diodes) have fixed parameters if they are functional (e.g., resistances in forward and reverse directions). A complementary analysis to that presented here could examine the consequences for circuit performance of various values of these parameters, as has been applied to a different adder implementation than discussed in this chapter, namely using a look-up table to evaluate all the combinations of inputs [34].

We do not consider errors in connections to external circuits. These include the inputs and outputs to the crossbar device, any feedback connections or inverters that may be needed to create complementary inputs and resistive connections to larger wires for the pullup and pulldown resistors. Other work has addressed connecting molecular electronics to larger scale circuits through the use of demultiplexer circuits with defects [7, 8, 16, 20].

4 An AND Gate

A simple logic circuit created from the crossbar architecture is the logical AND of k inputs. One implementation is as a single k-input AND gate, i.e., using k connections to a single output wire. Another implementation is to decompose

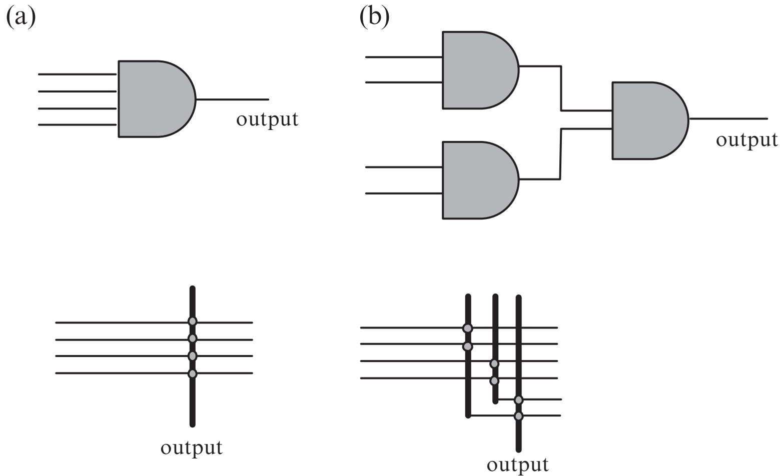


Fig. 4. Logic gates: a 4-input AND, and the same function using 2-input AND gates. Also shown is an implementation of these circuits using parts of a crossbar network

the AND into a circuit of several AND gates with fewer inputs. For example, $k - 1$ 2-input AND gates connected in a tree structure implements a k-input AND when k is a power of 2, as illustrated for $k = 4$ in Fig. 4.

For this example we suppose the assignments of input signals to input wires are fixed, e.g., either from external connections or from outputs of another part of a larger overall circuit. More generally, the circuit design could also involve searching for a suitable choice of these input wires among a larger number of rows in the crossbar network, as we discuss for the adder circuit in Sect. 5.

We also suppose the only defects in the crossbar are those preventing configuration as logical ANDs, rather than also considering other defects such as in the routing connections shown in Fig. 4. If each connection in the crossbar is defective with independent probability p, the probability a given column wire in the crossbar can be used to form a gate with k given inputs (i.e., row wires) is $(1-p)^k$. Thus in a crossbar with N columns, the probability that at least one column will be able to implement the k-input gate is

$$P_{\text{gate}}(k, N) = 1 - \left(1 - (1-p)^k\right)^N . \tag{1}$$

Computing the logical AND of k inputs using a single column, the probability to find a functional circuit is $P_{\text{circuit}} = P_{\text{gate}}$. If we use $k - 1$ 2-input gates to compute the same logical value, the probability to find a functioning circuit is complicated due to the requirement that each gate must use a distinct column. We can illustrate the consequences of different implementations by using simple bounds on the probability. For an upper bound on the probability, we ignore the requirement for distinct columns. In this case, each gate's implementation is independent of all the others giving the upper bound

$$P^{\text{upper}}_{\text{circuit}}(k, N) = P_{\text{gate}}(2, N)^{k-1}. \tag{2}$$

For a lower bound on the probability, we attempt to implement the gates in a fixed order without regard for difficulties implementing subsequent gates. This results in a lower bound for the probability since a failure to find an implementation with this procedure may still allow constructing a circuit by backtracking to make another choice for one of the earlier gates in the sequence. In this procedure, the first gate can use any of the N columns, allowing implementation with probability $P_{\text{gate}}(2, N)$. The second gate cannot use the column already selected for the first gate, allowing implementation with the somewhat smaller probability $P_{\text{gate}}(2, N-1)$. Each additional gate has one fewer column to use. Continuing with all $k-1$ 2-input gates gives the lower bound on probability to construct a circuit of

$$P^{\text{lower}}_{\text{circuit}}(k, N) = \prod_{i=0}^{k-2} P_{\text{gate}}(2, N-i) \tag{3}$$

Figure 5 shows the behavior of these expressions as a function of defect probability p. In this example, with low p values, both circuits for evaluating the logical expression are likely to be constructible. As p increases, the chance of finding a single-gate circuit drops more rapidly: it is easier to find seven column wires with functioning connections for 2-input gates than to find one column to implement a single 8-input gate. This difference becomes more extreme as the size of the circuits increases. The figure also illustrates the

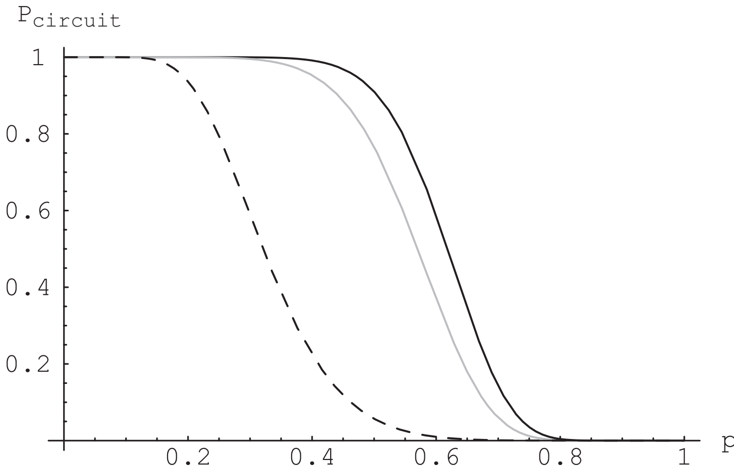


Fig. 5. Probability to be able to find a correct circuit in a crossbar with $N = 15$ columns as a function of defect probability p. The *dashed curve* is for a single 8-input AND gate, and the *solid curves* are for the upper and lower bounds on equivalent logical expression made with seven 2-input gates (*black and gray curves*, respectively)

threshold nature of the behavior: most of the drop in success probability occurs over a short range of p values.

Another way to characterize the ability to find functional circuits in spite of defects is by the number of additional columns, or, equivalently, increased circuit area, necessary to give at least, say, a 95% probability of being able to find a functioning circuit. For the single k-input gate, inverting (1) gives the minimum number of columns N required to have a success probability at least α. Similarly, inverting (2) and (3) give corresponding bounds on the number of columns required.[1]

With N columns, the circuit for the k-input gate has total area kN. With $k-1$ 2-input gates, each gate has 2 inputs (distinct from all the rest) and the connections among the gates do not add to the overall area, as seen in Fig. 4. Thus in this case the circuit area is $2(k-1)N$.

Figure 6 illustrates the behavior of the area requirements for the two implementations of the logical AND of k inputs. When p is low, both methods have a high chance of success (as also seen in the threshold behavior illustrated in Fig. 5). In this case, the smaller size of the k-input AND gate is

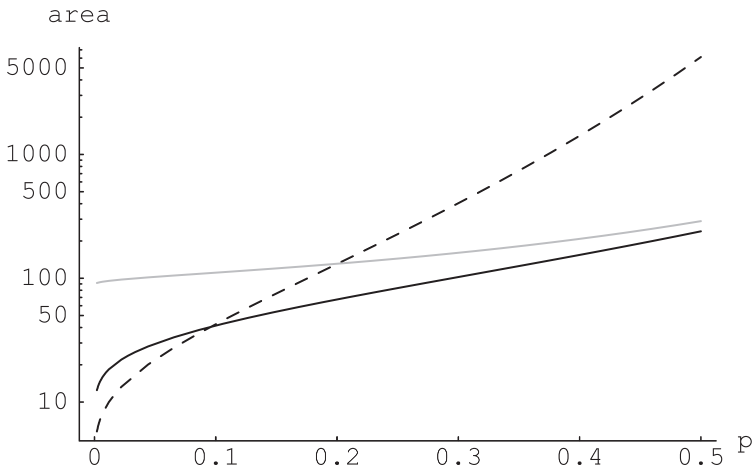


Fig. 6. Logarithmic plot of area required to have at least 95% probability to be able to find a correct circuit as a function of the defect probability p. The *dashed curve* is for a single 8-input AND gate, and the *solid curves* are upper and lower bounds for the equivalent logical expression made with seven 2-input gates (*gray and black curves*, respectively). Actual circuits must have an integer number of columns, resulting in slightly larger areas and step-functions in these plots, but with qualitatively the same behavior

[1] Actual circuits have integer numbers of columns. So the actual minimum number of columns is the smallest integer greater than or equal to the values obtained by inverting these equations. This slight increase in the number of columns and will give somewhat higher success probabilities than α.

more important than the slightly higher success probability with the combination of 2-input gates. As p increases, the success probability for the k-input gate drops more rapidly, leading to faster growth in area required, than the 2-input gates. Thus for large circuits, it is better to use more gates with few inputs than fewer gates with more inputs.

This discussion considers only two possible implementations of the logic formula. Additional possibilities include using a mixture of gates with different numbers of inputs, or combining 4-input, rather than 2-input, gates (so a k-input AND operation would be built from $(k-1)/3$ 4-input AND gates). This give qualitatively similar behaviors to those shown in Fig. 5 and additional choices for circuit implementations.

5 Adder Circuits

We now consider the mapping of small adder circuits onto crossbars. There are several ways to implement such circuits, with differing sensitivities to defects [18]. Figure 7 shows a straightforward 3-bit, ripple-carry adder that is essentially a direct translation of the logic circuit shown at the top, producing four output bits $S_0 \dots S_3$ representing the sum of two 3-bit numbers. For instance the bottom wire of the crossbar and the leftmost two columns compute the least significant bit of the sum, S_0, as $S_0 = A_0\overline{B_0} + \overline{A_0}B_0$. This logical formula is equivalent to the exclusive-OR of the two least-significant bits of the numbers to be added, i.e., A_0 and B_0.

Because this implementation uses several levels of logic, some of the intermediate output signals must be fed back to some of the inputs, possibly requiring signal regeneration in the process to compensate for degradation due to diode voltage drops. The signal restoration can be accomplished at the nanoscale using, for example, a restorative latch [23, 24]. The circuit uses 12 input wires: each of the two numbers to be added has 3 bits, and each bit must be presented as original and complementary values. The circuit has 4 outputs. The addition of the feedback signals gives a total of 30 rows. Forming the required logical operations on these values uses 25 columns, as shown in Fig. 7. This implementation, which uses 78 junctions configured as diodes, thus requires a minimum crossbar area of $30 \times 25 = 750$ junctions.

A second approach to the 3-bit adder is shown in Fig. 8. Here the entire circuit uses only two logic levels. It requires only enough rows to handle the input and output wires, i.e., 16 rows. The circuit requires 31 column wires to perform logic operations on the inputs, for a minimum crossbar area of $31 \times 16 = 496$. Again S_0 is computed as $S_0 = A_0\overline{B_0} + \overline{A_0}B_0$, using the bottom wire of the crossbar and the second and third columns from the right. This implementation eliminates the need for feedback and requires less area. On the other hand, it requires more diode junctions (147) and uses a greater number of diodes along some of the vertical and horizontal wires. For instance, the circuit in Fig. 7 never uses more than four diodes on any wire, while the circuit

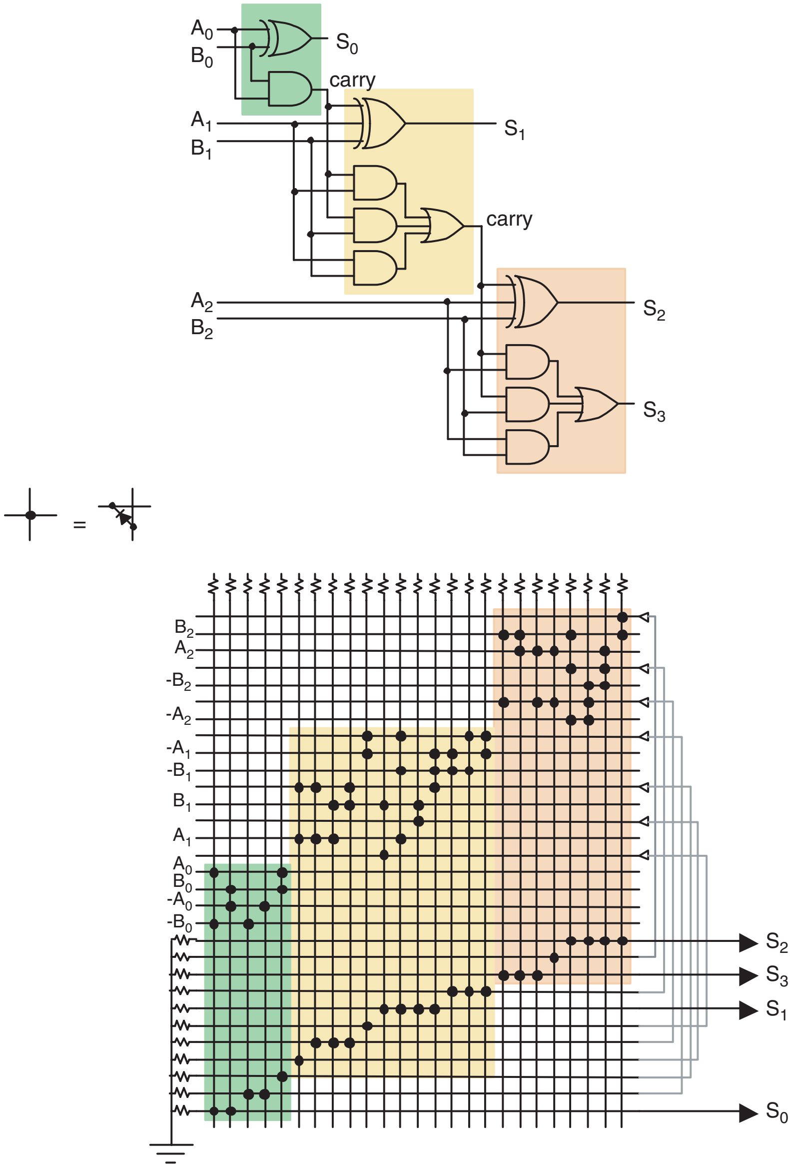


Fig. 7. A 3-bit adder which adds two 3-bit numbers (denoted as the bits $A_2A_1A_0$ and $B_2B_1B_0$, respectively) to produce a 4-bit sum (with bits $S_3S_2S_1S_0$). The ripple-carry logic implementation (*top*) translates directly to a diode crossbar implementation (*bottom*) using feedback from some of the outputs to the inputs (*gray lines*). Regenerative buffers (*left pointing triangles*) between stages regenerate signals degraded by diode and resistor voltage drops. The input wire marked $-A_0$ gives the complement of input bit A_0, and similarly for the other inputs. The carry bit between successive stages of the crossbar implementation must be presented in both original and complemented forms

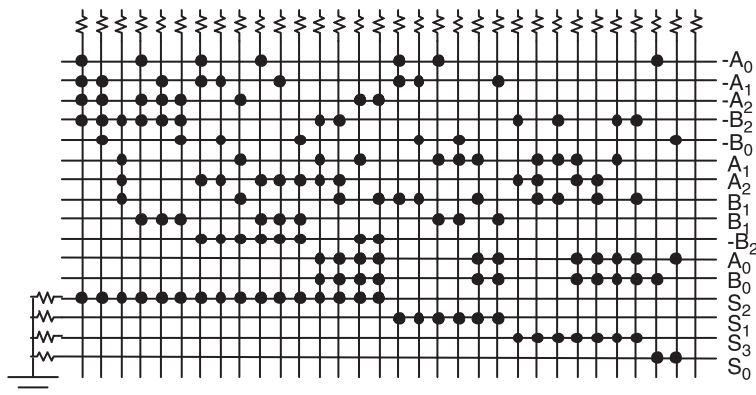


Fig. 8. A 3-bit adder implemented as 2-level logic in a single diode crossbar. Although this approach uses more diodes, it consumes less area, avoids the feedback and regenerative buffers between stages, and will likely offer less propagation delay. Inputs and outputs are labeled as in Fig. 7. The rightmost column wire is not used in this circuit

in Fig. 8 requires as many as 16. Thus we can expect this circuit, packing more diodes in a smaller area, will be more difficult to implement on a crossbar with defects than that of Fig. 7.

Both of these adder circuits produce output bits corresponding to the sum. This is suitable when these circuits are considered as stand-alone components whose outputs are delivered to an external circuit composed of conventional technology (which can implement inversion). If instead the crossbar adder is to be used as part of a larger molecular-scale circuit, the adder will need to be extended to also produce complement values of each of the output bits so subsequent crossbars, using the results of the adder, will have access to both original and complement values for their inputs. Such extensions are readily included, as described with the discussion of Fig. 3, and will result in doubling the number of output wires, as well as additional logic computations within the crossbar. For simplicity, we focus on the adders treated as stand-alone circuits without the need for complement values for the outputs.

In the remainder of this section, we first describe the algorithm used to find a mapping of an adder circuit implementation to a crossbar with a known set of defects. We then use this algorithm to produce implementations on simulations of defective crossbars to identify their ability to give functional adder circuits.

5.1 Allocation Algorithm

The diode/resistor fabric which we map onto is modeled as a set of four crossbars tiled together to form a mosaic, illustrated schematically in Fig. 9 and more explicitly with the example circuits of Figs. 2 and 3. The pullup and

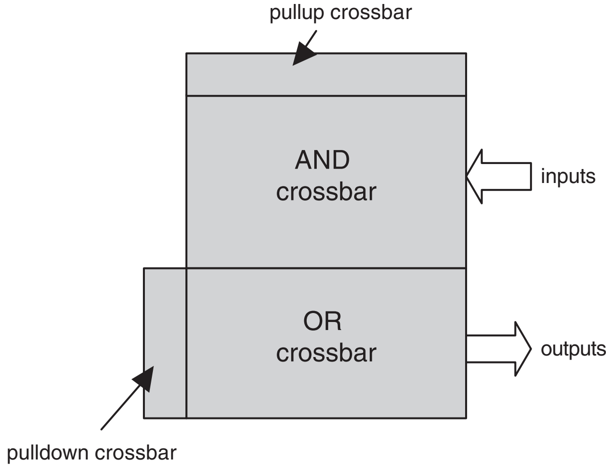


Fig. 9. Model of diode array as a set of four connected crossbars. The AND and OR crossbars have configurable diode junctions, while the pullup and pulldown crossbars have configurable resistor junctions. Any junction in any crossbar may be defective, though the defect rate for junctions in the pullup and pulldown crossbars is much lower than for other junctions

pulldown "crossbars" are only one wire tall and wide, respectively, but the resistors there can be defective just like the diodes in the diode crossbars, so it simplifies allocation to use a single model for all of the components. However, the consequence of a defective pullup or pulldown resistor is to disable the entire column or row, respectively, to which it is connected. Fortunately, these resistors, at the edge of the crossbar, are formed from junctions between the nanowires and much larger, microscale, wires. Thus the junction area per device is significantly larger than that for the diode junctions used in the AND and OR crossbars. This increased junction area means the chance of a defective resistor is far smaller than having a defective diode.

Even though the AND and OR crossbars share the same junction type and could be represented with a single crossbar, it is helpful to keep them separate since input signals may only be bound to horizontal wires entering the AND crossbar and output signals only bound to horizontal wires leaving the OR crossbar. The allocation problem, then, is to implement a circuit in the crossbars given a set of defective junctions in each. The allocation for the adder circuits also considers alternate choices for the input and output wires. However, input connections can only be made among wires preselected to be part of the AND crossbar, and output connections only among wires in the OR crossbar.

The allocation problem cannot be divided into separate crossbar allocation subproblems when the crossbars contain defects because a particular allocation of resources in one tile may actually preclude a successful allocation in another. We must also respect other constraints, such as input/output signal

restrictions (in the case where the crossbars are embedded in a larger system) and asymmetric junctions, where component direction or polarity (such as for diodes) must be respected. We address the problem by searching globally for a solution that meets all constraints (defect avoidance, input/output constraints, junction polarity, and crossbar interaction) simultaneously.

Our allocation algorithm uses graphs with annotated edges and nodes to represent both the original circuit to be mapped onto a set of crossbars as well as the crossbars themselves. Figure 10 shows how a crossbar is represented with a graph: a wire in the crossbar is represented by a node in the graph, and a junction is represented by an edge between the two nodes representing the wires that define the junction. A perfect crossbar (left) has an edge for every junction. A defective crossbar (right) has edges only for usable junctions.

Graphs for multiple crossbars are constructed by first creating a graph for each individual crossbar (Fig. 11, *top left*) and then interconnecting them (*top right*). The resulting graph may then be (*optionally*) optimized by merging identical nodes (*bottom*).

Allocation is accomplished by (1) creating graphs representing the desired circuit and compound crossbars; and (2) searching for an embedding or monomorphism between the circuit graph and the compound crossbar graph.

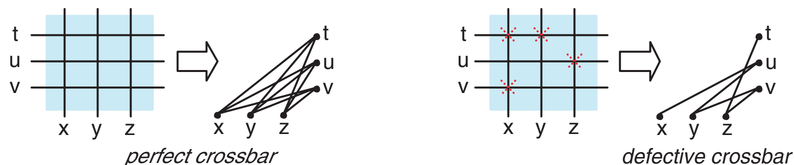


Fig. 10. Representing a crossbar with a graph. Wires and junctions in the crossbar correspond to nodes and edges of the graph, respectively. Defective junctions are shown marked with an “X”

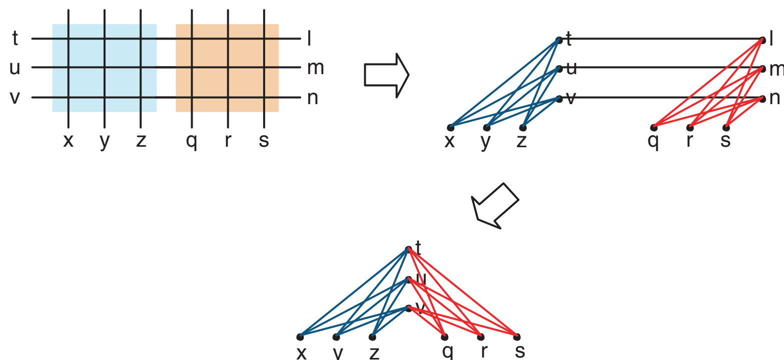


Fig. 11. Representing composite crossbars with a graph. Edges are colored to represent the functionality of the junction that each one represents, since each crossbar might have different functionality

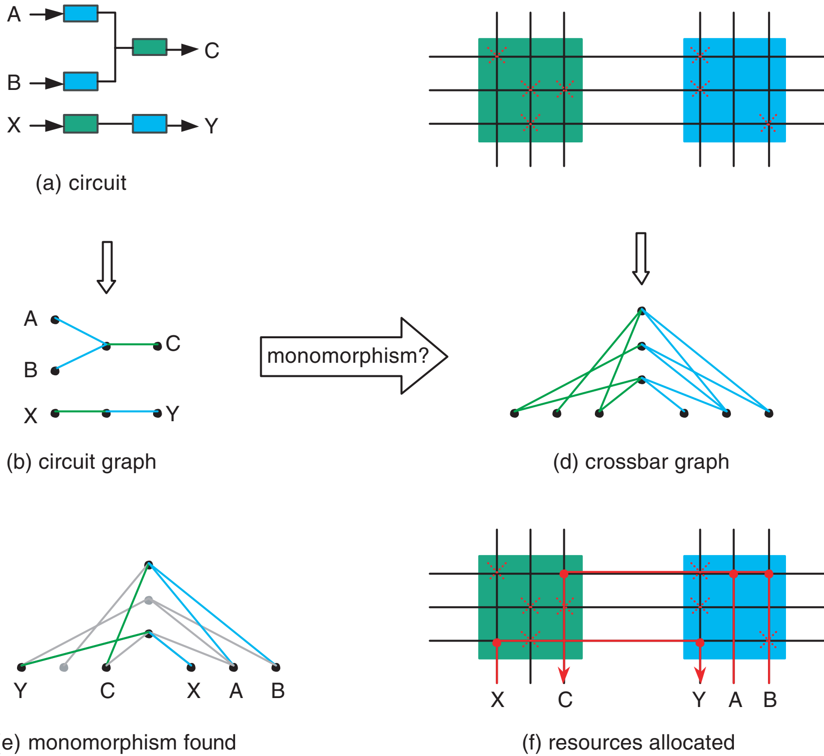


Fig. 12. Resource allocation: searching for a monomorphism between a circuit graph and a crossbar graph. The corresponding algorithm steps are described in the text

A graph monomorphism is the embedding of a small graph into a larger one, by specifying the correspondence between the nodes of the small graph and a subset of nodes in the larger graph so that the small graph forms a subgraph of the larger one. Figure 12 illustrates this in detail. The steps for allocation are:

1. For the desired circuit (Fig. 12a) create a circuit graph (Fig. 12b) representing it: wires and junctions in the circuit are represented by nodes and edges in the circuit graph, respectively.
2. For the desired target compound crossbar (Fig. 12c), create a compound tile graph (Fig. 12d) representing it. As in circuit graph case, wires and junctions in the crossbars are represented by nodes and edges in the compound tile graph, respectively. A defective junction in a crossbar is represented by the absence of its corresponding edge in the crossbar graph.
3. Annotate the edges of the circuit graph and the crossbar graph with annotations representing the functionality of those edges (junctions in the circuit represented by the graph). For example, edges in both graphs representing resistors would all be tagged with identical annotations.

4. Annotate the nodes of the circuit graph and crossbar graph with annotations to constrain matching between the two graphs. As will be shown later, this is done to either (a) enforce input/output constraints between the desired circuit and other circuitry that has been or will be mapped to other areas of a large compound tile graph; or (b) enforce directionality constraints on asymmetric junctions, such as diodes, that must have, for example, an input delivered on a horizontal wire and an output driven on a vertical wire; or (c) enforce both.
5. Search for a monomorphism (Fig. 12e) between the annotated circuit graph and the annotated target crossbar graph to do allocation (Fig. 12f), subject to the constraints that node and edge annotations must match. In other words, a node in the circuit graph can only be matched with a node in the crossbar graph if they both have identical annotations or both have no annotations. Similarly, edges can only be matched if they both have identical (or non-existent) annotations. Algorithms for searching for a graph monomorphism are well known [5,10,29]. They are efficient for problems with many solutions, but can be computationally expensive when there are few solutions.
6. Use the monomorphism to complete the allocation or mapping of wires and junctions in the desired circuit graph onto wires and junctions of the crossbar. For example, a node, A, in a circuit graph matched to a node, B, in the crossbar graph will be used to allocate the crossbar wire represented by B in the crossbar graph to carry the signal represented by A in the desired circuit. Similarly, an edge, X, in a circuit graph matched to an edge, Y, in the crossbar graph will be used to allocate the junction in the crossbar represented by Y in the crossbar graph for the electrical component represented by X in the desired circuit.

Edges in the circuit graph and crossbar graph are "colored" to reflect the component and junction functionality, respectively. These edge colors are additional constraints when searching for a monomorphism or embedding, since an edge in the circuit graph may only be matched with an edge in the compound tile graph with the same color. The edge coloring and the implied matching constraint are referred to as "edge annotation." Similarly, nodes may be annotated with the same matching constraint, namely that a node in a circuit graph may only be matched to a node in the crossbar graph with the same annotation. As shown in Fig. 13, this is useful to meet input/output constraints for a circuit. In this example the simple circuit (Fig. 13a) is to be mapped onto a set of crossbars (Fig. 13c). External constraints may require mapping the A signal onto a vertical wire and the B signal onto a horizontal wire. This constraint can be met by tagging the computation graph vertices (Fig. 13b) and the crossbar graph vertices (Fig. 13d) with legal matching tags. In this case, the input signal, A, is tagged with the α tag as are the three vertical wires in the crossbar graph. Similarly, B is tagged with a β tag as are the three horizontal wires in the crossbar graph. When a monomorphism is found (Fig. 13e), it meets the input/output constraints for the allocation (Fig. 13f).

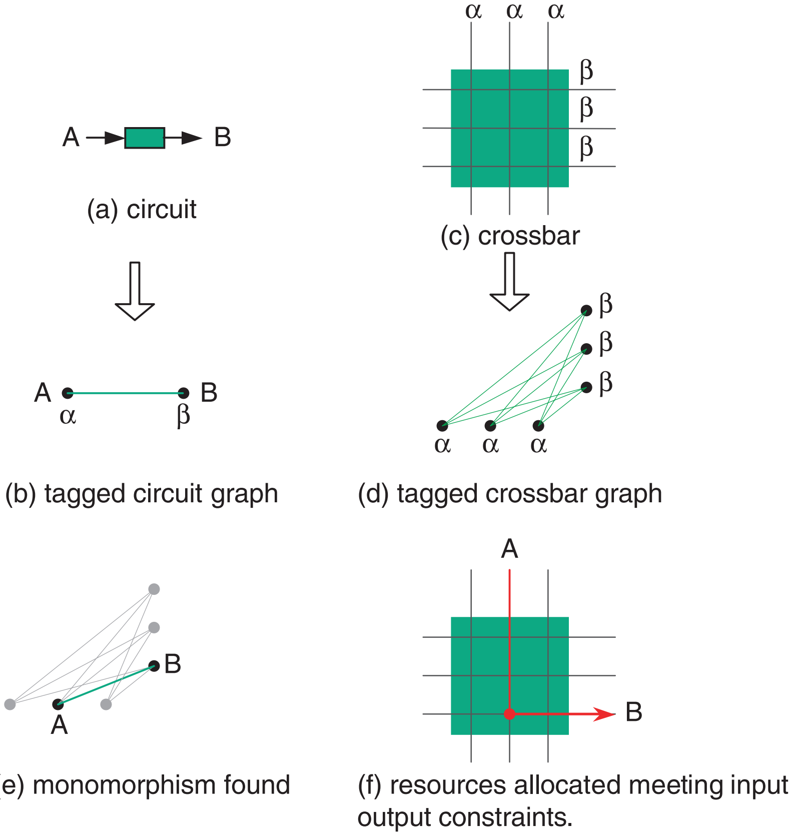


Fig. 13. Nodes (*wires*) and edges (*junctions*) may be annotated with a matching constraint

Node annotation is also useful for asymmetric junctions. The diode in Fig. 14 may only be configured correctly in the target compound tile in one direction. Annotating the vertices of the circuit graph and the compound tile graph appropriately assures the diode will be allocated with the proper orientation.

This allocation algorithm is a *complete* search method: if it does not find a possible circuit implementation in the crossbar with given defects, then no such implementation exists. In practice, the computational cost of such search methods can be prohibitive, especially for circuits with many components. In that case, one could instead use an *incomplete* search method, which often solves combinatorial problems more rapidly than complete methods, but offers no guarantee that failure to find a solution means no solution exists. For the simulation results discussed below, we employed an incomplete search method obtained by simply imposing a bound on the number of graph matching attempts allowed for the allocation algorithm. If a match is not found within

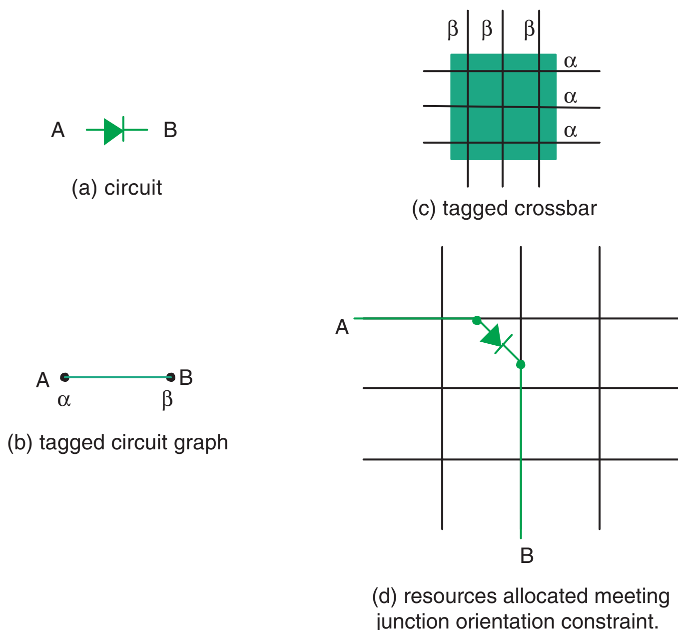


Fig. 14. Node annotation to support allocation of asymmetric junctions

this number of attempts, we consider the defective crossbar to be a failure. In a mass-production manufacturing context, the choice of the bound on the algorithm results in a trade-off between increasing the computing and testing time spent determining whether a crossbar is functional and decreasing the yield of functional circuits.

5.2 Simulation Results for Adder Circuits

To examine the behavior of implementing adder circuits on defective crossbars, we created a number of simulated test cases. Specifically, for a given adder implementation (e.g., single or multiple stage) and crossbar size, we mark each junction of the AND and OR crossbar as defective independently with probability p. Because the pullup and pulldown resistors have much lower defect rates, for simplicity, we restrict our attention to cases with no defective resistors. We then run the allocation algorithm, recording whether it found an allocation (within at most 30 s of CPU time, corresponding to about 7×10^7 graph matchings) and the number of steps required to reach a decision. We repeat this procedure on new crossbars with randomly selected defects (using the same parameters).

The limit on search time for the allocation algorithm was significantly larger than the typical number of matchings needed in the cases that produced a successful match. From a pragmatic standpoint, the need to rapidly

test circuits after fabrication would preclude spending an inordinate amount of time trying to distinguish a crossbar with a possible, but difficult to find, circuit implementation from one with no possible implementations. Thus, imposing a bound on CPU time is a simple approach to avoiding this excessive search cost, with the tradeoff of slightly reducing the yield.

Threshold Behavior

From the set of simulation trials, we estimate P_{circuit}, the probability an implementation exists for a set of parameters. Suppose we successfully find an allocation s times out of n trials. Because each trial uses an independently generated crossbar, the probability to observe s successful circuits out of n trials is the binomial distribution $\text{Bi}(n, s; P_{\text{circuit}})$ where

$$\text{Bi}(n, k; p) \equiv \binom{n}{k} p^k (1-p)^{n-k} \tag{4}$$

Thus the likelihood that P_{circuit} equals the value f is proportional to $f^s(1-f)^{n-s}$. Maximizing this quantity gives $f = s/n$ as the maximum-likelihood estimate of P_{circuit}. Evaluating the range of f values accounting for, say, 95% of the likelihood indicates how well our simulation determines the value.

Figure 15 is an example of the behavior of implementing a 3-bit adder circuit, using the two rewrites discussed in Sect. 5: a single stage and three stages. In this and subsequent figures, each point for the 1-stage adder is the result of 50 simulation trials at the corresponding defect rate p; and each point for the 3-stage adder is from 100 trials. We see a threshold behavior as P_{circuit} drops abruptly over a fairly short range of p values. The 3-stage adder can tolerate higher defect rates than the single-stage implementation.

As p increases, the probability of finding a given circuit monotonically decreases. As a simple summary of the results from multiple sets of trials, each using a different value of p to generate defective crossbars, we use a two-parameter sigmoidal form relating P_{circuit} to p. A sigmoid allows matching the location and steepness of the threshold behavior to the simulation results, and incorporates the monotonicity in the relation between P_{circuit} and p. As one specific choice for a sigmoidal function, let

$$S(x) = \frac{1}{1 + e^{a(x-b)}}. \tag{5}$$

Here the parameter a determines the sharpness of the threshold and b its location. Since P_{circuit} is zero when $p = 1$, and $P_{\text{circuit}} = 1$ when $p = 0$ (provided the crossbar is at least the minimum size required for the circuit), we shift and scale this sigmoid to match these extremes. Thus we use

$$P_{\text{circuit}} = \frac{S(p) - S(1)}{S(0) - S(1)} \tag{6}$$

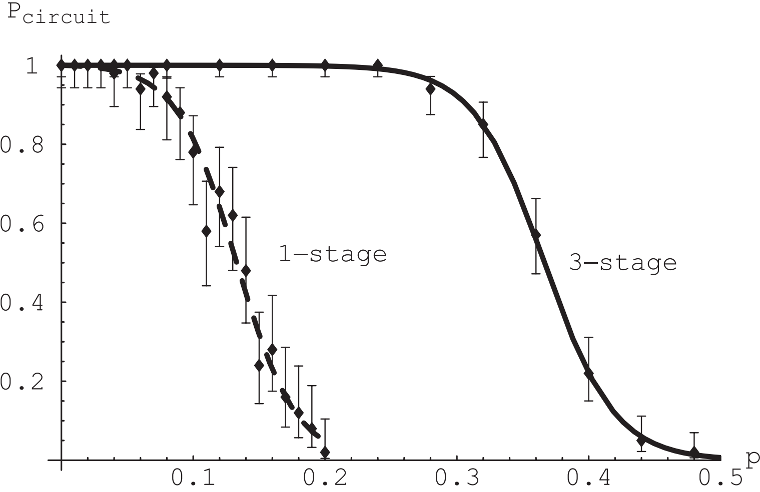


Fig. 15. Probability P_{circuit} to be able to find a correct circuit for a 3-bit adder in a crossbar as a function of defect probability p. The points show the estimates from the simulation runs, with error bars indicating the 95% confidence intervals in the estimates of P_{circuit}. The *curves* show the maximum-likelihood fits of a sigmoid function, for the single and three-stage adders (*dashed and solid*, respectively). The crossbar sizes for the two circuits are 768 and 896 for the single and three-stages adders, respectively

With multiple sets of trials, we select a and b to maximize the likelihood of obtaining the observed results from the simulation. Since each trial is independent of the others, this amounts to maximizing the product of the individual likelihoods, described above, for each p value. Figure 15 shows examples of the resulting fits.

To understand the existence of this threshold behavior, and how it depends on the circuit and crossbar area, consider a simplified version of the allocation in which we ignore all constraints on the locations of the functioning diodes. By ignoring these constraints, we obtain an upper bound on P_{circuit} and rough guides to the location and steepness of the threshold. Specifically, a crossbar with area A has k defective junctions with probability $\text{Bi}(A, k; p)$ given by (4). The expected number of defects is $\mu = Ap$ with standard deviation $\sigma = \sqrt{Ap(1-p)}$. A circuit requiring d diode junctions is very likely to exist when the number of defects is likely to be less than $A - d$. Conversely, when the number of defects is usually larger than this value, the crossbar is unlikely to be able to implement the circuit. More precisely, when $A - d$ is several standard deviations above or below μ, $P_{\text{circuit}} \approx 1$ or 0, respectively. This discussion predicts the threshold near the value of p for which $\mu = A - d$, i.e.,

$$p \approx 1 - d/A. \tag{7}$$

The change between these extremes takes place mainly over a range of p values corresponding to about a standard deviation around the mean, i.e., from the value where $\mu + \sigma = A - d$ to that where $\mu - \sigma = A - d$. The corresponding range in p values is $\frac{1}{A+1}\sqrt{1 + 4d - 4d^2/A}$ or $\sim \sqrt{1 + 4d}/A$ for large areas.

These specific values, derived by ignoring all constraints on the locations of the functioning devices, differ from the location and width of the threshold seen in the simulations. Nevertheless, they give some qualitative insight into the behaviors we observe. For instance, as the crossbar area increases, the threshold moves to larger values of p: as one would expect, larger crossbars provide more chances to find the required number of functioning junctions. The threshold width is small when the area is close to the minimum possible (i.e., just enough to hold the required number of functioning diodes) or when the area is very large. For a given area, comparing two implementations with different numbers of diodes, we see the implementation with more diodes, i.e., larger d, has a lower threshold: it is less tolerant of defects. As a final observation, consider the scaling to larger circuits (e.g., k-bit adders for $k > 3$). Taking the crossbar area to be a fixed factor larger than the required number of diodes, i.e., $A = rd$ gives a fixed threshold location of $1 - r$ while threshold width decreases as $O(1/\sqrt{d})$. That is, for larger circuits the threshold behavior becomes sharper.

Crossbar Area

Figure 15 shows the behavior for a fixed size crossbar. Since the single and 3-stage adder circuits require different areas, it is also useful to compare the area required to obtain a fixed value of P_{circuit}. To estimate the behavior of the adder circuit for different areas, we used the sigmoidal fit of (6). This fit has two parameters, a and b, characterizing the width and location of the transition from $P_{\text{circuit}} \approx 1$ to $P_{\text{circuit}} \approx 0$. For definiteness, we consider arrays of fixed shapes (i.e., ratio of number of columns to numbers of rows allocated for the AND and OR operations), and show the resulting behavior in Fig. 16. Specifically, for the single-stage adder, we examined arrays whose numbers of columns, input rows and output rows had the ratios $4:2:1$. By comparison, the minimum size array that can implement the circuit, with 31 columns, 12 inputs and 4 outputs, has ratios $31 : 12 : 4 = 7.75 : 3 : 1$. Thus the shape we use allocates relatively more of the array area to the rows, especially those for the outputs, than would be the case from uniformly scaling up the minimum array. This allocation is beneficial because the single-stage circuit is particularly sensitive to defects in the rows due to the need for large numbers of functioning diodes, especially for the outputs, as seen in Fig. 8. For the three-stage adder, various shapes close to scaled-up versions of the minimum size array (25 columns, 19 inputs, 11 outputs) had similar behaviors. As a convenient ratio to simulate with different sizes, we used $28 : 20 : 12$.

Over the range of crossbar areas we examined via simulation, the transition width had only small variation with area. On the other hand, the location of

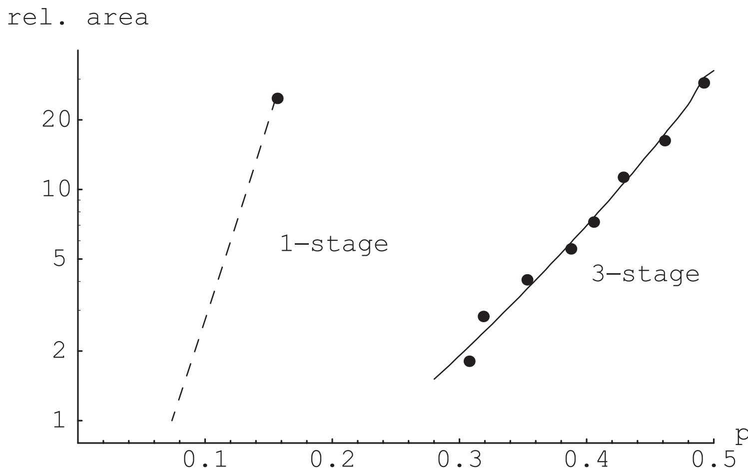


Fig. 16. Relative area required to have at least 90% probability to be able to find a correct 3-bit adder circuit as a function of defect probability p, based on interpolating the fit to the simulation results. The *dashed* and *solid curves* correspond to single and three-stage adders, respectively. The points correspond to values estimated from individual sigmoid fits to results from each crossbar area, for a fixed choice of array shapes. With the shapes used here, the lowest simulation point on each curve is the smallest possible functional array with that shape, and have somewhat larger area than the minimum area for each case (indicated by the bottoms of the two curves). Areas are relative to that required for a single-stage adder on a defect-free crossbar, i.e., 496 junctions

the threshold increased with area. Motivated by the discussion leading to (7), we suppose the sigmoid parameters a, b vary with area A according to

$$a = \alpha A^{\delta}$$
$$b = 1 - \beta A^{-\eta}$$

where α, δ, β and η are parameters with nonnegative values. We fit these four parameters to the simulation results for with different areas to produce a single functional form relating P_{circuit} to A and p for a given circuit and array shape. The resulting functional form fits the results for single areas about as well as sigmoid functions optimized individually for each area. This fit allows interpolating the behavior for other areas and defect rates than those evaluated via the simulation. In particular, it allows estimating the crossbar area required to have at least a given desired yield, i.e., value of P_{circuit}. For instance, Fig. 16 shows the resulting estimates for the area required to achieve $P_{\text{circuit}} = 90\%$. Thus, we have about 90% yield with the 3-stage adder at 30% defect rate and area 896, which is 1.8 times the minimum area of 496 junctions. We see the single-stage adder is much more sensitive to defects, so requires larger areas to compensate.

The lowest points on the curves in Fig. 16 correspond to the minimum crossbar size that can implement the adder: 496 and 750 for the one and three-stage circuits, respectively. These values do not occur at $p = 0$ because the minimum areas are determined by the required numbers of logic operations, inputs and outputs rather than the number of diode junctions. Hence even the minimum area crossbars can tolerate some defective junctions. Comparing the two adder implementations, we see that for $p < 0.085$, the single-stage adder gives 90% success with crossbars whose size is too small to also implement the three-stage adder. For $0.085 < p < 0.3$, the three-stage adder on the smallest possible crossbar that can implement it gives over 90% success, while the area required for the single-stage adder increases significantly. With $p > 0.3$, required areas of both circuits increase, though the three-stage implementation requires much less area. This discussion illustrates how achievable defect rate, choice of circuit implementation and available crossbar area interact to determine the circuit yield.

Array Shape

In addition to the area of a crossbar array considered above, its shape also influences the likelihood of being able to implement a circuit in spite of defects. A successful circuit requires not only enough functioning junctions, but also the ability to properly connect them to each other and the inputs and outputs. Thus P_{circuit} is, in general, a function separately of the three numbers characterizing the crossbar: the number of columns, the number of rows allocated for AND operations on inputs, and the number of rows for OR's on the results of the ANDs to produce the circuit outputs. For example, the one-stage 3-bit adder uses 12 input wires (two for each of the 3 input bits for each of the two numbers to be added) and 4 outputs, for a total of 16 rows. It uses 31 columns to form the required logic operations. Thus a crossbar with fewer than 16 rows could never implement this circuit, no matter how many columns or how low a defect rate it had. More generally, scaling up the number of rows and columns by the same factor may not be the best way to improve performance from a given crossbar area. For example, it may be better to increase number of rows more than number of columns for an implementation requiring many diodes on the same row, vs. a different implementation using many diodes on a single column.

Figure 17 shows an example of how array shape affects implementing the single-stage 3-bit adder, shown in Fig. 8. In this example, we examined various shaped arrays, all with the same area, 1,728, which is about 3.5 times the minimum array size for this circuit. For each shape, we fit the sigmoid of (6) to the simulation results and used this fit to estimate the defect rate with 90% probability to implement the circuit. The shapes cover a range from a very wide array (108 columns, 12 inputs, 4 outputs) to the narrowest possible arrays with area 1,728 still capable of implementing the circuit even when there are no defects, which have 32 columns and 54 rows, with allocations of

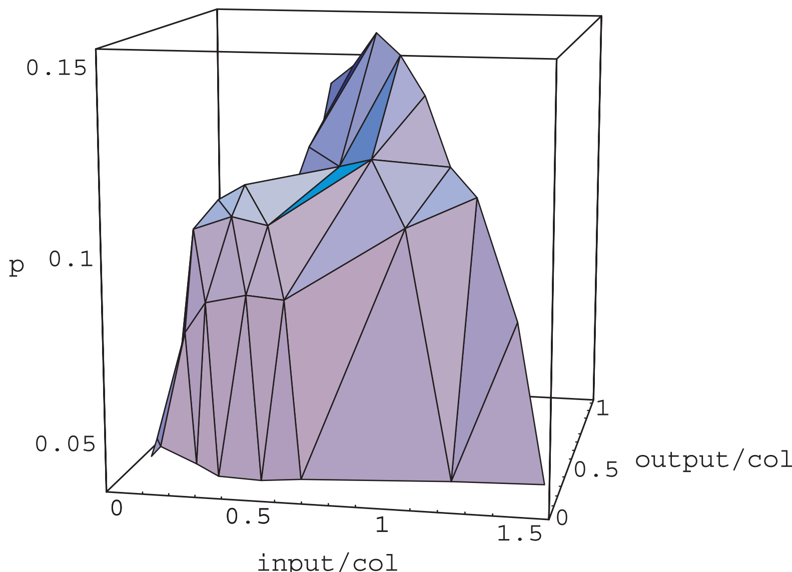


Fig. 17. Defect rate, p, at which a single-stage 3-bit adder circuit can be found with 90% probability as a function of array shape for area equal to 1728. The shape is specified by the ratios of numbers of input and output rows to the number of columns in the crossbar

those rows ranging from having 12 inputs and 42 outputs to 50 inputs and 4 outputs.

The best performance in Fig. 17 is for the array with 32 columns, 24 inputs and 30 outputs, giving at least 90% probability to produce a circuit with $p \leqslant 0.15$. This shape tolerates about three times as many defects as the worst shapes shown in the figure (which allocate only four wires to the outputs). Because the one-stage adder requires many diode connections on the outputs, this best performing array shape devotes a relatively large portion of the area to redundant output wires. By contrast, the shape used in Fig. 16 to illustrate behavior as a function of area arrays whose numbers of columns, input rows and output rows had the ratios 4 : 2 : 1, a shape with intermediate performance among those shown in Fig. 17, i.e., allowing $p \leqslant 0.11$.

We also found variation due to shape in the three-stage adder, and that the best choice of shape varies somewhat with array size.

Scaling and Allocation Run Time

For comparison, we also examined 1, 2 and 4-bit adder circuits. They gave the same qualitative behaviors, and show the threshold behavior becomes more abrupt as circuit size increases. Thus the threshold becomes more significant as circuit size increases, thereby giving a useful design criterion for the maximum allowable defect rate for a given desired circuit and crossbar area.

For a given circuit, we found the typical run time of the allocation algorithm increases with p up to the threshold region. For even higher p values, most crossbars cannot implement the circuit and the algorithm terminates by reaching the bound on its run time. Nevertheless, to examine the allocation cost behavior for larger p, we also ran the algorithm to completion (i.e., with no bound on the run time) for a smaller circuit, namely a 1-bit adder. We found that the median allocation cost peaked near the threshold at which $P_{\text{circuit}} \approx 0.5$ and then decreased for larger p values, as the additional constraints introduced with additional defects allows the allocation algorithm to prune large sets of possibilities and more rapidly conclude no implementation is possible. This behavior is consistent with that seen in many other studies of combinatorial search problems [17]. That is, as problems become more constrained, there is an abrupt transition from almost always to almost never having a solution, and the typical search cost for a variety of search methods peaks near this transition.

6 Discussion

We examined the ability to map adder circuits onto a crossbar architecture in spite of numerous independent defects. We also showed how logically equivalent choices for the mapping differ in their tolerance for defects and use of circuit area. We thus illustrate some design trade-offs for molecular electronics systems. In general, higher defect rates require sparser circuit mappings. Sparser circuits require fewer functioning junctions on a single row or column of the crossbar and hence are more likely to have successful maps to a defective crossbar. On the other hand, such circuits use more stages leading to larger, slower implementations.

The computation required for the monomorphism algorithm we used to map the circuits to the crossbar cannot easily be categorized with simple parameters of the graphs, e.g., their sizes. Instead, the computation time depends on the detailed topological properties of the two graphs to be matched. In particular, the typical computational costs depends significantly on the number of matchings. Thus, we used empirical testing to evaluate the performance. For any given circuit architecture and defect rate, one will need to experimentally trade-off logic density and mapping execution time to arrive at an economically viable mapping strategy. In practice, such a trade-off could be implemented by selecting an economically reasonable computational cost and simply considering as failures any crossbars that do not produce a mapping within that limit.

Quantitatively, the likelihood of tolerating defects shows a threshold behavior, i.e., changing from near one to near zero over a relatively small range of defect rates (for a given crossbar size) or over a small range of crossbar area (for a given defect rate). These thresholds become sharper as larger circuits are considered. Thus identifying the threshold locations gives useful guidelines for circuits feasible to implement with given defect rates and crossbar sizes.

More generally, thresholds are seen in many properties of numerous combinatorial structures, such as random graphs [11], and related search algorithms [17]. Of particular interest in our context are the thresholds in existence of various subgraphs [26] and extensions to any monotone property of graphs with various geometries [12,13]. Monotone properties of a graph are those that continue to hold when edges are added to that graph, and include the existence of subgraphs with specified constraints as required for implementing circuits on the crossbar with defects. These examples illustrate how definite behaviors can arise from unreliable, statistical systems. From a practical perspective, these thresholds are associated with large computational costs to determine whether a consistent structure exists. Our simulations show this behavior as well, with particularly high computational costs near the thresholds. In the context of molecular circuits, identifying defect rates corresponding to the thresholds provides a practical limit above which not only is yield low but also attempting to allocate a circuit would likely be computationally expensive.

Several extensions to this work would be interesting. First, we examined behavior with respect to independent point defects in the crossbar. It would be useful to examine how defects actually arise in manufacturing and whether they differ significantly from the independence we assumed. For example, defects might short out an entire row or column of the crossbar, or appear with strong spatial correlations. As with studies of other stochastic systems, these different error models will likely give rise to similar qualitative behaviors, including the existence of thresholds in implementation feasibility, but with different quantitative values.

A second extension to this work involves the rewrites for the circuit. We showed these rewrites have different resource requirements (i.e., crossbar area) and tolerance for defects. Another application for rewrites would be to make only minor modifications in a given circuit with the goal of removing just those parts of the original circuit that are most difficult to map to a defective crossbar. For instance, the fourth from last row of the adder in Fig. 8, computing S_2, requires 16 functioning diode junctions on the row. This is more than is required on any other row or column of the circuit. In a defective crossbar, the need to find one row with at least 16 functioning devices is a major limitation that could be removed by simply rewriting just that part of the circuit. For instance, instead of using one row to implement the logical-OR of 16 inputs, we could split the function into two rows, each performing a logical-OR of 8 inputs, followed by a logical-OR of those results to form the final value for S_2. This would improve the chances of mapping the circuit to a defective crossbar. This rewrite also increases the size and delay of the circuit, though not as much as the entirely different rewrite of Fig. 7. This example illustrates how rewrites could be targeted to remove just the parts of the circuit least tolerant of crossbar defects, thereby giving a range of options suitable for crossbars with different defect rates.

We only considered logically equivalent rewrites: e.g., the different implementations of the adder circuit compute exactly the same logical function of

their possible inputs. An additional possibility is allowing some non-equivalent rewrites, particularly in conjunction with fault tolerance in a higher-level system architecture of which the molecular device is only one part [9]. In this case, an occasional error in the circuit (e.g., due to undetected or new defects, or noisy operation) may be tolerable. As another application, logic circuits used for pattern recognition based on combinations of different sensors (e.g., each detecting a different chemical or concentration in the environment) could perform quite well even with a non-equivalent rewrite that gives different outputs from the ideal circuit on a small subset of inputs. For instance, suppose a device consists of three sensors for different concentrations of a chemical ("low", "medium" and "high") based on receptors with differing affinities and saturation levels. Here the low concentration sensor will be active whenever the medium or high are, and the medium one will be active whenever the high one is. Thus a rewrite of a pattern-recognition circuit that changes the output for the input corresponding to the low and high sensors on but the medium sensor off would have no practical significance since that combination of inputs would not arise with this particular set of sensors. More generally, extending consideration to a limited set on non-equivalent rewrites could provide substantial improvements in ability to implement the circuit (by being below a higher threshold) than would be possible among logically equivalent rewrites.

We focus on circuit area as the main cost criterion. In practice, other properties may also be important in developing practical molecular circuit designs. For instance, rewrites not only differ in the circuit area they require but also in propagation delay and their need for additional components such as restoring latches. Moreover, we have not included the cost to test the crossbar for defects and the possibility this cost could vary with accuracy of this test. In particular, false positives unnecessarily reduce the available circuit area and false negatives could result in erroneous outputs for some inputs. Including these criteria as part of the overall system manufacturing cost could alter the choice of the best design.

Acknowledgements

We thank Phil Kuekes and Li Zhang for helpful discussions.

References

1. J. G. Brown and R. D. Blanton, "CAEN-BIST: testing the nanofabric," In *Proc. of the Intl. Test Conf. (ITC2004)*, pages 462–471, 2004
2. Y. Chen et al., "Nanoscale molecular-switch crossbar circuits," *Nanotechnology*, 14:462–468, 2003
3. Y. Chen and R. S. Williams, Nanoscale patterning for the formation of extensive wires, US Patent 6,294,450, September 25 2001

4. C. P. Collier et al., "Electronically configurable molecular-based logic gates," *Science*, 285:391–394, 1999
5. L. P. Cordella et al., "An improved algorithm for matching large graphs," In *Proc. of the 3rd IAPR-TC-15 Intl. Workshop on Graph-Based Representations*, pages 149–159, 2001
6. A. DeHon, "Array-based architecture for FET-based nanoscale electronics," *IEEE Trans. on Nanotechnology*, 2:23–32, 2003
7. A. DeHon, S. C. Goldstein, P. J. Kuekes, and P. Lincoln, "Nonphotolithographic nanoscale memory density prospects," *IEEE Trans. on Nanotechnology*, 4:215–228, 2005
8. A. DeHon, P. Lincoln, and J. E. Savage, "Stochastic assembly of sublithographic nanoscale interfaces," *IEEE Trans. on Nanotechnology*, 2:165–174, 2003
9. A. DeHon and H. Naeimi, "Seven strategies for tolerating highly defective fabrication," *IEEE Design and Test of Computers*, pages 306–315, July–August 2005
10. Y. El-Sonbaty and M. A. Ismail, "A graph-decomposition algorithm for graph optimal monomorphism," In A. F. Clark, editor, *Proc. of the 8th British Machine Vision Conference (BMVC97)*, 1997
11. P. Erdos and A. Renyi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–61, 1960
12. E. Friedgut and G. Kalai, "Every monotone graph property has a sharp threshold," *Proc. of the American Mathematical Society*, 124(10):2993–3002, 1996
13. A. Goel, S. Rai, and B. Krishnamachari, "Monotone properties of random geometric graphs have sharp thresholds," *Annals of Applied Probability*, 15:2535–2552, 2005
14. S. Goldstein and M. Budiu, "Nanofabrics: Spatial computing using molecular electronics," In *Proc. of the 28th Intl. Symposium on Computer Architecture (ISCA)*, pages 178–191, 2001
15. J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams, "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, 280:1716–1721, 1998
16. T. Hogg, Y. Chen, and P. Kuekes, "Assembling nanoscale circuits with randomized connections," *IEEE Transactions on Nanotechnology*, 5:110–122, 2006
17. T. Hogg, B. A. Huberman, and C. P. Williams, editors, *Frontiers in Problem Solving: Phase Transitions and Complexity*, volume 81, Amsterdam, 1996. Elsevier. Special issue of *Artificial Intelligence*
18. T. Hogg and G. Snider, "Defect-tolerant adder circuits with nanoscale crossbars," *IEEE Transactions on Nanotechnology*, 5:97–100, 2006
19. Yu Huang et al, "Logic gates and computation from assembled nanowire building blocks," *Science*, 294:1313–1317, 2001
20. P. J. Kuekes and R. S. Williams, Demultiplexer for a molecular wire crossbar network, US Patent 6,256,767, 3 July 2001
21. P. J. Kuekes, R. S. Williams, and J. R. Heath, Molecular wire crossbar memory, US Patent 6,128,214, 3 Oct. 2000
22. P. J. Kuekes, R. S. Williams, and J. R. Heath, Molecular-wire crossbar interconnect (MWCI) for signal routing and communications, US Patent 6,314,019, 6 Nov. 2001
23. P. J. Kuekes, D. R. Stewart, and R. S. Williams, "The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits," *J. Appl. Phys.*, 97:034301, 2005

24. P. Kuekes, Molecular crossbar latch, US Patent 6,586,965, 1 July 2003
25. N. A. Melosh et al, "Ultrahigh-density nanowire lattices and circuits," *Science*, 300:112–115, 2003
26. E. M. Palmer, *Graphical Evolution: An Introduction to the Theory of Random Graphs*, Wiley Interscience, NY, 1985
27. R. Rad and M. Tehranipoor, "SCT: An approach for testing and configuring nanoscale devices," In *Proc. of the 24th IEEE VLSI Test Symposium (VTS06)*, pages 370–377, 2006
28. S. K. Shukla and R. I. Bahar, editors, "*Nano, Quantum and Molecular Computing: Implications to High Level Design and Validation*," Kluwer, Norwell, MA, 2004
29. SIVALab, VF graph matching library, University of Naples "Federico II", 2001
30. M. Stan, P. Franzon, S. Goldstein, J. Lach, and M. Ziegler, "Molecular electronics: From devices and interconnect to circuits and architecture," *Proc. of the IEEE*, 91:1940–1957, 2003
31. D. B. Strukov and K. K. Likharev, "CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, 16:888–900, 2005
32. J. V. Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, volume 34 of *Ann. Math. Stud.*, pages 43–98. Princeton University Press, 1956
33. Z. Wang and K. Chakrabarty, "Using built-in self-test and adaptive recovery for defect tolerance in molecular electronics-based nanofabrics," In *Proc. of International Test Conf. (ITC2005)*, pages 477–486, 2005
34. M. M. Ziegler and M. R. Stan, "Design and analysis of crossbar circuits for molecular nanoelectronics," In *Proc. 2nd IEEE Conf. on Nanotechnology (NANO-2002)*, pages 323–327, 2002

Chapter 2: Built-in Self-Test and Defect Tolerance in Molecular Electronics-Based Nanofabrics

Z. Wang and K. Chakrabarty

1 Introduction

Although complementary metal-oxide semiconductor (CMOS) chips are projected to continue their dominance for another 10–15 years [1], CMOS technology today faces a number of challenges. Quantum effects will soon make it nearly impossible to further scale devices. Deep sub-micron (DSM) technologies suffer from high leakage, and it is projected that stand-by power and active power for CMOS chips will soon become comparable [2]. Moreover, the high cost associated with chip masks and next-generation fabrication plants poses a formidable economic barrier to commercial nanometer-scale lithography.

Chemically-assembled electronic nanotechnology (CAEN) is a nanoelectronic technology that is under intense investigation as a possible alternative to CMOS integrated circuits [1, 3–6]. It has the potential to achieve high density, and it can be fabricated using low-cost chemical synthesis processes. CAEN uses self-assembly and self-alignment to construct electronic circuits out of nanometer-scale devices. CAEN-based systems, referred to as the nanofabric, can achieve a density of more than 10^8 gate-equivalents per cm^2 by using interconnected 2D-arrays of nano-scale wires that can be electronically configured as logic networks, memory units, and signal-routing cells [6]. The 2D arrays, referred to as nanoblocks, are the fundamental units of the nanofabric. A prototype nanowire-based crossbar design with 6.8 Gbits cm^2 density is reported in [7], and a manufacturer is promising memories using carbon nanotubes [8].

While CAEN-based systems offer the advantage of low manufacturing cost and high density, they are inherently unreliable. The low reliability is a direct consequence of the stochastic nature of self-assembly. It has been predicted that the defect density of CAEN-based systems can easily exceed 10% [6]; therefore it is not economically feasible to discard a nanofabric once a fault is detected. Defect tolerance is needed to make such nanofabrics commercially viable.

Defect tolerance refers to the ability to detect and locate fault sites on a chip, and then avoid the faults through reconfiguration methods. The first step in defect tolerance is to map designs to usable sets of resources; this step leads to increased yield and reduced manufacturing cost. New methods must therefore be devised to diagnose defective sections of the nanofabric. Unlike many testing and diagnosis techniques intended for CMOS chips, testing methods for nanofabrics cannot simply assume a small number of defects or use conventional fault models that only target a few fault sites.

Modern memory chips use built-in redundancy, in which spare rows and columns are used to replace defective rows and columns, respectively, to achieve defect tolerance [9]. However, it is unlikely that any row or column of nanoblocks will be defect-free. Moreover, the nanofabric is intended to implement complex logic functions, hence simple row/column replacement algorithms will not be efficient. Majority voting is another fault tolerance technique using redundant hardware resources to perform correct computation in the presence of defects. However, due to the high anticipated defect density of CAEN-based systems, the amount of additional physical resources required will not be affordable.

A nanofabric system is similar to a field-programmable gate-array (FPGA) because of its regular 2D-array architecture and reconfigurability. A number of testing methods have been proposed for various FPGA architectures, e.g., [10–15]. In FPGA-BIST presented in [10, 11], programmable logic blocks (PLBs) are configured as test pattern generators (TPGs), blocks under test (BUTs) and output response analyzers (ORAs) for built-in self-test (BIST). A TPG applies exhaustive test patterns to a BUT and output responses are fed to an ORA. Multiple configurations are needed to ensure that every PLB is tested as a BUT. Teramac [16, 17] is an FPGA-based custom computer system that is capable of running user designs even if up to 75% of its FPGAs contain defects. A testing phase is used after fabrication to identify defects in FPGAs. In the Teramac system, circuit components are configured as LFSRs that generate long psuedo-random bit streams and communicate them to primary outputs. If the output bit stream is correct, all components are assumed to be defect-free; if incorrect, these components are configured to create new LFSR signature generators. The resources at the intersection of defective LFSR configurations are marked as defective. However, the problem of nanofabric testing is different from FPGA testing due to two main reasons: (1) nanofabric systems are expected to have much higher defect densities and a larger number of resources, and (2) as detailed in Sect. 2, the fundamental units (nanoblocks) in the nanofabric are very simple compared to PLBs in FPGA. It is difficult to create complex comparators or LFSR signature generators using primitive units that are likely to be defective. Therefore, new methods must be devised to address these problems.

In this work, we propose a BIST and recovery procedure for the nanofabric that uses simple single-nanoblock TPGs, BUTs and ORAs. This fine-grained

test method allows us to handle high defect densities. We exploit the fact that even for high defect densities, a small number of neighboring simple nanoblocks can be expected to be defect-free. Instead of specifying a set of complex test patterns, our procedures rely on a set of configurations to test the nanofabric. Complex test patterns cannot be generated by simple TPGs in the nanofabric, and due to limited routing resources, it is difficult to feed test patterns using external testers. Since nanoblocks are tested in parallel, the testing time using the proposed procedure is independent of the size of the nanofabric. However, the method used to read out test results also affects the overall testing time. In the absence of manufactured nanofabric chips, it is not clear how much time will be needed for ORA access; the access time may depend on the size of the nanofabric. We consider the detectability of multiple faults in blocks within the nanofabric. We also present simple bounds on the recovery that can be achieved for a given defect density.

The configurations for fault detection are presented in detail in Sects. 5 and 6. Section 7 discusses the detection of multiple faults. Section 8 presents an adaptive recovery procedure to further improve the performance and Sect. 9 investigates the effectiveness of the whole procedure. We present our simulation results in Sect. 10 and conclude the chapter in Sect. 11.

2 Nanofabric

A feasible fabrication process for CAEN systems is bottom-up manufacturing, where basic components such as wires and switches are first obtained through chemical self-assembly, and then aligned and grouped into regular structured arrays through self-assembly to form complete systems [3]. Two planes of aligned wires are combined to form a two-dimensional grid with configurable molecular switches at the cross-points. The resulting grid is of the order of a few microns. A post-fabrication configuration step is used to create useful circuits out of these grids [3].

The nanofabric architecture has been proposed for a CAEN-based system in [1,3,6]. The self-assembly process does not allow precise end-to-end connections between nanoscale wires. The nanofabric architecture requires that all connections be made only at the cross-points between two orthogonal wires. Molecular latches based on resonant tunneling diodes, referred to as RTDs, are also incorporated in this architecture for saving states and for signal restoration [18].

Similar to FPGAs, the nanofabric is a regular 2D-mesh of interconnected fundamental units called nanoblocks, as shown in Fig. 1. A nanoblock can be programmed after fabrication to implement logic functions. The switchblock is the area where the input and output wires of nanoblocks overlap. It can be configured to route signals between nanoblocks [3].

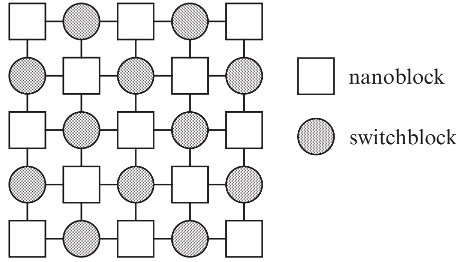


Fig. 1. The nanofabric architecture

2.1 Nanoblock

As shown in Fig. 2, a nanoblock consists of three parts: (1) the molecular logic array (MLA), which implements the functionality of the block, (2) the molecular latches, used for signal restoration and signal latching, and (3) the I/O area, used to connect the nanoblock to its neighbors [3].

The MLA is composed of two orthogonal sets of wires. At each intersection of two wires lies a configurable molecular switch. The switches, when configured to be "on", act as diodes [3]. The direction of the current flowing through a "on" molecular switch is determined during fabrication and is nonreconfigurable. Figure 3 shows the implementation of an AND gate. If either A or B is at logic "0", the corresponding diode is forward-biased and turned on. The resistors are manufactured appropriately, i.e., resistors attached to Gnd have smaller impedances than those attached to V_{DD}, such that the output vertical wire is pulled down to logic "0" [3]. Note that the resistance of nanowires and molecular switches are very low. Figure 3 also shows how an OR gate can be implemented. This diode-resistor logic is unable to perform the inversion operation, therefore complemented inputs are required and the complement of each logic function also needs to be implemented.

If the MLA portion of a nanoblock has k horizontal wires and k vertical wires, then the size of the nanoblock is referred to as $k \times k$. We only consider nanoblocks that have equal numbers of horizontal wires and vertical wires.

The above nanoblock design is dictated by fabrication constraints. Each side of the block can have either inputs or outputs, but not both. All nanoscale wire-to-wire connections are made between two orthogonal wires; precise end-to-end alignment is not possible. The outputs of the blocks are either facing south and east (SE) or north and west (NW), as shown in Fig. 2 [3]. Without loss of generality, we assume that a nanofabric consists of only SE nanoblocks whose outputs are facing south and east.

The MLA implements Boolean functions using diode-resistor logic. The drawback of this logic style is that a signal is degraded whenever it passes a molecular switch. The molecular latch, constructed entirely from molecular-scale devices, is used to perform signal restoration using power from the clock to provide gain. The molecular latch also provides the properties of I/O isolation and noise immunity [18].

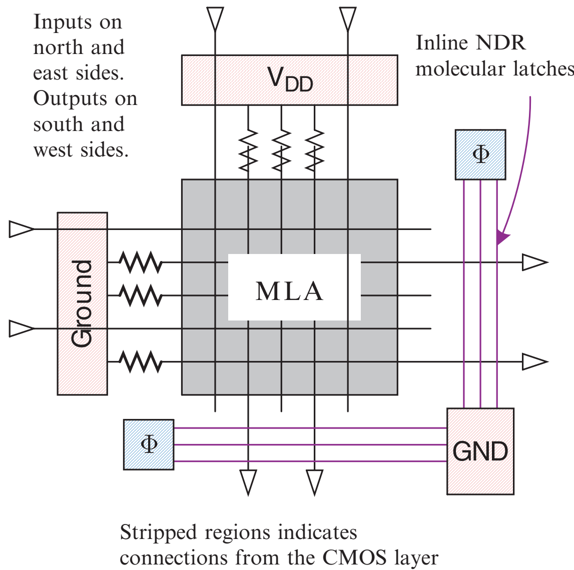


Fig. 2. Schematic of a nanoblock [3]

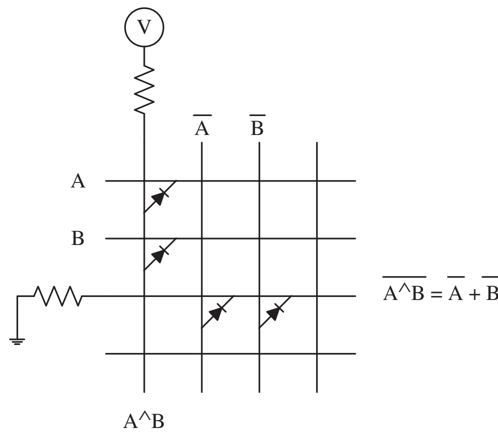
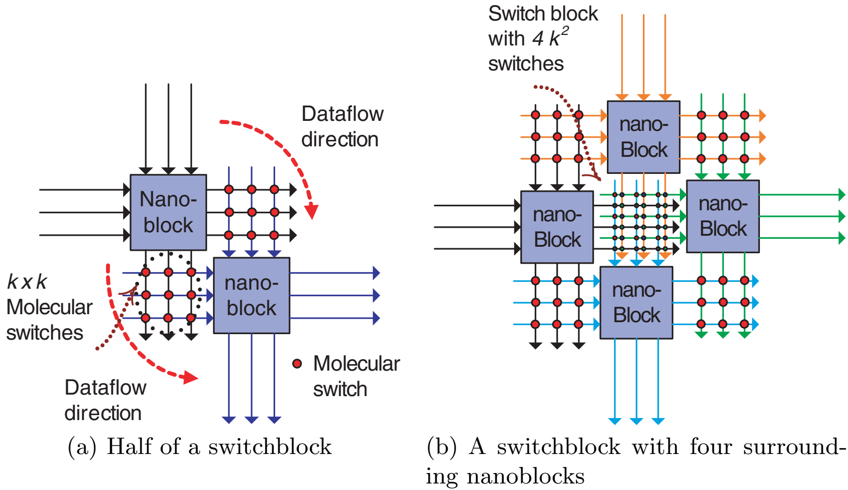


Fig. 3. An AND gate

2.2 Switchblock

A switchblock is similar to the MLA portion of a nanoblock, with the difference that it does not have inline NDR latches, I/O ports and connections to V_{DD} and Gnd. As shown in Fig. 4, a switchblock is formed by four nanoblocks; crossing horizontal wires and vertical wires from the surrounding nanoblocks are connected by configurable molecular switches. If the size of the nanoblocks is $k \times k$, then there are $2k$ vertical wires and $2k$ horizontal wires



(a) Half of a switchblock

(b) A switchblock with four surrounding nanoblocks

Fig. 4. Nanoblock connectivity [3]

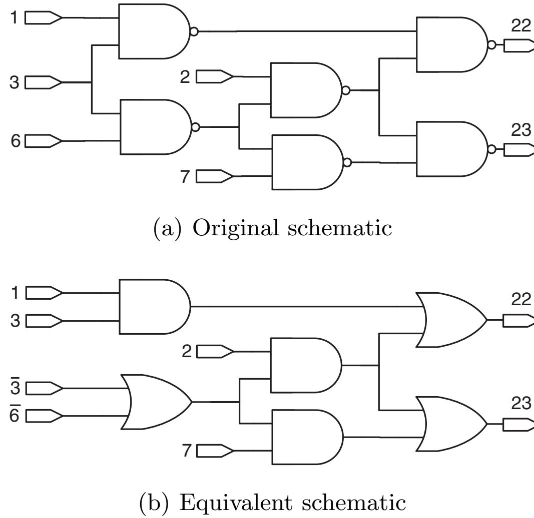
inside a switchblock, and $4k^2$ cross-points can be formed. For SE nanoblocks, a switchblock is capable of providing four directions of data flow: west to south (WS), west to east (WE), north to east (NE), and north to south (NS). WS and NE data flows can co-exist with each other in a same switchblock. On the other hand, WE and NS data flows cannot co-exist with any other data flows because vertical (horizontal) wires cannot be directly connected to vertical (horizontal) wires. Therefore if two vertical (horizontal) wires are to be connected, a horizontal (vertical) wire in the same switchblock must be used, which means that this horizontal (vertical) wire cannot be used by its own nanoblock in order to avoid conflicts.

2.3 Defect Tolerance

The nanofabric has a much higher defect density than standard CMOS chips due to the imprecise and nondeterministic manufacturing process. Wires will rarely all be equidistant from each other. Wires that should be parallel may be askew or they may intersect. The connections between wires may be open or wires may be shorted [6].

The nanofabric has a built-in capability for defect tolerance due to its reconfigurability. An effective testing procedure should lead to a *defect map*, which provides the locations of the defective nanoblocks and switchblocks. The defect map can then be used by software tools to avoid faulty resources during system reconfiguration.

One metric of defect map quality is *recovery*, defined as the percentage of defect-free nanoblocks and switchblocks that are correctly diagnosed [1]. This



(a) Original schematic

(b) Equivalent schematic

Fig. 5. Schematics of c17

metric indicates the diagnostic accuracy of a testing procedure. It should also be ensured that no faulty blocks are diagnosed as defect-free. An ideal recovery of 100% implies that every defect-free block in the nanofabric is correctly diagnosed. However, this metric is useful only for simulation; in practice, it is difficult to ascertain the actual number of defect-free blocks after fabrication. An effective testing procedure should correctly identify a large fraction of these defect-free blocks, thereby minimizing wastage.

To illustrate how a nanofabric (with defects) can be used to implement a real circuit, we take c17, an ISCAS'85 benchmark circuit, as an example. Figure 5a shows the schematic of c17. To map it to a nanofabric, since nanofabrics cannot implement inversion logic, we first transform it into an equivalent circuit as shown in Fig. 5b. We assume that complemented input signals (e.g., $\bar{3}$ and $\bar{6}$) are available. Figure 6 shows how this equivalent circuit can be mapped to a nanofabric with defects. We assume that defective blocks (marked by crosses) are identified by a testing process. Note that even if a nanoblock/switchblock is defective, it may still be used to route signals if the exact nanowire in use is defect-free (e.g., the switchblock that routes output 22).

3 Related Prior Work

The testing of nanofabrics was first addressed in [1, 19]. The none-some-many algorithm presented in [1] creates LFSR-based signature generators from a random selection of nanoblocks. This approach however makes the unrealistic assumption that unlimited interconnect resources are available to create

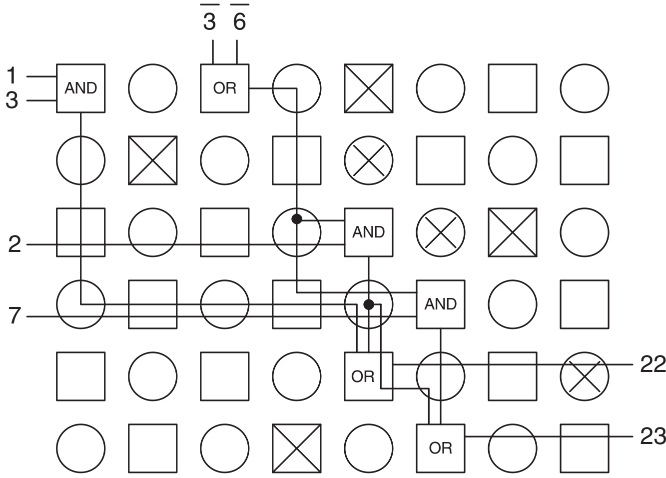


Fig. 6. The mapping of c17 to a defective nanofabric

signature generators from randomly-selected nanoblocks. Moreover, since this approach uses a large number of nanoblocks to implement LFSR-based signature generators, it is coarse-grained and it can only provide limited recovery.

The CAEN-BIST approach presented in [19] is a fine-grained test method. It configures a nanoblock as a tester to test its neighboring nanoblocks. Test patterns are fed to both the tester and the nanoblock under test (BUT) from an external source. A defect-free BUT generates output patterns that are identical to the input patterns. The tester compares the input test patterns and the output patterns from the BUT to see if the BUT is defective. The average recovery is reported to be almost 100% for defect densities up to 20%. However, this approach makes two strong assumptions: (1) a k-bit comparator can be implemented using a nanoblock, and (2) defect-free data paths from external test circuits to testers and BUTs can be dynamically identified during the test procedure. Since the nanoblocks can only implement simple logic functions, it is not clear how they can be configured to implement k-bit comparators. Moveover, [19] does not consider the limitations in dataflow imposed by the biasing of the molecular switches. In addition, because the input patterns are provided by external circuits instead of internal nanoblocks, CAEN-BIST can only be performed in a wave-like manner in which a set of nanoblocks in the same diagonal tests another set of nanoblocks until the entire nanofabric has been tested. Therefore, the complexity of CAEN-BIST depends on the size of the nanofabric under test.

Recently, a BIST method to test the nanoblocks was presented in [20]. However, this work does not address switchblocks; thus it can lead to defect-free switchblocks being deemed useless, and defective switchblocks being marked defect-free. Moreover, the recovery procedure in [20] is coarse-grained; it is limited by the fact that recovered blocks cannot be used to further increase the nanofabric recovery.

Another BIST method for nanofabrics was proposed recently in [21]; this method uses only TPGs and ORAs. The TPGs and ORAs are tested in parallel to decrease testing time and to improve recovery. However, this work does not address defects in switchblocks. In [22], an on-chip, application-specific BIST approach is described for nanoscale devices. This BIST method is performed whenever the chip is configured for a given application. It performs functional test for the blocks in the nanoarray and it eliminates the use of defect maps. However, because the BIST circuitry is implemented on-chip in the CMOS domain, the area overhead tends to be excessive compared with the size of the nano-domain circuitry. Moreover, during the BIST procedure, a large amount of memory is needed to save intermediate test results.

4 Nanofabric BIST Approach

We now present a BIST approach for nanofabric testing that exploits the reconfigurability of nanoblocks and switchblocks. The nanoblocks are configured as either TPGs, BUTs, or ORAs. Three nanoblocks (i.e., one TPG, BUT and ORA) along with the switchblocks between them form a test group (TG) in which the TPG applies input signals to the BUT, and the ORA examines the output responses from the BUT to determine if there is a defect in the group. The whole fabric is partitioned into a set of TGs such that all BUTs inside them can be tested in parallel. We assume that nanoblocks along the edges can be accessed by external circuits, which in turn can serve as TPGs or ORAs for those blocks. In the nanofabric architecture described in [3], there are "long wires" that are used for interconnection purposes. We assume that such long wires can be used by external circuits to access blocks along the edges. Since the number of long wires is limited, multiple configurations are needed to access each of these blocks.

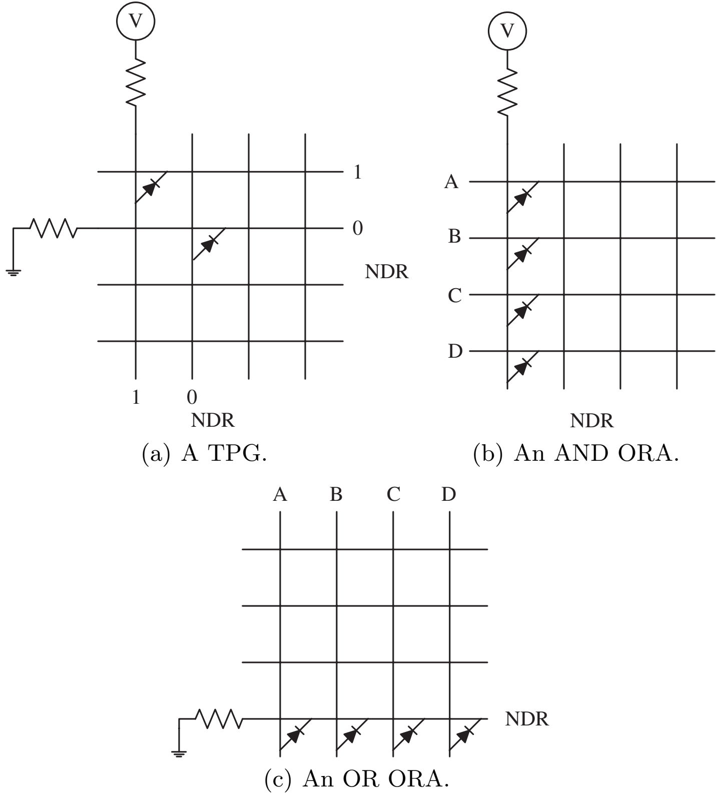
Our strategy relies on a set of fault detection configurations (FDCs) where different faults of the BUT can be tested. The proposed configurations can provide 100% fault coverage for any stuck-at, stuck-open, bridging, and connection faults in the nanoblocks. The details of these configurations, referred to FDC-1, are discussed in Sect. 5. A BUT is deemed to be defect-free only if it operates correctly in all FDC-1 configurations. In addition to FDC-1 configurations, another set of configurations is proposed to target faults in switchblocks (referred to as FDC-2). Switchblock testing is discussed in Sect. 6.

The test procedure consists of a sequence of test phases, where each phase consists of the following steps: (1) partition the nanofabric into TGs, (2) configure the TGs into one of the FDCs, (3) apply the test and read outputs of the ORAs, and (4) repeat from step (2) until all FDCs that are compatible with the current set of TGs are applied. Multiple test phases are needed to test each nanoblock and switchblock in the fabric. Each configuration uses only one test pattern. When the test procedure is completed, a defect map can be constructed. The test procedure is discussed further in Sect. 4.2.

The method used to access ORAs affects the overall testing time. If long wires, as described in [1], are used to read the outputs of ORAs, due to restricted interconnect resources, it may not be possible to access all the ORAs simultaneously. This will make the access time dependent on the size of the nanofabric. However, in the absence of manufactured nanofabric chips, it is not clear how much time will be needed to access ORAs. Contactless measurement and testing techniques [23], e.g., electron beam testing [24] and massive observability [25], may be used to reduce the ORA access time. Embedded CMOS optical sensors may also facilitate ORA readout [26].

4.1 BIST Architecture

Figure 7a shows the structure of a TPG. In the proposed fault detection configurations, a BUT only needs all-“1” and all-“0” input signals, thus the TPG can be very simple. A TPG provides “1” and “0” on both output sides. Every output port is connected to a molecular latch, which provides pull-up and pull-down paths to the down-streaming block.



(a) A TPG.

(b) An AND ORA.

(c) An OR ORA.

Fig. 7. Nanoblocks configured as TPG and ORA

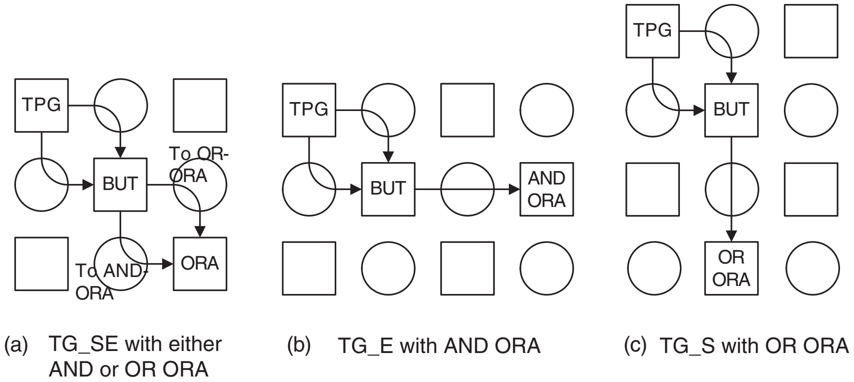


Fig. 8. Illustrating the need for different TGs

Figure 7b, c shows the structures of the ORAs. We carefully designed the BUTs such that the outputs of a defect-free BUT are identical, either all-"1" or all-"0". Therefore an ORA is simply a k-input AND gate or a k-input OR gate. Due to the restrictions of the CAEN architecture, an AND gate can only accept inputs on its east side for SE blocks (or west side for NW blocks), and an OR gate can only accept inputs on the north side for SE blocks. This restriction implies that three different types of TGs are needed such that all FDCs can be applied. As shown in Fig. 8a, in TG_SE the ORA is to the SE of the BUT, and it can be either an AND or an OR gate, provided that the outputs come from the appropriate side of the BUT. Similarly, the ORA in TG_E (Fig. 8b) is an AND gate and the ORA in TG_S (Fig. 8c) is an OR gate. which will be discussed in Sect. 6. We assume that the results of the ORAs can be read out using an access mechanism that is used for configuring the fabric. A similar assumption is made in recent work on nanofabric testing [1, 19].

Due to the connectivity restrictions shown in Fig. 4, none of the three TGs can be used for all FDCs. To achieve full fault coverage, we need a separate test procedure for each type of TG, which results in three partial defect maps. An overall defect map can then be derived from these partial defect maps. A nanoblock is considered to be *fully-recovered* or defect-free only if it is defect-free in all the three partial defect maps; otherwise a nanoblock is considered as *partially-recovered* if it is defect-free in any of the partial defect maps. A partially-recovered nanoblock can be used in some specific applications but its complete functionality cannot be guaranteed.

4.2 The BIST Procedure

The BIST procedure is shown in Fig. 9. In Step 2, TGs are allocated so that the BUTs can be tested in parallel. A *TG allocation* is a high level configuration for the entire nanofabric defining how TGs are created. The number of TG allocations is independent of the size of the fabric. For TG_SE, three

```
BIST Procedure: input: type of test group (TG)

1. while not all nanoblocks and switchblocks are tested
2.   partition the fabric into TGs;
3.   while not all compatible FDCs are applied
4.     apply one FDC;
5.     run the test;
6.     read out ORA responses;
7.   end while (FDC)
8. end while (nanoblocks)
9. generate the defect map.
```

Fig. 9. BIST procedure for any given TG

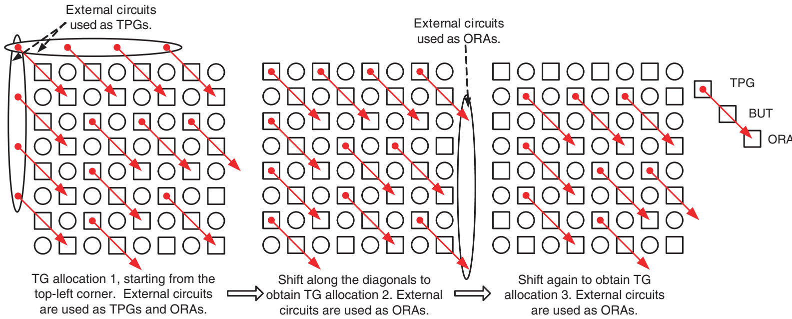


Fig. 10. TG allocations for TG_SE

```
Overall BIST Algorithm:

1. Run the BIST procedure using TG_SE;
2. Run the BIST procedure using TG_E;
3. Run the BIST procedure using TG_S;
4. Run the BIST procedure using TGs only for switchblock testing;
5. Generate an overall defect map;
6. Run the adaptive recovery procedure to further improve recovery.
```

Fig. 11. Overall BIST algorithm

allocations are needed, as shown in Fig. 10. A TG is represented by an arrow that starts from the TPG and ends at the ORA. Arrows whose start or end is outside the fabric represent those TGs that use external circuits as TPGs or ORAs. We simply shift all the arrows along the diagonals to obtain the three allocations. Similarly, for TG_S and TG_E, four allocations are needed.

In Step 9, initially all nanoblocks and switchblocks are deemed to be defective. If a BUT produces the correct outputs for all the applied FDCs, it is deemed to be defect-free.

A nanoblock cannot be recovered if any nanoblock or switchblock in the same TG is defective. However, we can use recovered nanoblocks and/or external circuits that are not adjacent to, but are reachable from the candidate defective block, as its TPG and ORA. This procedure, referred to as *adaptive recovery*, can greatly improve recovery and it is discussed in detail in Sect. 8. The overall BIST algorithm is shown in Fig. 11.

5 FDC-1: Fault Detection Configurations for Nanoblocks

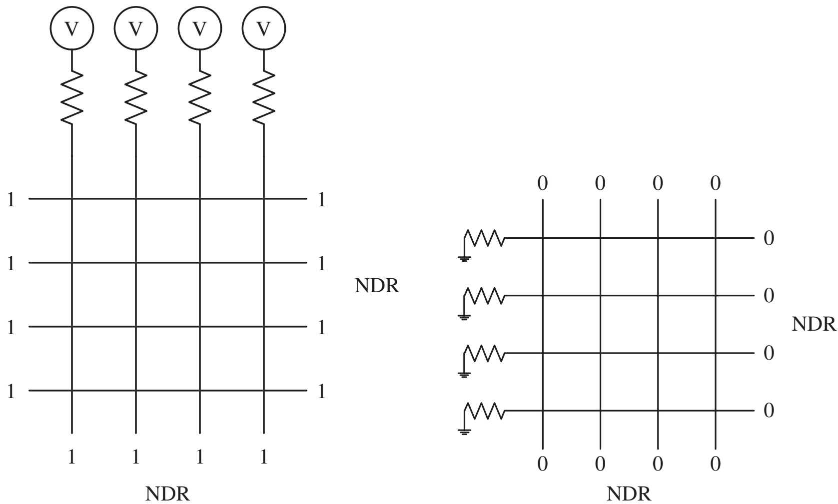
FDC-1 configurations configure the TGs into different circuits to provide 100% fault coverage for stuck-at, stuck-open, bridging, and connection faults in nanoblocks. These configurations are classified into five categories.

In the FDC-1 configurations, we make the following assumptions based on published papers on nanofabrics: (1) Each output port has an associated inline NDR latch, whose state can be reset to "0" [18], and (2) Resistors attached to Gnd have much smaller resistance that those attached to V_{DD} [3].

5.1 Category 1: FDCs for Stuck-At and Stuck-Open Faults

In Category 1(a) illustrated in Fig. 12a, all inputs are connected to V_{DD} or "1", therefore all outputs should be high for a defect-free BUT. The inline NDR molecular latches are initially set to "0" by adjusting V_{ref} [3]. For a defect-free BUT, since each output port is connected to an inline NDR, the latches are driven to "1". However, a line with a stuck-at-0 and/or stuck-open fault will fail to drive its associated NDR to the "1" state. By using a k-input AND gate as an ORA, any stuck-at-0 and stuck-open fault on a line can be detected. Category 1(a) includes two configurations, one corresponding to TG_SE and the other corresponding to TG_E. The TG, BUT and ORA for TG_SE are shown in Fig. 8.

Similarly, in Category 1(b) shown in Fig. 12b, all inputs are connected to Gnd or "0". With a k-input OR gate, any stuck-at-1 fault can be detected.



(a) Category 1(a): stuck-at-0 and stuck-open faults.

(b) Category 1(b): stuck-at-1 faults.

Fig. 12. BUTs for Category 1

Category 1(b) also has two configurations, out of which one configuration is compatible with TG_SE and the other is compatible with TG_S.

5.2 Category 2: FDCs for Connections of Forward-Biased Diodes and AND-Bridging Faults

Category 2 includes k configurations. In each configuration, one of the vertical wires is connected to V_{DD} and the other vertical wires are connected to "0". All horizontal wires are connected to Gnd. The k junctions along the vertical wire that is connected to V_{DD} are configured to "on" and are forward-biased. The outputs are "1" for a defect-free BUT. If a forward-biased diode is defective, the horizontal wire attached to it becomes "0". With a k-input AND gate ORA, we can test all the $k \times k$ cross-points.

The above category can also detect AND-bridging faults among the vertical wires (v/v). For a given configuration, if the wire connected to V_{DD} is shorted to any of the other vertical wires, the output will become "0" and can be detected by the ORA. All the k configurations together can test any AND-bridging faults between vertical wires.

AND-bridging faults involving a vertical wire and a horizontal wire (v/h) can also be detected using the FDC in Category 2. If any of the vertical wires connected to "0" is shorted to any of the horizontal wires, the output becomes "0" and the error can be detected. The k configurations for this FDC can together detect any AND-bridging faults of this type.

Configurations in Category 2 can only be used with TG_E because an AND ORA is used and the output responses come from the W-side of the BUT. Hence TG_S and TG_SE cannot be used for this category.

5.3 Category 3: Reverse-Biased Diodes

In Category 3, all molecular switches are configured to be closed. Horizontal wires are connected to "1" and vertical wires to "0". Therefore all the cross-points are reverse-biased and the output should be "1" on the east side and "0" on the south side. If any of the reverse-biased diodes is defective and has a small enough resistance to bridge the wires forming this cross-point, the output on the east side will be pulled down to "0" (AND-bridging), or the output on the south side will be pulled up to "1" (OR-bridging). By using an AND/OR gate we can detect defective reverse-biased diodes.

A total of two configurations are needed for this category, one using TG_E to observe W-side outputs and the other using TG_S to observe S-side outputs.

5.4 Category 4: AND-Bridging Fault Among Horizontal Wires (h/h)

Category 4 is the same as Category 3, with the difference that the vertical wires are connected to V_{DD}. The correct output on a wire is "1". For a given

Table 1. Summary of proposed FDC-1 configurations

Fault model	1a	1b	2	3	4	5
stuck-at-0	x					
stuck-at-1		x				
open-line	x					
AND-bridging (v/v, v/h)			x			
AND-bridging (h/h)					x	
OR-bridging						x
cross-points (forward)			x			
cross-points (reverse)				x		
# of configurations needed	2	2	k	2	k	$2k$
Type of TG	SE, E	SE, S	E	E, S	SE	SE, S

configuration, if the horizontal wire connected to "1" is shorted to any of the other horizontal wires, assuming AND-bridging fault, the output will become "0" because there are pull-down resistors attached to V_{DD}. A k-input AND ORA can detect this fault. This category requires the use of TG_SE. The k configurations can together detect any AND-bridging faults in nanoblocks.

5.5 Category 5: OR-Bridging Fault

Category 5 is used to detect OR-bridging faults between any of the nano wires. Only one wire is connected to "0" and all other wires are connected to V_{DD} or "1". If the "0"-wire is bridged to any other wire, it will be pulled up to "1". We only need to monitor the voltage level of this single wire using a 1-input OR gate. Clearly, $2k$ configurations are needed to test all the possible OR-bridging faults. Note that TG_SE and TG_S are compatible with this category.

In summary, a total of $4k + 6$ configurations are needed to test for the stuck-at, stuck-open, bridging and defective cross-points. Table 1 lists the faults indicated with x entries that each category can detect.

6 FDC-2: Fault Detection Configurations for Switchblocks

FDC-1 configurations only address faults in nanoblocks. As shown in Fig. 4, switchblocks and nanoblocks are closely coupled with each other. Hence, if a test group generates an error for a given configuration, it is difficult to ascertain whether switchblocks or nanoblocks are faulty. On the other hand, if a test group is error-free for a given set of configurations, and if all the switchblocks and nanoblocks in the test group are exercised adequately by these configurations, we can conclude that these blocks are all defect-free.

In this section, we describe an additional set of configurations, referred to as FDC-2, to provide 100% coverage for switchblock faults.

As discussed in Sect. 2.2, a switchblock contains $4k^2$ cross-points and provides four data flow directions. For SE blocks, directions WS and NE each use k^2 cross-points, while directions NS and WE each use $2k^2$ cross-points. All the four directions need to be thoroughly tested.

Similar to [19], we use a walking binary sequence to test switchblocks. As shown in Fig. 15, to test the k^2 connections used by the WS direction, a sequence of test stimuli, 1000, 0100, 0010 and 0001, is applied to the inputs of the 4×4 switchblock. If the switchblock is defect-free, the same sequence should appear in the outputs. After the test sequence is applied, the configurations are shifted, as shown in Fig. 16. The walking sequence is repeated k times so that all connections are tested. Therefore a total of k^2 configurations are needed. This test set ensures that every connection in the switchblock is tested in both the "on" and "off" configurations. It provides 100% fault coverage for single stuck-line, bridging, and connection faults [19].

6.1 Testing the WS Dataflow Direction

In FDC-1 Category (2), the input to the BUT is simply a walking sequence of ones. If the TPG is configured to provide this sequence to the BUT, the switchblock that connects the TPG to the BUT, and provides the WS dataflow direction, can be tested; see Fig. 17. A total of k^2 additional configurations need to be added to Category (2). Only the TPG and the switchblock under test are changed in these configurations. If the test group operates correctly in this category, we can conclude that the k^2 connections of the switchblock that are used by dataflow direction WS are defect-free.

6.2 Testing the NE Dataflow Direction

In FDC-1 Category (4), the input to the BUT is also a walking sequence of ones. Therefore, we can add k^2 additional configurations to Category (4) to test the swtichblock that connects the TPG and the BUT and provides the NE dataflow direction, as shown in Fig. 18.

6.3 Testing the NS and WE Dataflow Directions

To test the NS and WE directions, FDC-1 category (2) and (4) are used again. However, we need two new types of TGs, where the TPG is to the north and west of the TPG, respectively.

As shown in Fig. 19, a new type of TG, TG_SE_W, is used in order to test the WE direction. The BUT is configured as FDC-1 category (4). Although the WE direction uses $2k^2$ connections, we still need k^2 additional configurations. This is because among the $2k^2$ connections, the k^2 connections between the horizontal wires from the west nanoblock and the vertical wires from the south nanoblock are tested in the WS direction; only the other k^2 connections

between the horizontal wires from the west nanoblock and the vertical wires from the north nanoblock need to be tested (Fig. 20). Figure 21 illustrates how the NS direction is tested.

In summary, a total of $4k^2$ configurations are needed to test the four dataflow directions that a switchblock provides. Each direction requires k^2 configurations. Directions WS and NE are tested in TG_E and TG_SE respectively, and therefore can share TG allocations with FDC-1. Directions NS and WE, however, must be tested separately using TG_SE_W and TG_E_N.

7 The Detection of Multiple Faults

In this section, we discuss the effectiveness of the FDC-1 and FDC-2 configurations in detecting multiple faults in nanoblocks and switchblocks. If none of the proposed configurations can detect a specific combination of multiple faults within one test group, then there may be two cases: (1) more than one block within the test group are faulty and these faults mask each other; (2) faults inside one nanofabric or switchblock mask each other, such that the output response of the faulty block is the same as the error-free response.

While Case (1) cannot be ruled out, it is not likely in practice for the proposed BIST approach. Since the test groups are simple and each test group contains only a small number of blocks (one TPG, BUT, ORA and the associated switchblocks), any faulty block will cause the whole group to generate incorrect outputs. The probability for a faulty test group to pass a test is extremely low. Moreover, the nanoblocks in a test group are also used in other test groups in different roles, which facilitates fault detection. Therefore, in this section we will only consider Case (2).

Since FDC-2 is an exhaustive functional test for the switchblocks, in which each cross-point is tested for both on and off states, FDC-2 can detect any combinations of faults in a switchblock. If a switchblock passes all tests provided by FDC-2, then it is guaranteed to be fault-free for any number of connection faults.

Next, we consider whether multiple faults in a BUT can mask each other and become undetectable. Intuitively, since the BUT is intensively exercised by $4k+6$ FDC-1 configurations, the probability is extremely low that a specific combination of multiple faults inside the BUT can escape these tests.

7.1 Multiple Stuck-At Faults

In Category 1 of FDC-1, since the ORA directly observes the output responses of the nanowires, any number of stuck-at faults on horizontal or vertical nanowires can be detected. Therefore, FDC-1 can provide 100% coverage for multiple stuck-at faults in one nanoblock. Moreover, stuck-at faults on these wires cannot be masked by other faults, such as bridging faults and cross-point faults, in the same BUT because the responses of the nanowires

are directly observed. This implies that whenever a BUT contains stuck-at faults, it can always be identified as being faulty.

7.2 Multiple Bridging Faults

Multiple bridging faults inside a BUT, including AND- and OR-bridging faults among all nanowires, can also be detected by FDC-1. When detecting bridging faults using Category 2, 4 and 5, respectively, k configurations are applied to the BUT, and each nanowire is tested separately to see if it is bridged to any other nanowire. Therefore, if there are multiple pairs of nanowires that are bridged, they can be detected separately.

Bridging faults that involve more than two nanowires can also be detected. For example, as in Category 5 (Fig. 14b), only one nanowire is connected to 0 and all other nanowires are connected to 1. If the nanowire connected to 0 is bridged to multiple nanowires, as long as it is OR-bridged to one nanowire, it will be detected. This is also true for Category 2 and 4. Note that in this work, we only consider AND- and OR-bridging faults. We have not considered the physical phenomenon of resistive bridging faults.

A defective diode behaves either like a stuck-open switch or a stuck-on switch, depending on the bias voltage across it. If a defective diode is reverse-biased, then the two nanowires associated with this diode are either AND-bridged or OR-bridged with each other. Therefore, bridging faults can be detected even if there are defective reverse-biased diodes. Defective forward-biased diodes are simply stuck-open switches, whose existence will not affect the detection of any other fault. Hence, multiple bridging faults are detectable even when there are defective diodes in the same nanoblock.

In some cases, multiple bridging faults in a BUT may form feedback loops and turns the BUT into a sequential circuits. The detection of such faults is complex and we will not consider it in this work.

7.3 Multiple Cross-Point Faults

In Category 2, the diodes associated with the nanowire that is connected to V_{DD} are tested if they operate correctly when forward-biased. If other diodes in the same nanoblock are defective, they behave as either stuck-open or stuck-on switches. It can be seen from Fig. 13a that these defective diodes will not affect the test of the forward-biased diodes, even if they cause their associated nanowires to be AND-bridged or OR-bridged. In Category 3, all diodes are reverse-biased and tested in parallel. If there are multiple defective diodes in the nanoblock, then they may form feedback loops in the nanoblock. We do not consider such faults in this work.

In summary, FDC-1 can detect most combinations of multiple faults in a single nanoblock. FDC-1 can provide 100% coverage for multiple stuck-at faults, 100% non-feedback bridging faults among nanowires, and 100% coverage for multiple cross-point faults. Feedback bridging faults and feedback cross-point faults are beyond the scope of this work.

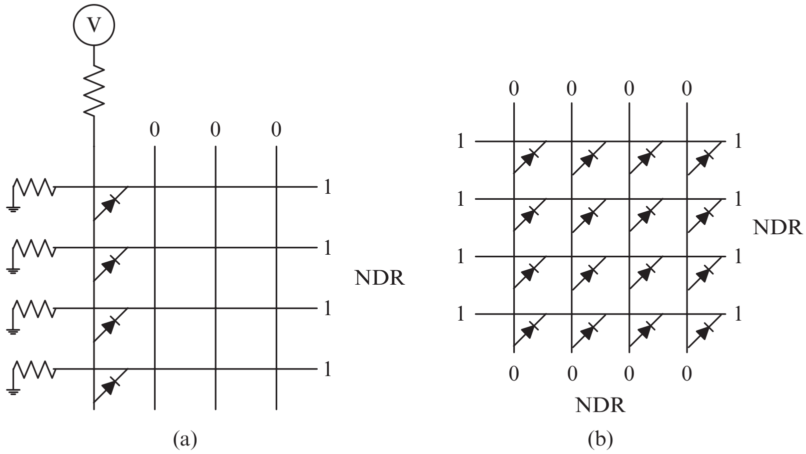


Fig. 13. (**a**) Category 2: forward-biased diodes faults and AND-bridging faults (v/v, v/h). (**b**) Category 3: reverse-biased diodes faults

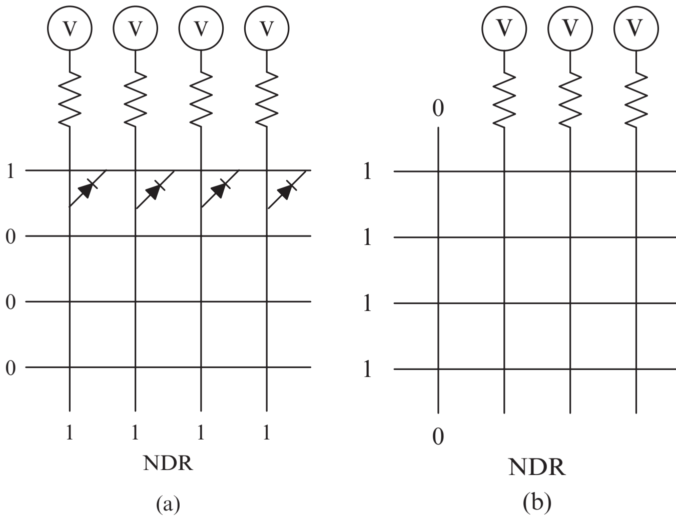


Fig. 14. (**a**) Category 4: AND-bridging fault among horizontal wires (h/h). (**b**) Category 5: OR-bridging fault

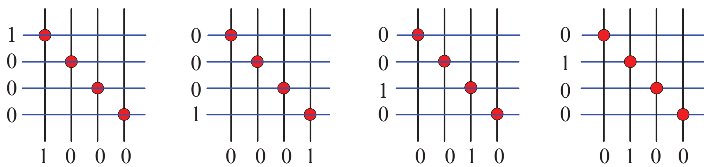


Fig. 15. A walking sequence of ones is applied to the inputs and appears in the outputs

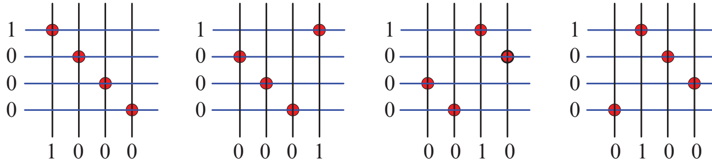


Fig. 16. The configuration of the switchblock is shifted after all test patterns have been applied and the process is repeated until all connections have been tested

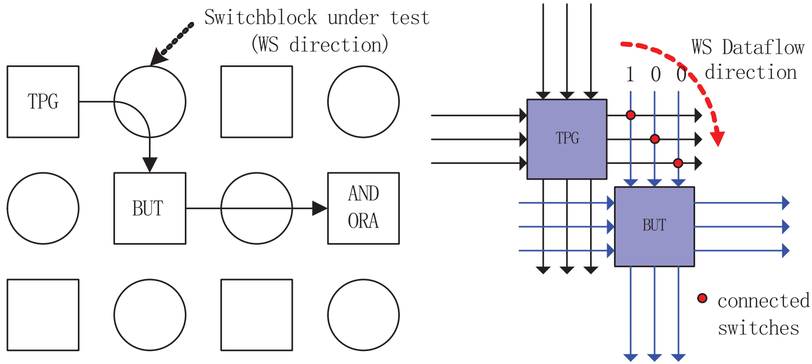


Fig. 17. Testing the WS direction of a switchblock: The BUT is configured as FDC-1 category (2), except that all inputs are from the switchblock under test. In the k^2 configurations, the TPG and the switchblock under test are changed so that the walking sequence are repeated k times and the k^2 connections are tested. FDC-1 category (2) only works in TG_E

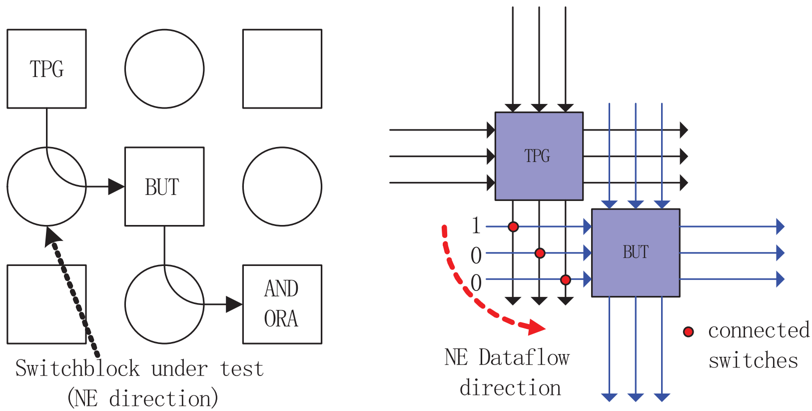


Fig. 18. Testing the NE direction of a switchblock: The BUT is configured as FDC-1 category (4), except that all inputs are from the switchblock under test. In the k^2 configurations, the TPG and the switchblock under test are changed so that the walking sequence are repeated k times and the k^2 connections are tested. FDC-1 category (4) only works in TG_SE

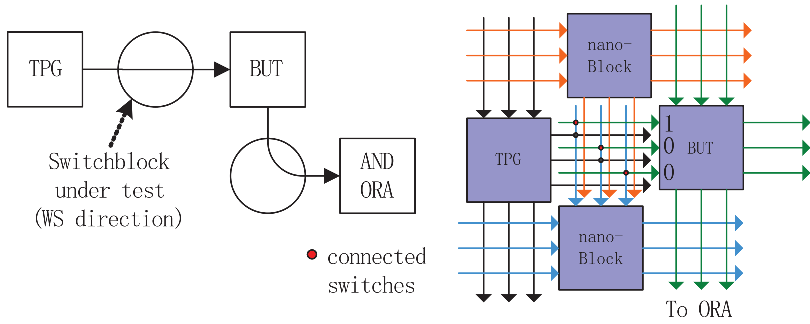


Fig. 19. Testing the WE direction of a switchblock: The BUT is configured as FDC-1 category (4). These configurations need a new TG type, i.e., TG_SE_W

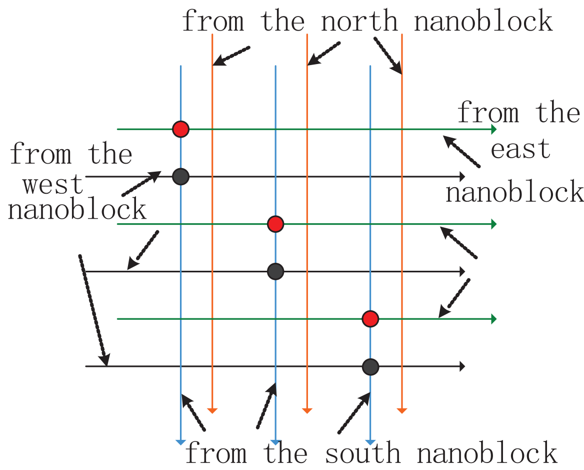


Fig. 20. Only connections between the vertical wires from the north nanoblock and the horizontal wires from the west nanoblock need to be tested in the WE direction

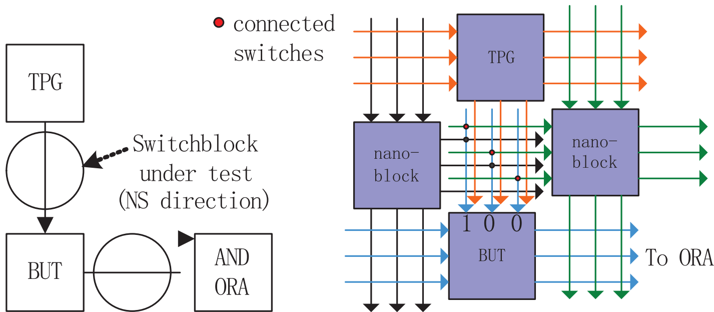


Fig. 21. Testing the NS direction of a switchblock: The BUT is configured as FDC category (2). These configurations need a new TG type: TG_E_N

8 Adaptive Recovery Procedure

As discussed in Sect. 4.2, a defect-free nanoblock or switchblock B cannot be recovered if there are faulty blocks in the same TG. However, if other blocks outside the TG can be used to form a new TG to test B, this block may be recovered. For example, as shown in Fig. 22, a new TG that is similar to TG_E but consists of non-adjacent nanoblocks is created to test the BUT whose west-side neighbor is defective. In this section, we propose an adaptive recovery procedure that dynamically generates configurations to recover blocks that are not recovered by Steps (1)–(5) in Fig. 11.

The adaptive recovery procedure only tries to recover blocks that have a high probability to be defect-free. These blocks are referred to as *candidates*. A candidate nanoblock or switchblock must have at least one of its surrounding blocks (1) fully-recovered, or (2) partially-recovered and the failing tests do not involve the candidate block itself. This is because a defective block will cause all its surrounding blocks to fail the tests whose TGs contain it.

In the adaptive recovery procedure, first the partial and overall defect maps are searched and candidates are identified. Then a search procedure is invoked for each candidate. Paths that start from the current candidate block and lead to a recovered block, another candidate block, or an edge of the nanofabric are searched, such that new TGs can be formed. The same procedure as shown in Fig. 9 is then executed to test the current candidate. To avoid using known defective blocks, the paths only consist of fully or partially-recovered blocks and other candidate blocks, as shown in Fig. 22. In this work, a simple back-trace search algorithm has been implemented.

The nanofabric is reconfigured multiple times until either all candidates are tested or a satisfactory recovery level is achieved. In each configuration, several candidates can be tested in parallel. The adaptive recovery procedure is summarized in Fig. 23.

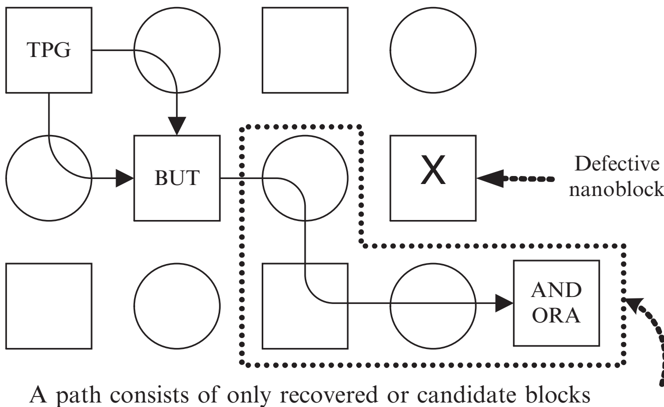


Fig. 22. A TG_E variant TG that uses non-adjacent nanoblocks

```
Adaptive Recovery Procedure:

 1. identify candidates;
 2. while (not all candidates are tested) and (satisfactory recovery not reached)
 3.    generate TGs for each remaining candidate;
 4.    while not all compatible FDCs are applied
 5.      apply one FDC;
 6.      run the test;
 7.      read out ORA responses;
 8.    end while (FDC)
 9. end while (candidates)
```

Fig. 23. The adaptive recovery procedure

9 Recovery Analysis

In order to better understand the effectiveness of the proposed BIST procedure, we derive simple upper and lower bounds of the recovery. As shown in Fig. 26, a defective switchblock causes two of its surrounding nanoblocks to become non-recoverable. Thus the minimum recovery is obtained when all defective blocks are switchblocks and each of them causes two nanoblocks to become unrecoverable. For a $n \times n$ nanofabric with defect density d, there are a total of $(1-d)n^2$ defect-free blocks, among which $2dn^2$ defect-free nanoblocks are not recovered in the worst case. A lower bound LB on the recovery is therefore given by

$$LB = \frac{(1-d)n^2 - 2dn^2}{(1-d)n^2} \times 100\% = \frac{1-3d}{1-d} \times 100\%$$

For $d = 0.1$ ($d = 0.2$), LB is 78% (50%). Figure 24 shows how LB varies with d.

The best recovery is achieved when all defective blocks are nanoblocks and they are not close to each other so that no switchblocks are rendered non-recoverable. Hence the upper bound is simply $UB = 100\%$. However, this upper bound is meaningful only when the defect density is below a maximum value d_{max}, such that each switchblock has at least three defect-free surrounding nanoblocks. If a switchblock has more than two defective surrounding nanoblocks, the probability that it is not recoverable is 1/3 (i.e., the two defective nanoblocks are both at the inputs or outputs of the four dataflow directions). To derive d_{max}, consider a nanofabric consisting of m nanoblocks and m switchblocks with defect density d. Since each nanoblock is shared by four switchblocks, there should be at least $\frac{3m}{4}$ defect-free nanoblocks. Hence $m - d_{max} \times 2m = \frac{3m}{4}$ and $d_{max} = 12.5\,\%$. If $d > d_{max}$, the upper bound is below $100\,\%$ and it depends on both the defect density and the distribution of the defective blocks; it is difficult to derive a closed-form expression for the upper bound.

In some special cases, even when d is higher than $12.5\,\%$, the recovery can still be 100%. Figure 25 illustrates a special case, in which all defective

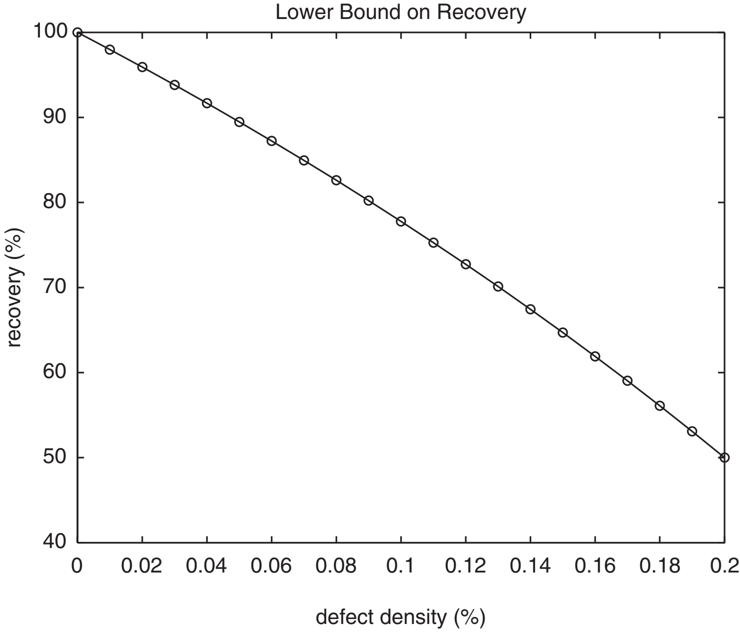


Fig. 24. Lower bound on the recovery

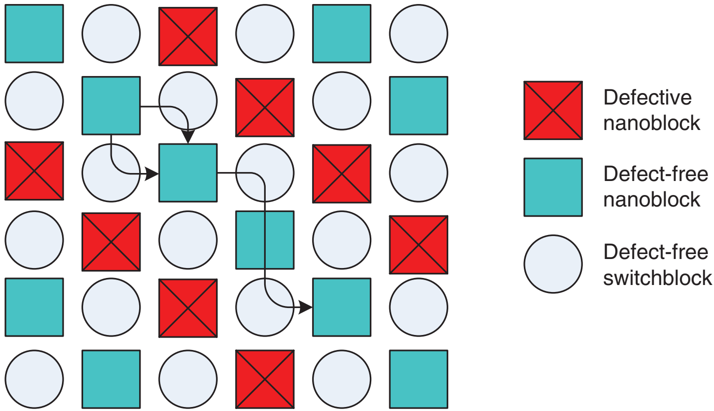


Fig. 25. Special case of 100% recovery

blocks are nanoblocks and they only appear along alternate major diagonals. In this case, the defect density $d = 25\%$. Using adaptive recovery as shown in Fig. 25, a recovery of 100% is achievable. Note that all switchblocks are partially-recovered.

10 Simulation Results

In this section, we present simulation results for the proposed BIST approach. The BIST procedure was implemented in C++ and an array of bits is used to represent the nanofabric. Blocks are randomly selected to be faulty and a simple array look-up is performed to determine if a nanoblock or a switchblock is defective.

Figure 26 shows the defect maps obtained from simulation on a 10×10 nanofabric with 10% defect density. The overall recovery, defined as the percentage of the number of fully or partially-recovered blocks to the total number of defect-free blocks, is 92.1%. There are 11 defective blocks, 59 fully-recovered blocks, and 23 partially-recovered blocks. Among the 82 recovered blocks, 13 blocks are recovered by the adaptive recovery procedure.

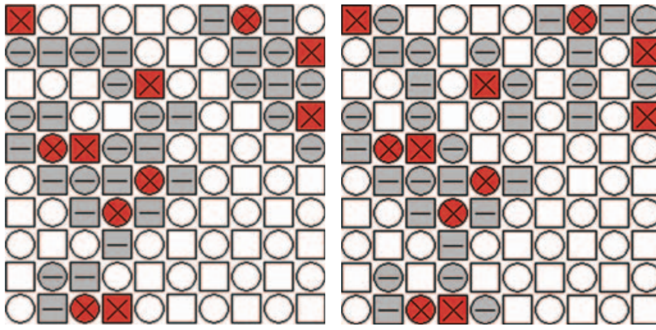
In TG_SE and TG_E, both nanoblocks and switchblocks are tested. Therefore the corresponding partial defect maps contain recovered nanoblocks and switchblocks. In other partial defect maps, there are only recovered nanoblocks or switchblocks. The adaptive recovery procedure is performed as the last step, so only the overall defect map shows blocks recovered by it.

As shown in Fig. 26, a defective switchblock consistently renders the nanoblocks to the east or south of it non-recoverable, because these nanoblocks, when configured as BUTs, cannot receive input signals from TPGs through the defective switchblock. A defective nanoblock, however, only causes its surrounding switchblocks to be partially-recovered. These switchblocks fail the tests involving the defective nanoblock, but pass other tests not involving it. Therefore, a defective switchblock has greater impact on recovery than a defective nanoblock.

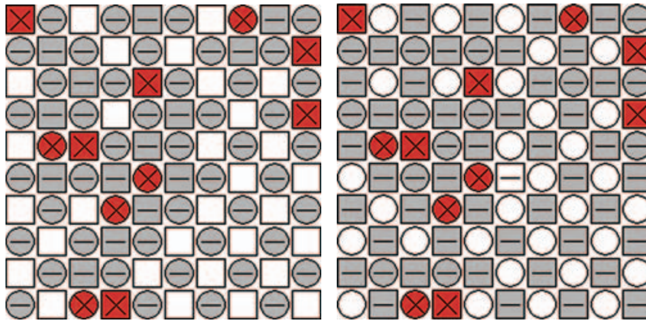
Figure 27 shows how the recovery varies with defect density and the size of the nanofabric. For each value of defect density and nanofabric size, 100 simulations were performed and the average recovery as well as the maximum and minimum recovery values were determined. As shown in Fig. 27a, the average recovery is almost independent of the size of the nanofabric. This implies that the BIST procedure and the recovery algorithm scale well with the size of the nanofabric. Moreover, the average recovery is inversely proportional to the defect density. This is obvious because a defective block, especially a defective switchblock, affects the functionality of its neighbors.

Figure 27b, c show that the recoveries of small nanofabrics are more likely to be affected by the defect distribution. With the same defect density, if relatively more switchblocks than nanoblocks are defective, the recovery is likely to be lower. In large nanofabrics, the number of defective nanoblocks and the number of the defective switchblocks are more likely to be the same. The recovery values lie within the upper and lower bounds derived in Sect. 9. In particular, the recovery obtained is close to the lower bound. The minimum recoveries when $d = 0.1$ and $d = 0.2$ are about 78 and 56%, respectively.

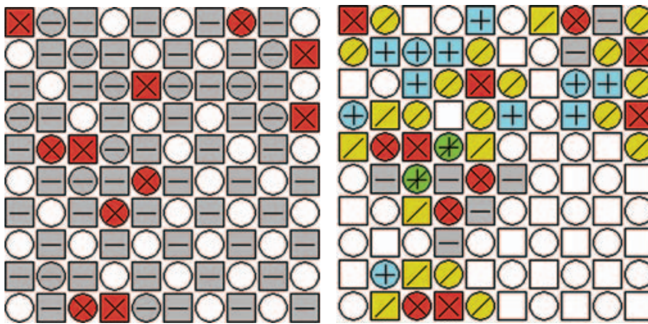
Figure 28 shows how clustered defects affect recovery. We use two clustered defect models: (1) defective blocks are in the same row or column, and (2)



(a) Defect map for TG_SE (b) Defect map for TG_E



(c) Defect map for TG_S (d) Defect map for TG_E_N



(e) Defect map for TG_SE_W (f) Overall defect map

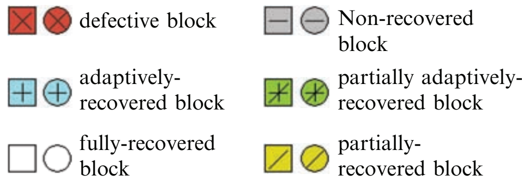
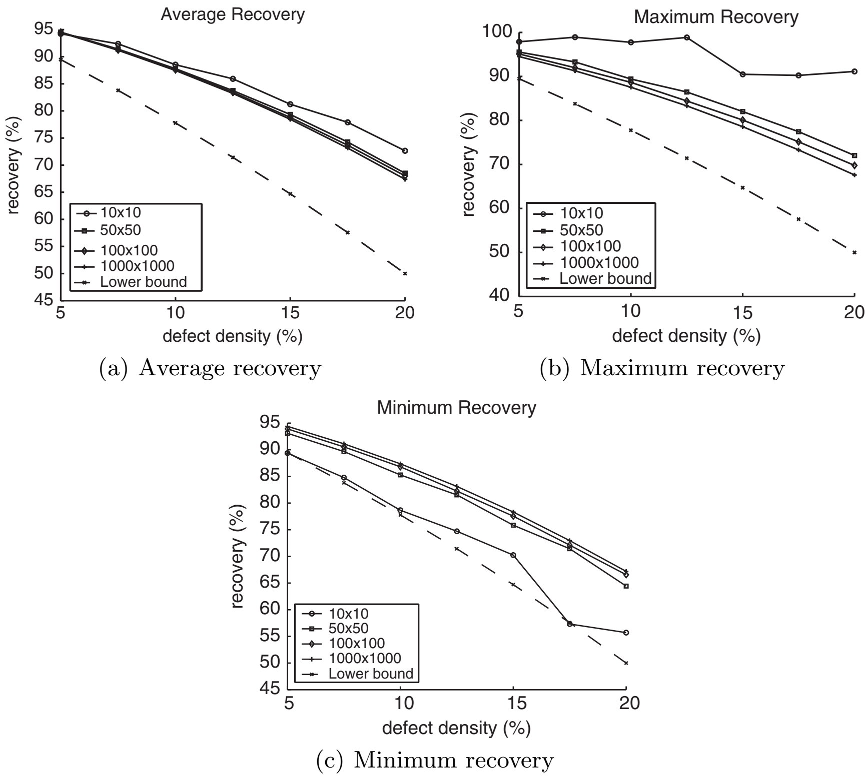


Fig. 26. Defect maps for a 10×10 fabric with 10% defect density



(a) Average recovery (b) Maximum recovery

(c) Minimum recovery

Fig. 27. Recovery results

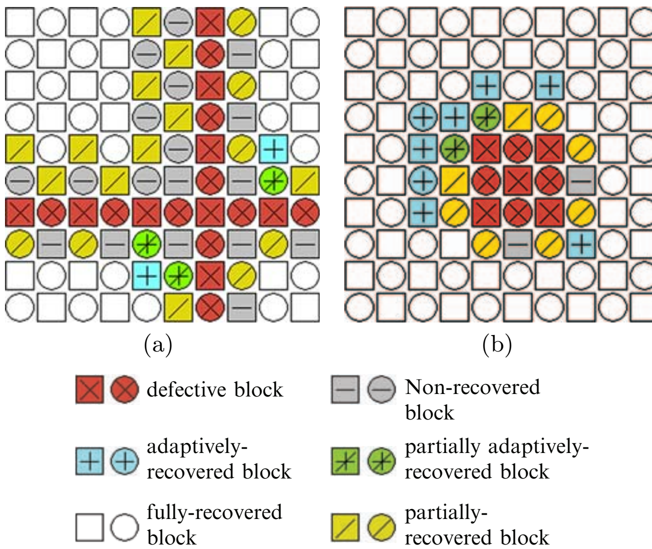


Fig. 28. Clustered defective blocks

defective blocks are inside a rectangle. The recoveries of Fig. 28a, b are 77.8 and 97.8% respectively, and the defect densities are 19 and 16% respectively. Only the nanoblocks and switchblocks adjacent to the edges of the cluster cannot be recovered. Clustered defects do not influence the effectiveness of the BIST approach.

11 Conclusions

We have presented a new built-in self-test strategy for the CAEN-based nanofabric. The proposed BIST procedure configures the nanoblocks into test pattern generators (TPGs), blocks under test (BUTs) and output response analyzers (ORAs), which in turn form test groups where the BUTs and switchblocks are tested. A test group only contains three nanoblocks and the switchblocks between them, therefore the algorithm is able to handle nanofabric systems with a large number of devices and high defect densities. A set of configurations have been presented to detect various faults in both nanoblocks and switchblocks; these configurations provide 100% fault coverage for stuck-at, stuck-open, bridging, and connection faults. All multiple stuck-at faults, as well as non-feedback bridging and crosspoints faults can also be detected. The complexity of this algorithm and its recovery capability are independent of the size of the nanofabric (if the time required to read out output responses is O(1)). The recovery procedure can also utilize recovered defect-free blocks in an adaptive fashion.

References

1. M. Mishra and S. Goldstein, "Defect Tolerance at the End of the Roadmap," in *Proc. International Test Conference*, 2003, pp. 1201–1210.
2. E. J. Nowack, "Maintaining the Benefits of CMOS scaling when Scaling Bogs Down," *IBM Journal of Research and Development*, no. 2/3, Mar.-May 2002.
3. S. C. Goldstein and M. Budiu, "NanoFabrics: Spatial Computing Using Molecular Electronics," in *Proc. International Symposium on Computer Architecture*, 2001, pp. 178–189.
4. S. C. Goldstein and D. Rosewater, "Digital Logic Using Molecular Electronics," in *Proc. IEEE International Solid State Circuits Conference*, vol. 1, 2002, pp. 204–459.
5. M. Butts, A. DeHon, and S. C. Goldstein, "Molecular Electronics: Devices, Systems and Tools for Gigagate, Gigabit Chips," in *Proc. International Conference on Computer-Aided Design*, 2002, pp. 433–440.
6. M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler, "Molecular Electronics: From Devices and Interconnect to Circuits and Architecture," *Proc. IEEE*, vol. 91, Nov. 2003, pp. 1940–1957.
7. Y. Chen, G.-Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. StanleyWilliams, "Nanoscale molecular-switch crossbar circuits," *Nanotechnology*, vol. 14, Mar. 2003, pp. 462–468.

8. Nantero Inc., http://www.nantero.com/.
9. A. J. van de Goor, *Testing Semiconductor Memories: Theory and Practice.* ComTex Publishing, 1998.
10. C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-In Self-Test of Logic Blocks in FPGAs (Finally, A Free Lunch: BIST Without Overhead!)," in *Proc. IEEE VLSI Test Symposium*, 1996, pp. 387–392.
11. M. Abramovici, E. Lee, and C. Stroud, "BIST-based Diagnostics for FPGA Logic Blocks," in *Proc. International Test Conference*, 1997, pp. 539–547.
12. C. Metra, G. Mojoli, S. Pastore, D. Salvi, and G. Sechi, "Novel Technique for Testing FPGAs," in *Proc. Design, Automation and Test in Europe*, 1998, pp. 89–94.
13. S. J. Wang and T. M. Tsai, "Test and Diagnosis of Fault Logic Blocks in FPGAs," in *IEE Proceedings: Computers and Digital Techniques*, vol. 146, 1999, pp. 100–106.
14. M. B. Tahoori, E. J. McCluskey, M. Renovell, and P. Faure, "A multi-configuration strategy for an application dependent testing of FPGAs," in *Proc. IEEE VLSI Test Symposium*, 2004, pp. 154–159.
15. M. Tahoori and S. Mitra, "Techniques and algorithms for fault grading of FPGA interconnect test configurations," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, Feb. 2004, pp. 261–272.
16. W. B. Culbertson, R. Amerson, R. J. Carter, P. Kuekes, and G. Snider, "Defect Tolerance on the Teramac Custom Computer," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, 1997, pp. 116–223.
17. J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams, "A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology," *Science*, vol. 280, Jun. 1998, pp. 1716–1721.
18. S. C. Goldstein and D. Rosewater, "What Makes a Good Molecular-Scale Computer Device?" School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-02-181, Sep. 2002.
19. J. G. Brown and R. D. S. Blanton, "CAEN-BIST: Testing the NanoFabric," in *Proc. International Test Conference*, 2004, pp. 462–471.
20. Z. Wang and K. Chakrabarty, "Built-in Self-Test of Molecular Electronics-Based Nanofabrics," in *Proc. European Test Symposium*, 2005, pp. 168–173.
21. M. Tehranipoor, "Defect Tolerance for Molecular Electronics-Based NanoFabrics Using Built-In Self-Test Procedure," in *Proc. International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2005, pp. 305–313.
22. R. M. Rad and M. Tehranipoor, "SCT: An Approach for Testing and Configuring Nanoscale Devices," in *Proc. IEEE VLSI Test Symposium*, 2006 (to appear).
23. S. Sayil, D. V. Kerns, and S. E. Kerns, "A survey contactless measurement and testing techniques," *IEEE Potentials*, vol. 24, Feb.-Mar. 2005, pp. 25–28.
24. M. Vallet and P. Sardin, "Electrical testing for failure analysis: Ebeam testing," *Microelectronic Engineering*, vol. 49, 1999, pp. 157–167.
25. A. Mabrouk and A. Hubbard, "Design and implementation of an optical testing technique for VLSI chips using a potential-sensitive fluorescing dye," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1997, pp. 568–572.
26. S. Sayil, "All-Silicon Optical Contactless Testing Of ICs," *International Journal of Electronics*, vol. 89, 2002, pp. 537–547.

Chapter 3: Test and Defect Tolerance for Reconfigurable Nanoscale Devices

M. Tehranipoor and R. Rad

1 Introduction

According to international technology roadmap for semiconductors (ITRS) [1] scaling of CMOS technology will face many practical and theoretical difficulties within the next few years. In order to enhance the domain of information processing applications beyond CMOS capabilities and continue Moore's law, several technologies are being examined for future computing devices. Among these technologies, quantum devices, optical devices, biologically inspired devices, molecular devices, nanowire and carbon nanotube based devices are under intense investigation. High defect rate is expected to be the common problem for all these emerging technologies. Defect density of such technologies will be considerably higher than that of CMOS due to indeterministic fabrication processes and dominance of quantum effects at such scale. Dealing with such high defect densities requires wide research on new test and defect tolerance techniques that they are able to provide high defect tolerance while the amount of area overhead and test/configuration time are kept reasonable.

Among different proposed technologies, devices built based on crossbars of nanowires or nanotubes are most widely discussed in literature [1]. These crossbars can be used to implement memories [2] or memory-based logic (PLA or LUT) [3, 4]. Although feature sizes of individual nanowires (or nanotubes) is not much smaller than scaled CMOS wires (5–10 nm compared to 22 nm scaled CMOS), yet area efficient circuits can be designed using these nanoscale wires mainly because the switching task in this technology is performed by molecular switches which are much smaller than transistors used in CMOS devices [4, 5].

Several test and defect/fault tolerance methods have been proposed for devices built using crossbars of nanowires or nanotubes and molecular switches. Most of the proposed dynamic fault tolerance techniques are based on traditional techniques used in CMOS devices such as triple modular redundancy

(TMR), NMR, cascaded TMR (CTMR) and coding-based fault tolerance methods [6, 7]. However, redundancy based methods like TMR/NMR result in high area overhead for devices with very high defect rates. Coding-based methods also suffer from high area overhead due to implementation of coder and decoder circuits and also delay overhead due to delay of the coder-decoder circuits. As [7] suggests, coding methods can still provide good fault tolerance to some parts of the device such as address decoders in a memory or memory-based logic.

1.1 Defect Map Strategy

Static defect tolerance techniques for nanoscale crossbars have been proposed mostly based on defect avoidance through reconfiguration and defect map. The basic idea of such techniques was originated from Teramac experiments [8]. Teramac was a custom computer built upon reconfigurable blocks (i.e. FPGAs). It could tolerate defects in its components through reconfiguration and avoiding the defective components. This defect avoidance scheme required that the location of defective components to be identified during a pre-configuration test process and stored in a defect database called *defect map*. Similar defect avoidance methods are proposed to be employed for nanoscale devices [9, 10]. Also test methods have been proposed to find the location of faulty components in crossbar-based circuits and store them in defect map [11, 13, 14]. However, there are major difficulties associated with using defect map strategy in mass production of nanoscale devices. For instance, due to random nature of defect occurrence, there will be different defect maps for different chips. Even if we assume that it is possible to find defect locations through a test process, it will be very time consuming task for such high density devices.

Another issue related to defect map-based strategies is that defect map size for the device will be extremely large and it will be almost infeasible to store it on-chip. This will be prohibitively expensive since, for reliability purposes, a CMOS scale memory should be used to store the defect map. Shipping defect map of each chip along with the chip on a separate storage device does not also seem to be a practical solution. Coarse-grained techniques will not alleviate the concern either. This is because decreasing the test resolution and storing defect location information for larger blocks instead of low level molecular switches or wires, will result in considerable loss in utilizable blocks of the device. In other words, a large number of blocks will be considered as faulty and they will be avoided during reconfiguration phase. Hence, in this case, blocks of the device will be used very inefficiently.

Another problem associated with defect map strategy is the requirement of performing placement and routing (PNR) process for each chip separately because each chip will have a unique defect map. If defect map method is to be used to provide defect tolerance, placement and routing should be performed for each device. This results in high amount of time spent for PNR during the configuration phase of each device. Also configured devices might have

different performances since the blocks will be placed and routed differently in each chip due to different defect maps.

To avoid per chip placement and routing [15] proposed a different defect avoidance mechanism. Assuming that the configurable nanoscale array is made of $N \times N$ crossbars, authors of [15] suggest that based on defect probabilities of nanoscale components of the device, a $K \times K$ defect-free crossbar can be found within each $N \times N$ crossbar (where $K < N$). Therefore, to configure a circuit on the nanoscale device, it must be partitioned into blocks of size K. During configuration process, in order to implement each block, the $K \times K$ defect-free crossbar should be identified inside each $N \times N$ crossbar. This is suggested to be done by referring to the defect map. As a result, per-chip placement and routing will not be required using this method. However, it still relies on having defect location information. Meanwhile, for high defect rate situations, N might be very large in order to confirm the availability of a defect-free crossbar of size K. In other words, one will have to implement rather very large crossbars in order to make sure that an acceptable percentage of $N \times N$ crossbars in the device are usable as blocks of size K. And this is an important achievement.

1.2 Inherent Redundancy in Reconfigurable Devices

As discussed in FPGA literature, memory-based logic provides considerable amount of redundancy in each PLA or LUT of the device [17]. In other words, a configured application into a FPGA uses only a small amount of hardware resources provided in each block of the FPGA. Figure 1 shows the average utilization of blocks in FPGAs vs. different block sizes (block inputs and outputs) [17]. This fact proposes that there is always a large amount of redundancy in blocks of reconfigurable devices. We refer to this as inherent redundancy of the device. This redundancy can be used to provide the required defect tolerance for reconfigurable nanoscale devices. This can be done by searching for fault-free configuration of functions mapped onto each PLA or LUT. Hence, by doing so and identifying fault-free implementation of the functions in their assigned blocks, the requirement of moving functions into different blocks and performing per chip placement and routing can be removed. We will further discuss this in the next section.

1.3 Overview of the Proposed Methods

In this chapter, we propose novel defect tolerance methods for nanoscale devices with no dependence on defect map. It also enables avoiding per chip placement and routing. It is shown that per chip placement and routing can be avoided due to availability of sufficient redundancy in crossbar blocks so that defect-free implementation of the functions can be found. We first propose a method that simultaneously test and configure the reconfigurable nanoscale devices without identifying the exact location of defects. The second method

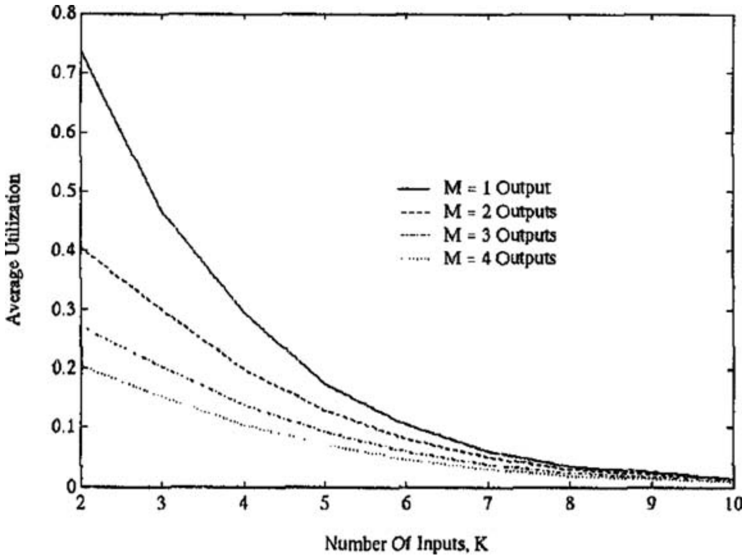


Fig. 1. Average utilization of FPGA PLA blocks (source: [17]). Utilization drops (*redundancy increases*) as the block size increases

we propose here searches redundant parts inside each block and configures the functions on defect-free sections of the blocks and tests the mapped function at the same time. The available redundancy of the configurable blocks (inherent redundancy) is utilized to provide defect tolerance. Hence, the amount of additional redundancy required for having acceptable yields would be small. We have performed analyses to evaluate the proposed methods for different defect probabilities that may occur in defect prone crossbar-based devices. Also, simulation programs have been developed to examine the method for different MCNC benchmarks implemented on such devices.

2 Simultaneous Configuration and Test (SCT) Method

As mentioned above, locating all defects inside a nano-device with very high density (10^{10} gate-equivalents cm^{-}2) and high defect density (up to 10%) will be a challenging and time consuming task. In this chapter, a method is proposed for testing and configuring circuits that can be used to avoid the time consuming process of locating all defects. We assume that the architecture offers rich interconnect resources and is able to provide efficient access to its logic blocks through its input/output interfaces. Such architectures are presented in literature [3, 9, 18].

The proposed method, in this chapter, is conceptually similar to those proposed for FPGAs [19], but we consider TPG and RA to be components of BIST circuit and to be implemented in CMOS scale to provide tests and

analyze the responses. Detailed architecture of the proposed BIST circuit for testing configured blocks is described in this section and interconnect testing issues will be discussed in next section. The main difference between our method and the FPGA BIST method suggested in [19] is that in our method the goal of testing is not to confirm the correct functionality of a BUT for all functions. Here, the goal is to make sure that each function (f_i) of an application configured into a block is working correctly. So, the test patterns should be applied for testing that function only.

In SCT method [16], instead of testing all resources of a reconfigurable device to locate all the defects, each block of the architecture can be tested for the specific function of a circuit, i.e. f_i, after f_i is configured into a block of the device. The applied test here just checks the correct functionality of the configured function (f_i), rather than diagnosing all defects of the block. So, there might be defects in molecular switches or nanowires of the block but as long as those defects do not cause any malfunctioning the function f_i is identified as fault-free. In other words, creating the function f_i on a block b_j requires just a subset of all nanowires and switches of that block. So, if the defective components of the block are not used during configuring f_i into that block, then the function can operate without fault. Therefore, the defects of the block are tolerated.

In the SCT procedure, the application is divided into m-input functions and each time one of these functions (f_i) should be configured into a block of the device and the input and output lines from the BIST circuit to f_i must be configured. Also, the same function f_i should be configured into the LUT of the BIST circuit (see Fig. 2). Next, the BIST circuit can simply apply exhaustive set of test patterns (2^m) to the function and test its functionality.

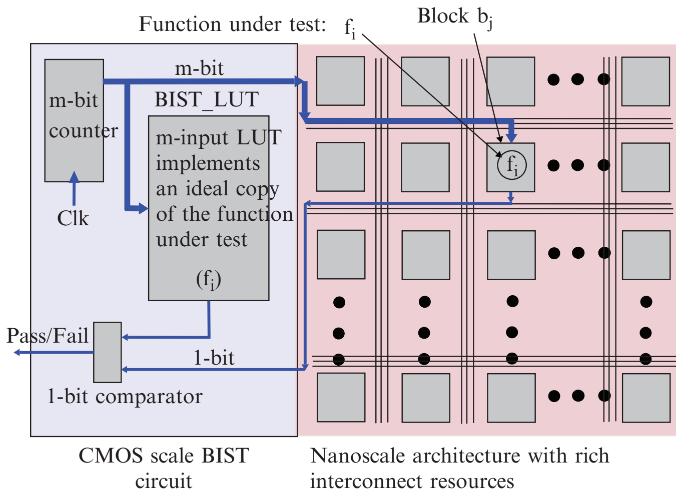


Fig. 2. The proposed CMOS BIST circuit used to test nano devices

If the implemented function passes the test, then it can be reliably used in the circuit. The process of selecting a function, mapping it onto a block of the device and creating interconnections between the configured block and BIST circuit and testing the function will be repeated for all functions of the application. If a function fails the test then we should configure another block with that function and the test process should be repeated.

Note that methods and tools for configuring nanoscale reconfigurable architectures will be similar to those used for FPGAs but some modifications may be required due to the architectural differences. This will be further discussed later in this chapter.

Figure 2 shows the proposed BIST scheme. The BIST circuit is composed of an m-bit counter, an m-input LUT and a comparator, resulting in low BIST area overhead. BIST circuit is assumed to be implemented in reliable CMOS scale.

Figure 3 shows the detailed SCT procedure. In this procedure, a function f_i of the desired application is selected (line #10) from the list of functions (F) and configured onto the LUT of the BIST circuit (BIST_LUT, line #10). If there are available blocks in the block list (B), one of them, e.g. b_j, will be selected and removed from the list of available blocks (line #13). Then function f_i will be mapped into this block (b_j) and the wires between this block and BIST circuit are configured (line #14). Test patterns are then applied to the block. If the test fails (line #15), the block will be sent to the set of defective blocks, i.e. DB (line #16). This will be repeated until either all functions are configured into the blocks of the device ($F = \{\}$) and the procedure is successfully finished or there are still some functions of the application left unmapped and the set of available blocks (B) becomes empty. In this case all the blocks of the device have been tried once, either they have been successfully used to implement one of the functions or they have shown defective behavior and were sent to the set of defective blocks (DB). If a block shows faulty behavior for one function, it does not mean that it will necessarily be faulty for every function because different functions will use different subsets of switches and nanowires in a block. Therefore, if there are some functions of the application left, defective blocks of the device can be tried (line #21). However, if a block is tried for a number of different functions and for all of them has shown a faulty behavior, i.e. $n_k \geqslant Discard_Threshold$ (where n_k is the number of times block b_k was used to implement a function and the result was faulty), then the block should be discarded and sent to discarded block list, i.e. DiB (line #26). After trying all the defective blocks of the device if still there are some functions of the application left in F, then the application is identified to be not implementable on the device.

As seen in the figure, two selections should be made during these steps. First, one of the functions of the application should be selected from the set of all functions, i.e. F (line #10). Then, an appropriate block of the device should be selected and configured with the selected function (line #13 or #21). Block selection is a decision that programming device should make

```
01: \\ Define:
02: \\ Set of available blocks: B={b_1, b_2, .., b_M};
03: \\ Set of functions of the application: F={f_1, f_2, .., f_T};
04: \\ Set of (defective block, # of defects):
05: \\ DB={(b_1,n_1), (b_2,n_2), .., (b_x,n_x)};
06: \\ Set of discarded blocks: DiB;
07: \\ # iterations a block should be detected as
08: \\ defective before it is discarded: Discard_Threshold;

09: while (F ≠ {}) {
10: Choose (f_i from F) ; F = F - {f_i} ;
    Config (BIST_LUT with f_i);
11: if (B ≠ {}) {
12:   while (B ≠ {}) {
13:       Choose (b_j from B); B = B - {b_j} ;
14:       Config (b_j with f_i) ; ConfigRoutes (b_j to BIST);
15:       if (BIST (f_i,b_j) = fail )
16:       DB = DB + {(b_j,1)} ;
17:       else Break ; }
18:       }
19: else if (B = {}) {
20:     while (DB ≠ {}) {
21:     Choose ((b_k,n_k) from DB) ;
22:     Config (b_k with f_i) ; ConfigRoutes (b_k to BIST);
23:     if ( BIST (f_i,b_k)=fail ) {
24:       increment (n_k) ;
25:       if (n_k ≥ Discard_Threshold)
26:       DB = DB - {(b_k,n_k)} ; DiB = DiB + b_k ;
27:       }
28:     else Break ;  }
29:       }
30: if (B = {} and DB = {}) {
31:     Application is not implementable on the device.}
32:   }
```

Fig. 3. The SCT procedure

based on the available blocks in the architecture, interconnections between the function being implemented (f_i) and other functions of the application and timing requirements of the circuit.

Low area overhead of this BIST structure provides the opportunity of parallel implementation of these testers on the chip so that at any time more than one function can be implemented and tested on the device as shown in Fig. 4. When multiple BIST circuits are implemented, test time will be reduced. In this case, more than one function and more than one block of the device should be selected at any time. Efficient methods based on heuristics can be used for these selections.

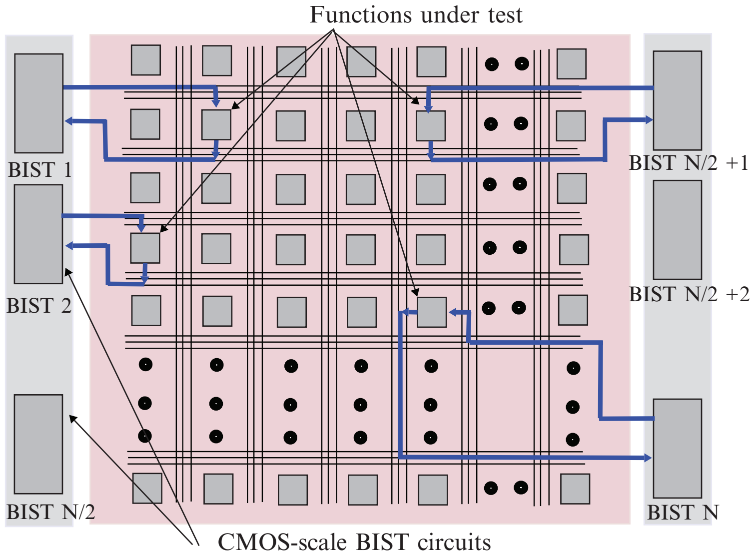


Fig. 4. Parallel use of multiple BIST circuits for testing nano devices

The interconnections between functions of the circuit also need to be configured and tested. This will be further discussed later in this chapter.

2.1 Analysis of the SCT Method

Since no nano-device is currently available to perform any real implementation, in this section, a probabilistic analysis is presented to show the timing requirements of the proposed process. We first calculate the probability of occurring a fault in a function f_i configured into a block of the device. Based on this probability we calculate the average number of blocks that must be configured to finally find a fault-free implementation for the function. Hence, we can calculate the required number of clock cycles for configuring and testing function f_i. Finally, for an application with T functions ($\{f_1, f_2, ..., f_T\}$ the average number of clock cycles required to configure and test all functions can be calculated. The results are compared with the number of cycles required to apply the BIST method presented in [14].

To keep the analysis simple, reliable CMOS scale interconnects are assumed for the architecture. We target an application that can be partitioned into T small functions each with m inputs, i.e. functions $\{f_1, f_2, ..., f_T\}$. As Fig. 5 shows, an m-input function can be implemented on a $(2m + 1) \times 2^m$ crossbar of nanowires configured as an m-input lookup table (LUT). Note that since diode logic cannot provide signal inversion, complement of the m input signals should either be applied to the block or created inside the block. As seen in the figure, switches on the junctions of the vertical nanowires with first

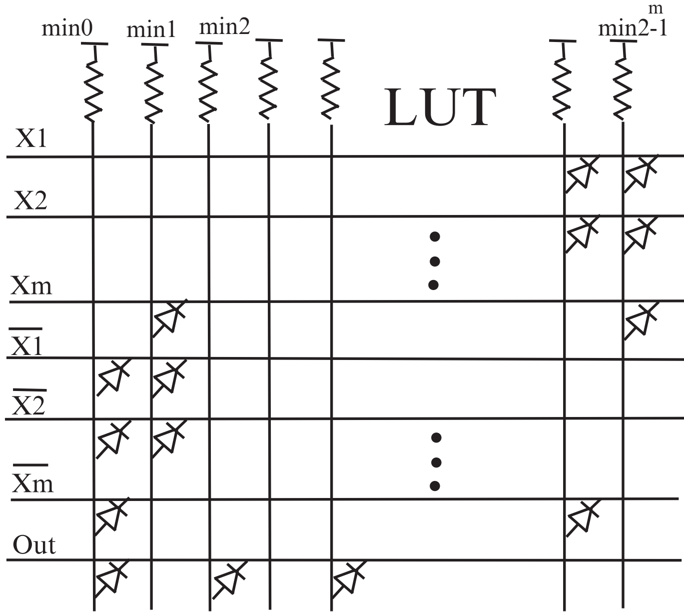


Fig. 5. Implementation of an m-input function on a $(2m+1) \times 2^m$ crossbar configured as a lookup table. Function $F(x_1, x_2, .., x_m) = \sum Minterms(0, 2, 4)$ is implemented as an example

$2m$ horizontal nanowires are configured to provide the minterms. The switches on the crosspoints of vertical nanowires with last horizontal nanowire (Out) are configured to provide the sum of the minterms specified by the function. So, the first $2m$ horizontal wires create the AND terms (minterms) and the last horizontal wire creates the OR term.

First, we calculate the average probability of having a fault in a function configured in a LUT. Let's assume that P_o is the probability of an open fault in a molecular switch, P_c shows the probability of a closed fault in a molecular switch and P_w denotes the probability of a fault caused by a defect in one of the nanowires.

The probability of an implemented minterm to be faulty is given by:

$$\begin{aligned} P_{(faulty\ minterm)} &= 1 - P_{(fault-free\ minterm)} \\ &= 1 - (1-p_c)^m (1-p_o)^{(m+1)} (1-p_w)^{(2m+2)} \end{aligned}$$

The probability of an m-input function with x minterms to be faulty is obtained by:

$$\begin{aligned} P_{(faulty\ function)} &= 1 - P_{(fault-free\ function)} \\ &= 1 - [(1-p_c)^m (1-p_o)^{(m+1)} (1-p_w)]^x (1-p_w)^{(2m+1)} \end{aligned}$$

The average probability of having a fault in a function (P_{ff}) and the probability of successful implementation (P_{si}) of a function on this structure after a number of repeated configurations (N_r) will be:

$$P_{ff} = \frac{\sum_{x=1}^{2^m} P_{(faulty\ function)}}{2^m}$$

$$P_{si} = (1 - P_{ff})P_{ff}^{(N_r-1)} \ \Rightarrow \ N_r = 1 + \frac{\log \frac{P_{si}}{1-P_{ff}}}{\log P_{ff}}$$

N_r can also be defined as the average number of blocks that should be configured to finally find a fault-free implementation of a function f_i on the device with a probability higher than P_{si}. The average number of switches to be configured for a function can be calculated as the average number of minterms in a function multiplied by the number of switches for each minterm. As seen in Fig. 5, there are $(m+1)$ switches for each minterm to be configured, so the average number of switches to be configured for a function (N_{sf}) is:

$$N_{sf} = \frac{m+1}{2^m} \sum_{x=1}^{2^m} x = \frac{(m+1)(2^m+1)}{2}$$

We assume that access and configuration structure of the architecture is capable of configuring N_{cs} switches in each cycle. It should also be noted that 2^m tests should be applied to each of the configured functions to test it exhaustively. Therefore, the average number of cycles required to configure and test an m-input function with a success probability higher than P_{si} would be:

$$\#\ Config\ \&\ Test\ Cycles\ (prob \geqslant P_{si}) = N_r \cdot 2^m + \frac{N_r \cdot N_{sf}}{N_{cs}}$$

First term in the above equation is the number of cycles required for testing the configured functions (when a function is configured, 2^m test patterns should be applied to it). Second term is the number of cycles required for configuring the function. For a circuit with T m-input functions, the time of performing SCT procedure assuming N parallel BIST circuits, as shown in Fig. 4, can be calculated as:

$$\#\ Cycles\ (SCT) = N_r \times (\frac{T}{N}) \times (2^m + \frac{N_{sf} \cdot N}{N_{cs}}) \qquad (1)$$

In this equation, first term of the sum is the cycles required to apply 2^m test patterns through N parallel BIST circuits to N configured functions and the second term of the sum is the cycles required to configure the functions into the blocks through configuration circuitry (N_{cs} switches in each cycle).

To compare this with the number of cycles required to test a nanoscale architecture using previously proposed test methods, we estimated the number of cycles for testing a circuit with the same specifications mentioned earlier

(T m-input functions) based on the BIST method proposed in [14]. The number of configurations required for testing a $K \times K$ nanoblock of a nanofabric is calculated to be $4K + 6$ and the number of configurations required for testing a $K \times K$ switchblock of a nanofabric is estimated to be $4K^2$ [14].

To make a fair comparison, here we use the same assumptions and parameters used in the analysis of our SCT method. We assumed using CMOS scale interconnects for the architecture hence, the required number of cycles for test and configuration of the interconnects were omitted from the calculations. Therefore, to keep the same conditions for [14], we assume the switchblocks of the architecture to be fault-free. So, we omit the $4K^2$ configurations required for testing each switchblock in the architecture. Based on our analysis, the number of cycles required for BIST method proposed in [14] based on these assumptions can be calculated as:

$$\#\ Cycles(BIST\ [14]) = \frac{T}{(1-p_w)^{2K}(1-p_c)^{K^2}(1-p_o)^{K^2}} \times \frac{4K+6}{N_{cs}} \quad (2)$$

where $K = \sqrt{(2m+1)2^m}$.

Once defect map is created and stored, it can be used to configure the functions of the design on the device. Therefore, additional cycles will be required to implement the functions based on the calculated defect map. Also, as discussed in [14], applying the test patterns using their BIST process (and almost all other proposed methods) will not result in finding the exact location of the faulty block. Hence, there will be some fault-free blocks that are identified as faulty during these test methods. That means recovery, i.e. the ratio of detected fault-free blocks to all fault-free blocks, in these test methods is lower than 100%. To achieve high recovery it is suggested that a recovery procedure should be applied to exactly locate the faulty blocks. This diagnostic procedure will require considerable number of reconfigurations that will significantly increase the test time. In analysis presented above we assumed that the test process can exactly locate the faulty blocks (in other words we assumed 100% recovery). This assumption makes (2) to show test cycles lower than the actual number of cycles required by the BIST method presented in [14].

The number of cycles for SCT procedure and for BIST process proposed in [14] are compared for different defect probabilities and design parameters and the results are presented in Figs. 6 and 7. In each of these figures, constant values are assigned to m, T, N_{cs} and P_{si}. Different values are assigned to N, i.e. the number of parallel BIST circuits, and to the fault probabilities of the molecular switches and nanowires. As the results demonstrate, in most cases the required cycles for our SCT procedure is considerably less than cycles of BIST method of [14].

As Fig. 8 shows, increasing N, i.e. the number of concurrent BIST circuits, will result in decreasing the overall time of the SCT procedure and increasing the BIST area overhead. However, increasing N will not decrease overall SCT procedure time linearly. If the configuration circuitry is capable of configuring

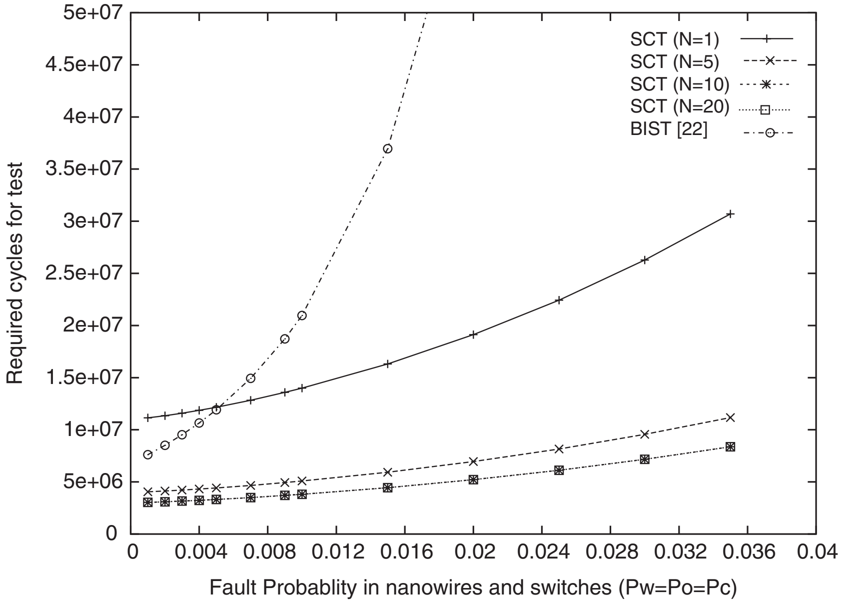


Fig. 6. Number of cycles required for testing a nano device using SCT method and BIST method proposed in [14] when $m = 3, T = 1,000,000, P_{si} = 0.999, N_{cs} = 5$

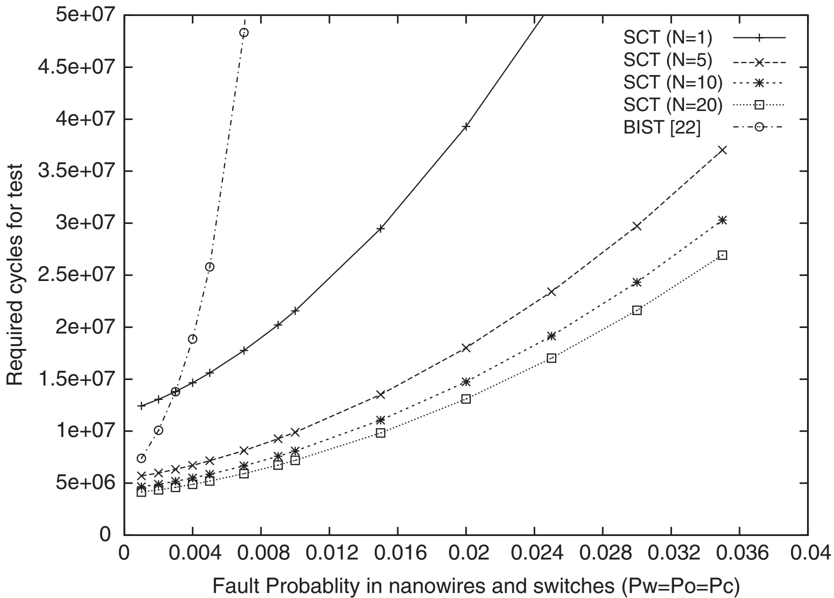


Fig. 7. Number of cycles required for testing a nano device using SCT method and BIST method proposed in [14] when $m = 4, T = 500,000, P_{si} = 0.999, N_{cs} = 5$

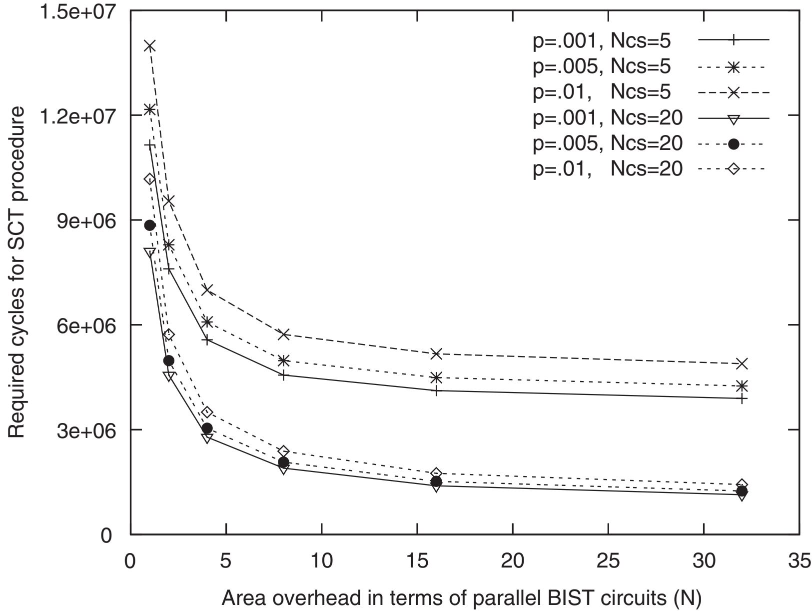


Fig. 8. SCT procedure time vs. N, i.e. the number of parallel BIST circuits, which is a measure of BIST area overhead $(m = 3, T = 1,000,000, P_{si} = 0.999)$

a constant number of switches in each cycle (N_{cs}), then as N increases configuration time becomes the dominant part of the SCT procedure and this part cannot be reduced through adding the number of parallel BIST circuits (N). Adding more parallel configuration circuitry, i.e. increasing N_{cs} from 5 to 20 as shown in the figure, will reduce the configuration time and this in turn reduces the SCT time. Area overhead resulted by adding the parallel BIST circuits will be still small comparing to the area required to store the defect map as previously proposed by BIST methods such as [11, 13, 14].

3 PNR-Aware Defect Avoidance

One of the disadvantages of the proposed SCT procedure is that it still does not take place and route (PNR) tool information into account. In other words, it may result in different performance every time a nano-device is configured. To alleviate the problem, in this section we propose a new method that uses SCT as the basic engine but it considers the placement and routing provided by the tool to avoid per chip PNR. The proposed method takes advantage of the existing redundancy in the nano-LUTs or nano-PLAs to configure the target function. It still does not require creation of defect map.

3.1 Redundancy in Crossbar-Based PLAs

Figure 9 shows two implementations of function $f = ab + \bar{b}c$ on a defective crossbar-based PLA. The PLA, shown in the figure, is a combination of an AND plane and an OR plane and can be implemented based on diode logic by using a crossbar of nanowires and configurable molecular switches on the crosspoints of the wires [10]. The PLA may also be implemented using switching properties of FETs created on carbon nanotubes. As seen in the figure, even in the presence of a number of defects in the crossbar, there are several choices for fault-free implementation of function f. This is due to the fact that there is high amount of inherent redundancy in this crossbar and large number of resources (switches and wires) available for implementing a function.

In general, if we consider the case of configuring a function of size $(K_f, M_f, 1)$, i.e. K_f inputs, M_f product terms and one output, on a PLA of size (K, M, N), i.e. K input lines, M product lines and N output lines as shown in Fig. 9, the number of possible ways to implement a function on PLA can be calculated as:

$$P_{M_f}^{M} \times P_{K_f}^{K} \times P_{1}^{N}$$

where $P_{M_f}^{M}$ represents all possible permutations of M_f product terms among M product lines. ($M_f \leqslant M$ and $K_f \leqslant K$). This relation holds true for a defect-free crossbar that all its resources are utilizable for implementing a

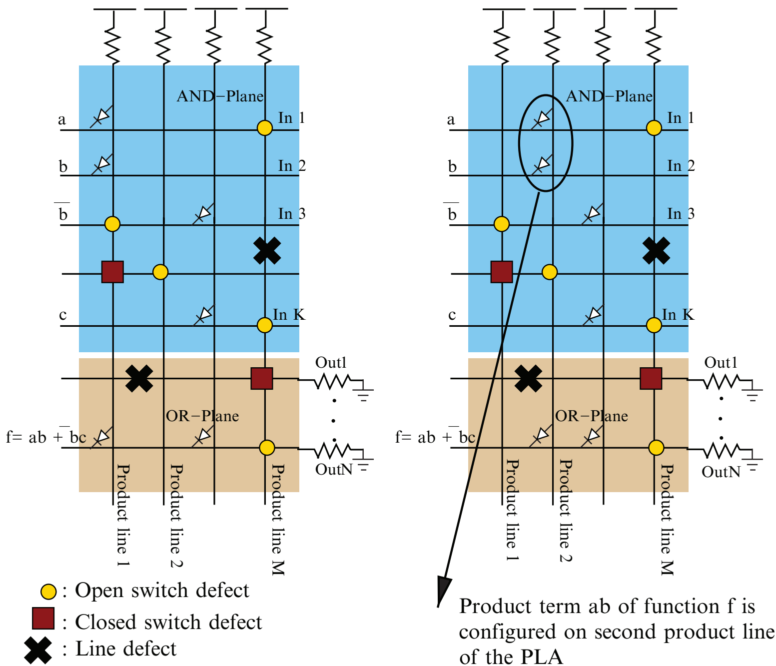


Fig. 9. Two different implementations of function $f = ab + \bar{b}c$ on a defective crossbar

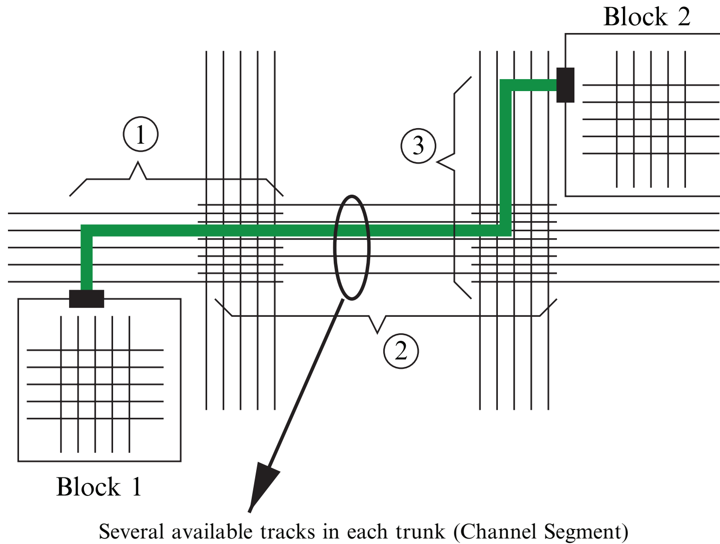


Fig. 10. There are several tracks in each routing channel that can be used in the interconnect between two blocks. They can be carefully selected to avoid the defects

function. For example, for implementing a function of size (5,6,1) on a PLA of size (7,8,3), there are 7065 possible configurations. Efficient algorithms must be devised to find a fault-free mapping of the function onto the PLA.

Redundancy is also available in interconnects formed using crossbars of nanowires. As Fig. 10 shows, there is redundancy in interconnect resources used between two blocks of a device. Therefore, one can find several paths from one block to another without changing the structure of the path between them. Hence timing characteristics of this path will not change. In other words, in each segment of the route between two blocks (segments 1, 2 and 3 in the figure) there are several tracks that can be used. The amount of redundancy in routing resources of the device is a design parameter (as well as the block size) that should be precisely determined for architecture of the device. In FPGA design, an optimum value for the number of available tracks in each routing channel should be considered so that an acceptable trade-off between area and performance can be reached. In nanoscale reconfigurable devices, the number of tracks in each routing channel should be selected carefully such that routing resources can provide the required defect tolerance while maintaining the area in an acceptable range.

3.2 Configuration and Test Procedure

Top level flow of the proposed test and configuration procedure is depicted in Fig. 11. The first phase of the method is the conventional placement and

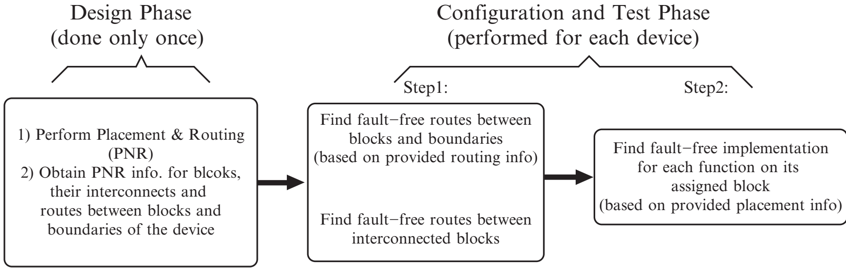


Fig. 11. The proposed test and configuration procedure considering placement and routing information

routing which is performed only once during the design phase. Placement and routing information of the blocks will be used in the next steps of the procedure. In this phase, in addition to placement and routing of blocks, connections between the blocks and boundary of device are also determined. Routes between blocks and boundaries of the device will be temporarily configured during test phase for each block and used by tester and programming device to apply tests and configuration voltages to the block.

Test and configuration procedure starts after placement and routing phase. For every block, routes between the block and boundaries are used to access the block (Step 1). The block is configured as pass-through and a small test and selection phase is performed to find a defect-free route from the block to boundaries. After finding the defect-free routes between the boundaries and blocks, these routes can be used to find defect-free routes between the blocks in a similar approach. Performing the proposed method for interconnections requires detailed information about the routing architecture of the device. However, since both routing resources and blocks of these devices are based on crossbars, it can be expected that similar defect tolerance capabilities exist in routing section of these devices and performing the proposed method for interconnects will require similar test and configuration efforts and will result in similar defect tolerance and yield values. Simulation programs discussed in this chapter focus on Step 2 of Fig. 11, i.e. searching for a fault-free implementation of the functions on the blocks of the device.

Note that the SCT procedure is used to test all blocks and routes. Once fault-free routes are found, the procedure goes into the step of configuring and testing each of the functions on the block specified in placement phase (Step 2). This can be performed through various approaches. The following subsections will present two of our proposed methods.

3.3 Identifying Fault-Free Configuration of Functions

Figure 12 shows a high level pseudo code for the identifying fault-free implementation of each function of the target application. It shows that either of

```
01 \\ Define:
02 \\ Set of crossbar blocks of the device: B={b1, b2, .., bM} ;
03 \\ Set of functions of the application: F={f1, f2, .., fT} ;
04
05 Main()
06 {
07 Foreach (fi ∈ F) {
08   \\bj is a block of the device that is assigned to fi through PNR
09   Config(interconnects from bj to device boundaries);
10   Exhaustive_Search(fi,bj); OR Greedy_Search(fi,bj);
11   if(failed)
12     {Discard the device (this is considered as yield loss), exit;}
13 } End Foreach;
14 } End Main;
15
```

Fig. 12. The proposed procedure for identifying fault-free implementation of functions

```
01 Exhaustive_Search(f,b)
02 {
03 For(i: 1 to N) {
04 \\ Select one output line out of N
05   Select(output i of b);
06   Foreach(Permutation(Kf,K)) {
07   \\ Select Kf input lines out of K
08     Foreach(Permutation(Mf,M)) {
09     \\ Select Mf product lines out of M
10     Configure_Switches();
11     Apply_Test();
12     if(pass) Return;
13     } End For;
14   } End For;
15 } End For;
16 }
```

Fig. 13. Using exhaustive method for identifying fault-free implementation of each function on its assigned crossbar block

the two algorithms (Greedy or Exhaustive) can be used (Figs. 13 and 14). An exhaustive search method can be implemented to examine all possible configurations of the functions on their blocks. In each case, once the function is configured, test patterns should be applied through selected paths from boundaries of device to the block. If the test results for the function shows a fault-free behavior then the desired configuration is found. This, search should be repeated for all functions of the application. Exhaustive searching of all possible configurations of functions on crossbar blocks cannot be practical due

```
01 Greedy_Search(f, b)
02 {
03 \\f is a function with K_f inputs, M_f product terms and one output
04 \\b is a crossbar block with K inputs, M product lines and N outputs
05 \\First step is selecting K_f input and one output for f
06 For(i: 1 to N) {
07   Select(output i of b);
08   For(j: 1 to K) {
09     Select(input j of b);
10     Configure_Switches(input j and output i);
11     Apply_Tests(input j , output i);
12     if(test pass and number of selected inputs more than K_f)
13     Break;
14   } End For;
15 } End For;
16 \\Second step is to find M_f product lines of b
17 For(i: 1 to M_f) {
18   For(j: 1 to M){
19   if(product line j of b is not already assigned to a product term of f)
20   {
21   Select (product line j of b);
22   Configure_Switches(product term i of f into product line j of b);
23   Apply_Tests(from K_f selected inputs of b for the configured product term);
24   if(pass) {
25     Assign(product term i of f to product line j of b);
26     Break; }
27   } End For;
28 } End For;
29 }
```

Fig. 14. Using greedy algorithm for identifying fault-free implementation of each function on its assigned crossbar block

to high amount of time required for performing this search on each device. However, it can provide the upper limit of defects that can be tolerated in crossbar devices. Function *Exhaustive_Search*() in Fig. 16 is a pseudo code for this exhaustive approach.

Having the access routes from device boundaries to each of the blocks, implementation of each function of the application on its assigned block can be performed by a greedy search algorithm. Function *Greedy_Search*() in Fig. 14 shows the proposed approach. First, a set of simple test patterns are applied to find K_f fault-free lines among K inputs of the PLA and one fault-free line from its N outputs (lines 6–15 in Fig. 14). Then, each product term of the function is configured on one of the available product lines of PLA and test vectors are applied to test it. If the product line shows faulty behavior, the process configures the product term on the next available product line and

tests it again. This will be repeated until all product terms of the function are configured and tested on appropriate product lines of the block (lines 17–28 in Fig. 12).

3.4 Probabilistic Analysis

To evaluate the feasibility of the proposed method, a probabilistic analysis is performed. We assume that the crossbar-based PLA has M product terms, K inputs and N outputs (see Fig. 9). Also assume that the function that is intended to be implemented on PLA has M_f product terms ($M_f < M$), K_f inputs ($K_f < K$) and one output. Also assume that p_l, p_o and p_c are the probabilities of a defect in lines (vertical/horizontal) of the crossbar, stuck-open defect on each molecular switch and stuck-closed defect on a molecular switch, respectively. Probability of finding K_f defect-free input lines (P_{in}) among all K inputs of the PLA will be:

$$P_{in} \geqslant 1 - \sum_{i=K-K_f+1}^{K} (p_l)^i$$

Probability of finding one defect-free output line (P_{out}) among N outputs of the PLA is given by:

$$P_{out} = 1 - (p_l)^N$$

Once K_f defect-free inputs and one defect-free output is found for implementing the function, M_f defect-free product terms should be found among M product lines of the PLA. To have a product line of the PLA usable as a product term of the function, the product line itself should be defect-free and the switches on the crosspoints of the line with K_f input lines and one output line should also be defect-free. In other words, when a switch needs to be configured as open in the function, then it should not have a stuck-closed defect on the product line of PLA and if it must be configured as closed in the function, then it should not have stuck-open defect in the PLA. Therefore, probability of being defect-free for a product line (or product term) of the PLA (P_{pterm}) is computed by:

$$P_{pterm} = (1 - p_l) \times (1 - \frac{p_o + p_c}{2})^{K_f+1}$$

At least M_f of these defect-free product lines are required to be able to implement the function on the PLA. Probability of having these defect-free product lines ($P_{at\ least\ M_f\ P\ lines}$) is given by:

$$P_{at\ least\ M_f\ P\ lines} \geqslant 1 - \sum_{i=M-M_f+1}^{M} (1 - P_{pterm}^i)$$

Finally, probability of successful implementation of the function on the PLA ($P_{success}$) can be calculated as:

$$P_{success} \geqslant P_{in} \times P_{out} \times P_{at\ least\ M_f\ P\ lines}$$

A_r denotes the ratio of the area of a PLA with M product lines, K inputs and N outputs to the minimum area required for implementation of a function with M_f product terms, K_f inputs and one output on the PLA. Ar is computed by:

$$A_r = \frac{M \times (K + N)}{M_f \times (K_f + 1)}$$

Figure 15 shows the probability of successful implementation of a function on a PLA for different defect densities and different amounts of redundancy provided in the PLA. As seen in the figure, having higher amount of redundancy in the PLA results in very high probability of successful implementation ($P_{success}$) of functions. It should be noted that for the results shown in Fig. 15 we have set $p_c = p_l = p_o$. Therefore, the probabilities shown in horizontal axis in Fig. 15 are the total probability of stuck-open, stuck-closed and line defects. In fact, this is a pessimistic assumption because the probability of occurring stuck-closed and line defects are expected to be much less that that of stuck-open defects [12]. Therefore, we can expect higher $P_{success}$ in implementing functions on crossbar PLA.

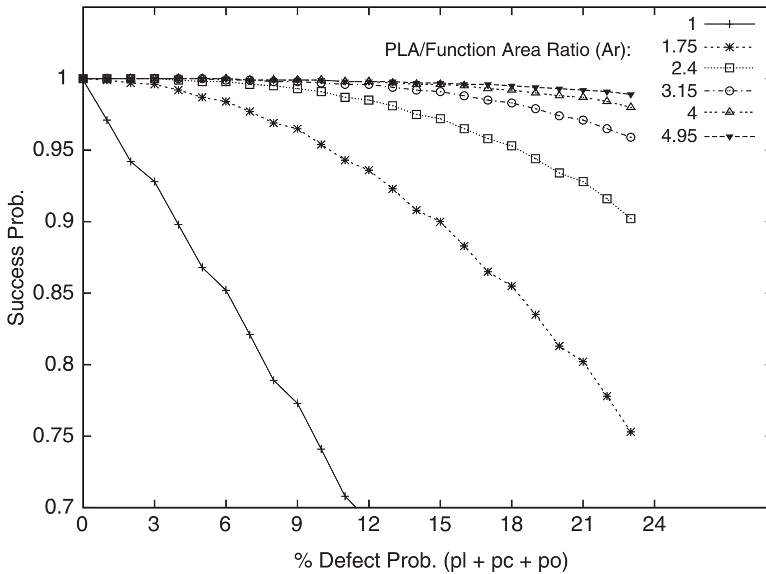


Fig. 15. Probability of successful implementation ($P_{success}$) of a function on crossbar PLA for different defect probabilities and different amounts of redundancy

3.5 Two Different Implementations of the Proposed PNR-Aware Defect Avoidance Method

In order to evaluate the proposed method in terms of the ability to find defect-free implementation of target application's functions on PLAs and at the same time testing the configured functions, we have developed two simulation programs for implementing the proposed method on MCNC benchmarks. They implement the second step of the block diagram shown in Fig. 11, i.e. Step 2 of the second phase which is identifying a fault-free implementation of functions on configurable crossbar blocks. Both implementations use outputs of SIS (Flowmap and Flowpack). SIS is a synthesis package that can be used to map the benchmarks into partitions (here functions, f_i) of a specified size. Each of these partitions are implementable on a PLA. In our simulations, SIS provides the required partitioning of benchmarks into small functions of size K_f.

Implementation 1: Exhaustive Method

The first method performs an exhaustive search on a crossbar to implement the logic functions. The crossbar is modeled as a two dimensional array. Each array element specifies the status of one of the crosspoints of the vertical and horizontal nanowires in the crossbar. Defects are randomly generated in this array. As discussed in the previous section, open diode defects, closed diode defects and nanowire defects are considered with probabilities of P_o, P_c and P_l, respectively. The logic functions are extracted from MCNC benchmarks through a Perl script. In this experiment the search program is called a hundred times for each function and average rate of success in implementing the function on defective crossbar is measured. The results of these experiments show the success rate in implementing functions on defective crossbars under different defect rates through *an almost exhaustive search.* Figure 16 shows various steps required for our exhaustive search.

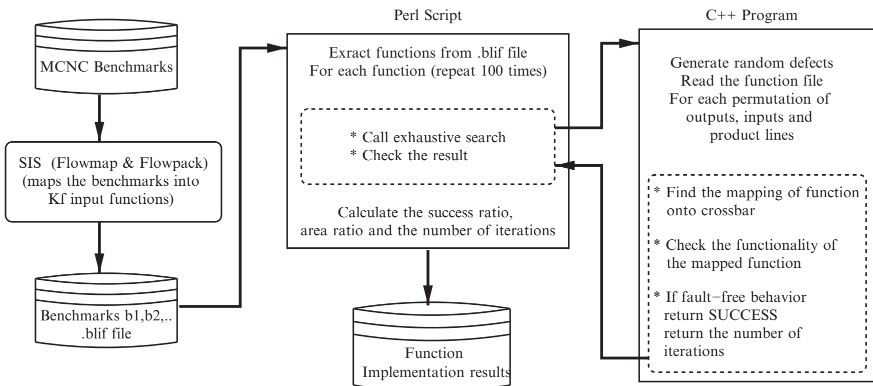


Fig. 16. Exhaustive method for mapping functions onto defective crossbars

Implementation 2: Greedy Method

The second implementation consists of three parts: the SIS package, a perl script and a PLA Verilog model. Figure 17 shows a high level flow of the developed simulation program. The Perl part of simulation program reads outputs of SIS and extracts each of the functions of the benchmark, i.e. the product terms of each function is obtained. Then, the program searches different possible configurations for each function on the PLA. This is performed using the greedy algorithm shown in Fig. 14. For each of these configurations, input and configuration files are generated and applied to the Verilog model of PLA and Verilog simulator (Synopsys VCS) is then called to simulate that configuration. This will be repeated for all product terms of each function of the benchmark. For implementing each function on the PLA model, a defect file should be created for the model. This file is created by random assignment of defects to different PLA components. Probabilities of line defects, stuck-open and stuck-closed switches are the inputs to the program and they can be tuned to simulate different defect densities. The generated defect file will also be applied to Verilog model of PLA and used in all steps of implementing a function. For each function a new random defect file should be created. This is because each crossbar PLA block has its own set of defects.

We have written behavioral Verilog models for each component in a PLA including *product line*, *input line, output line and molecular switches between input/output and product lines*. For each of these components appropriate defect types are considered and behavior of the component is described under each of these defects. Finally, all these components are put together in an array to create the PLA module. The PLA module is capable of reading three different files: configuration information file, defect information file and the

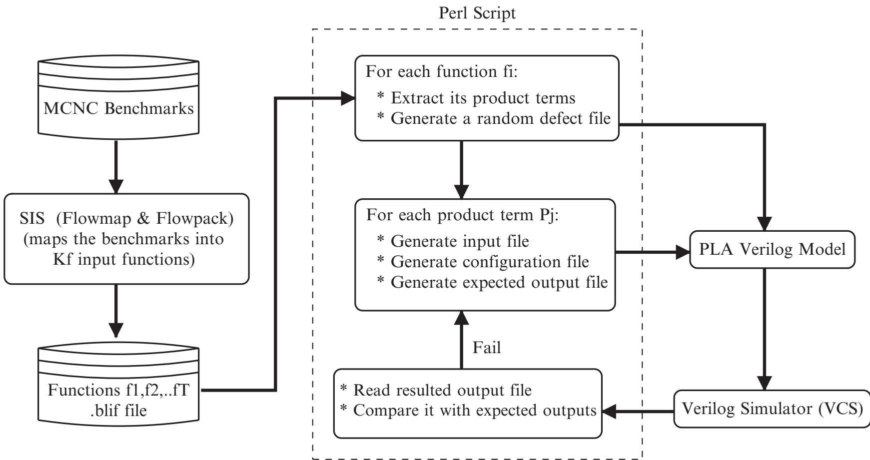


Fig. 17. Greedy method for mapping functions to defective crossbars

input file. Once these three files are prepared for the PLA, Verilog simulator will be called to obtain the outputs which will be stored in an output file.

In order to find a defect-free implementation of function f_i on the PLA, the simulation program looks for appropriate positions for each product term (pt_j^i) of function f_i in the PLA. The program maps the product term pt_j^i into a product line of the crossbar. Then, Verilog simulator is called and the simulation output file is analyzed. If the Verilog simulation result is correct for the specific product term, then the process will be repeated for another product term of the function. Otherwise, a new product line of the PLA should be selected for implementation of the product term of function f_i and new configuration and input files will be created and the Verilog simulation will be repeated.

Once all product terms of the function f_i are placed on product lines of the PLA model and their functionality is verified the task of implementing function on the PLA model is accomplished. This task should be performed for all functions of the benchmark. If the program cannot find defect-free implementation of function f_i on a PLA, then the chip will be discarded. This is considered as yield loss because no defect-free mapping of the benchmark on the device was found.

For our simulations, size of the functions in the benchmarks, i.e. number of their inputs (K_f), is specified through the mapping phase in the SIS package. Size of the PLA model can be specified as a parameter to the Verilog code. So, the amount of redundancy available in the PLA model can be calculated. Also defect densities can be specified as parameters in the program. The overall yield resulted from the process of testing, configuring and searching for defect-free implementation of benchmarks is calculated and reported by the simulation program.

In this section we present the results obtained from running the simulation programs on 10 MCNC benchmarks ($i1 - i10$). As discussed in the previous section, the SIS package was used to partition the MCNC benchmarks into K_f input functions implementable on PLAs. The .blif output file obtained from SIS are used in our simulation program. In all the experiments, defects are randomly injected in the PLA models. In our experiments, we have assumed a uniform defect distribution and made the experiments under different defect probabilities.

3.6 Results Obtained from Exhaustive Method

We have performed the exhaustive search simulation for very high defect rates. In these experiments stuck-open defects have probability values of 10 and 15% and stuck-closed defects occur with probability 10, 12, and 15%. Also probability of nanowire defects is considered to be 5 and 10%. These defect conditions are clearly beyond the expected rate of defects for technology of nanowire crossbars. However, by assuming such extremely high defect rates we can evaluate the upper limits of defect tolerance on a crossbar if an exhaustive

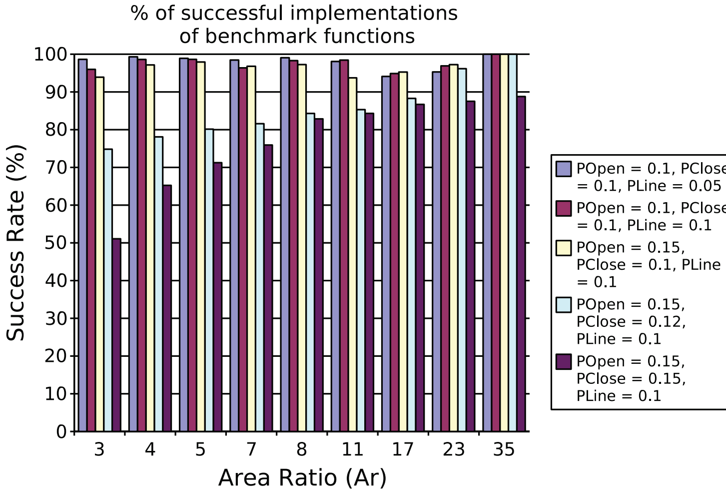


Fig. 18. Success ratio in implementing logic functions of the benchmarks on defective crossbar PLAs using exhaustive method

search is employed to find fault-free implementation of functions. The number of inputs of the functions (K_f) is set to be less than four through SIS. For each of these functions the search program is executed exhaustively and the number of successful mappings of the function onto defective crossbars is obtained. Also the average number of iterations the exhaustive search program required to find a fault-free implementation of the function is obtained. In these experiments we have considered a crossbar of size $M = 7$, $K = 7$ and $N = 3$. Since the functions of benchmarks are bounded to be less than four inputs then there will be different area ratios based on the function size (number of its inputs and product terms) and size of the crossbar.

Figure 18 shows the average success ratio in implementing functions into the crossbar under different defect rates. As figure depicts, very high defect rates can be tolerated in crossbars given enough amount of time to search the crossbar for a fault-free implementation. Although exhaustive search cannot be considered a practical approach, the observed tolerance against defects encourages developing fast search algorithms for fault-free implementation of the functions on crossbars.

Figure 19 shows the average number of iterations required for finding fault-free implementation of functions on crossbars. As seen in the figure, increasing A_r (increasing redundancy) will reduce the number of required iterations.

3.7 Results Obtained from Greedy Method

In these experiments we have assumed that stuck-open probability of molecular switches changes from 0 to 10% and stuck-closed probability of molecular switches and line defect probabilities changes from 0 to 6%. Therefore, in

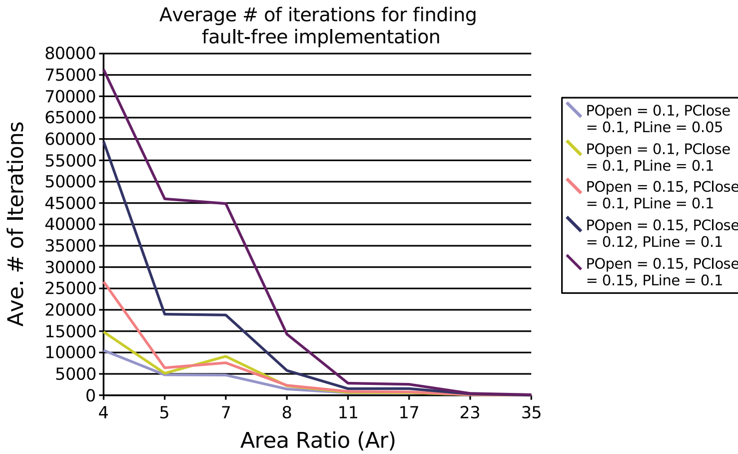


Fig. 19. Average number of iterations for implementing functions of the benchmarks on defective crossbar PLAs using exhaustive method

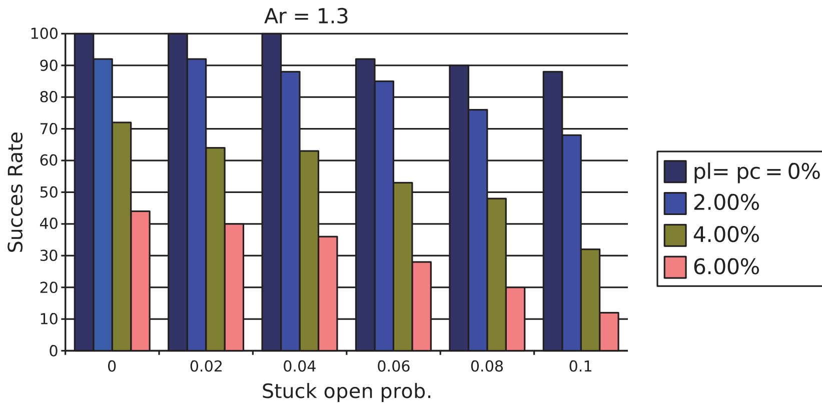


Fig. 20. Success rates for implementing the benchmarks on defective crossbar PLAs when Ar = 1.3

worst-case condition, there will be 10% stuck-open plus 6% stuck-closed and 6% nanowire (line) defect in the PLA model that is still beyond the expected defect rates for nanowire crossbars.

Another parameter that must be taken into consideration in these experiments is the amount of redundancy provided in crossbar PLAs. This redundancy can be measured as the area ratio (Ar) between the crossbar PLA of size $M \times K \times N$ and the functions of size $M_f \times K_f \times 1$. The first set of results shown in Figs. 20 and 21 are the average values of success rate obtained from running the simulation program on the 10 MCNC benchmarks. The figures show the obtained success rates for two different amounts of redundancy and for different values of defect probability.

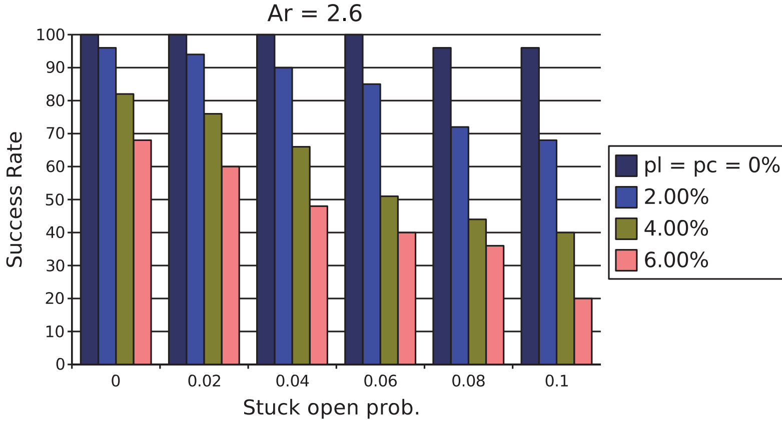


Fig. 21. Success rates for implementing the benchmarks on defective crossbar PLAs when Ar = 2.6

Here, we have defined success rate as ratio of the number of times that implementation of functions on the crossbar PLA is successful to the total number of functions in the benchmarks.

$$Success\ \ Rate = \frac{no.\ of\ functions\ successfully\ implemented\ on\ the\ crossbar\ PLA}{Total\ no.\ of\ functions\ in\ the\ benchmarks}$$

This definition of success rate can provide an understanding of how much success the method has had in providing defect tolerance in a nanoscale device. This is the same definition used in the exhaustive search experiments implemented for functions of benchmarks as described in the previous subsection. Although the results are only for the device blocks and not routing, yet high success rate can be achieved. For example, as seen in Fig. 21, when $po = 6\%$ and $pl = pc = 2\%$, 80% success rate can be achieved. As reported in literature, in fabrication methods of nanoscale crossbars, defect probabilities of up to 10% for stuck-open faults are expected and considerably lower probabilities are predicted for stuck-closed and line defects [7]. As seen in Figs. 20 and 21, success rate considerably decreases for higher values of stuck-closed and line defects. Stuck-closed and line defect will cause parts of the crossbar (a defective line or two closed lines) to function incorrectly. Therefore, their adverse effect on achievable success rate is more considerable than stuck-open defects.

Figure 22 provides a comparison between achieved success rates in four different redundancy conditions. These results are shown for 2% probability of stuck-closed and line defects ($pl = pc = 2\%$) and different values of stuck-open probability (po). As seen in Fig. 22, relatively high success rate is achievable through the proposed test and configuration method.

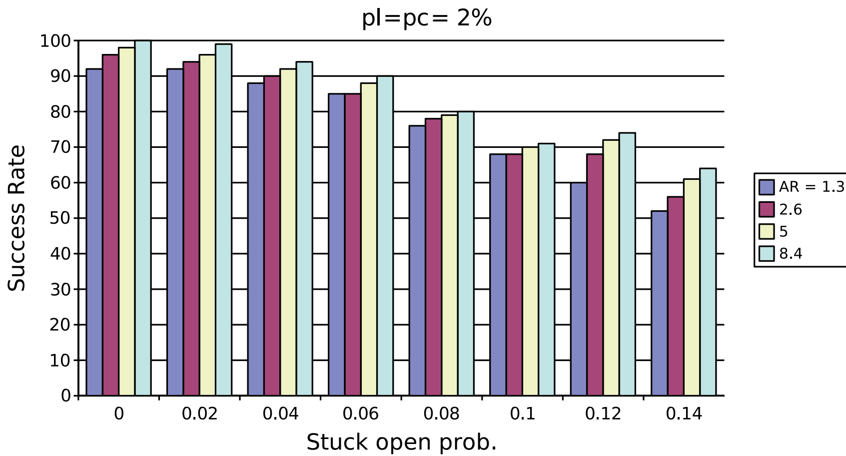


Fig. 22. Success rate for implementing the benchmarks on defective crossbar PLAs under different values of redundancy when $pl = pc = 2\%$

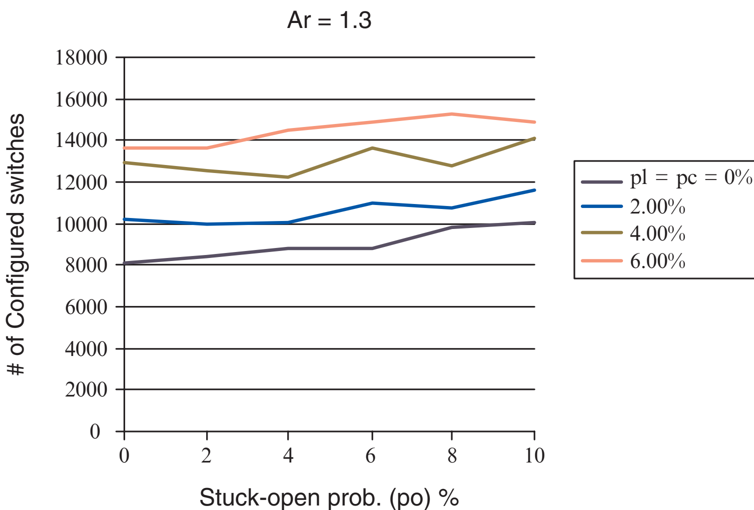


Fig. 23. Average number of configured switches for implementing the benchmarks when Ar = 1.3

Results shown in Figs. 23 and 24 represent the estimated average number of required molecular switch configurations for finding defect-free implementations of each benchmark on the defective crossbar PLAs. The time required for these configurations is a major part of the total test and configuration time. This time depends on the bandwidth of the programming device, i.e. the number of molecular switches that can be programmed in parallel. It will also be proportional to the total number of switches to be configured.

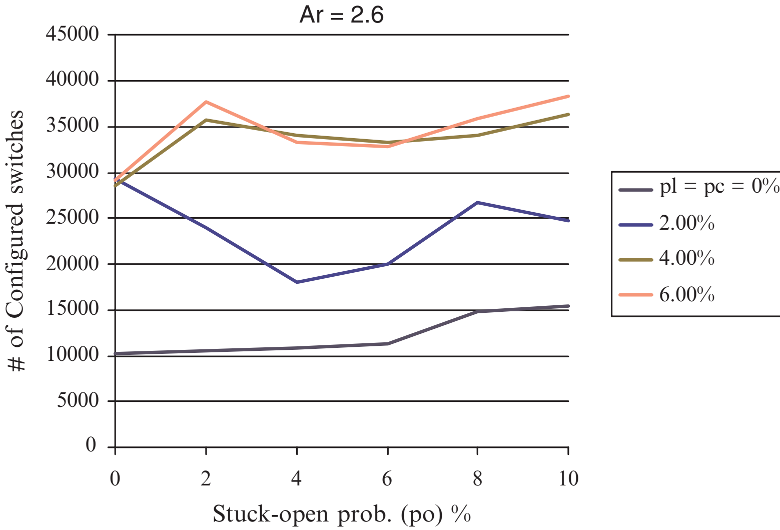


Fig. 24. Average number of configured switches for implementing benchmarks when Ar = 2.6

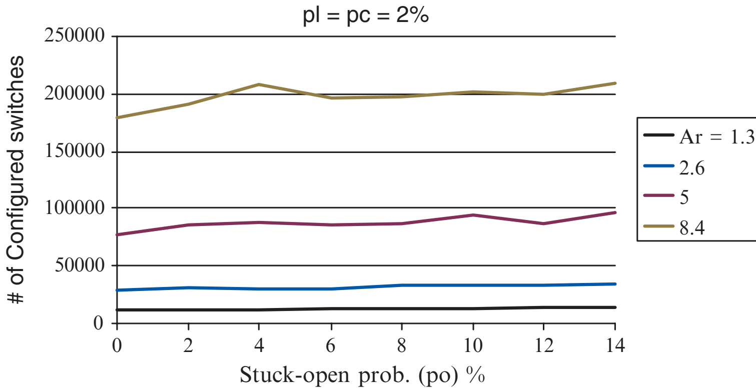


Fig. 25. Average number of configured switches for implementing the benchmarks under different values of redundancy ($pl = pc = 2\%$)

Therefore, we have reported the number of switch configurations as a parameter directly affecting the process time. As seen in the figures, higher probabilities of stuck-open faults will increase switch configurations gradually. However, higher values of stuck-closed and line defects will increase the number of required switch configurations more considerably.

Figure 25 shows the average number of switch configurations for different redundancy values. As redundancy increases, crossbar size and hence number of its molecular switches increases. Therefore, number of switch configurations and overall process time increases as well.

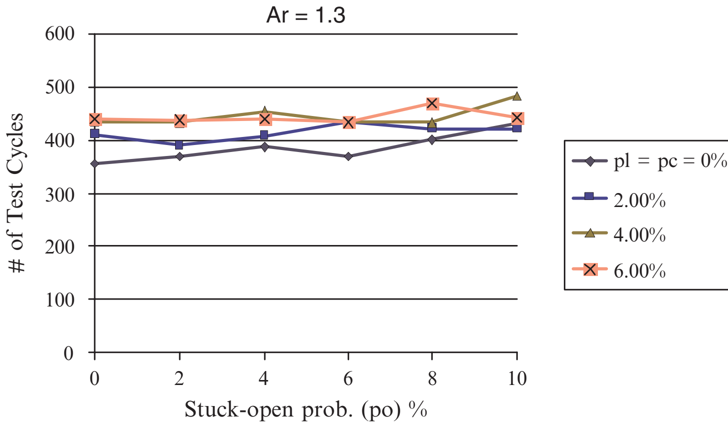


Fig. 26. Average number of test cycles for implementing the benchmarks when Ar = 1.3

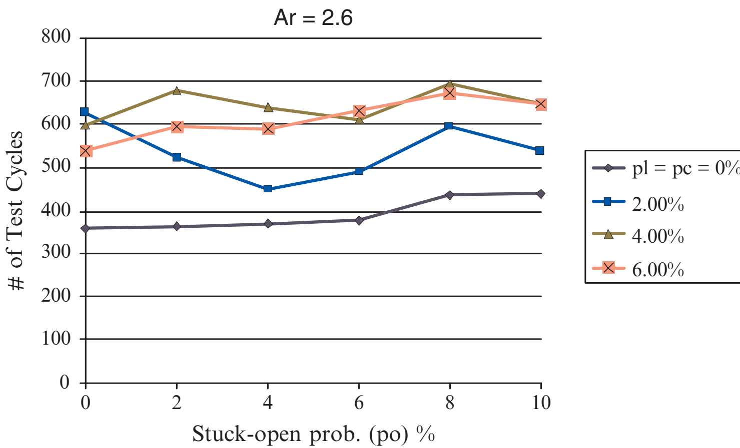


Fig. 27. Average number of test cycles for implementing the benchmarks when Ar = 2.6

Figures 26 and 27 show estimated average number of required test cycles for finding defect-free implementations of benchmarks. Number of test cycles is another parameter that directly affects total time of the test and configuration method. As seen in the figures, for higher defect probabilities, there is a slight increase in the number of required test cycles. This, along with results shown in Figs. 23 and 24, suggests that the total time of the method does not have dramatic changes in different defect probabilities and the method can be used even for very high defect rates.

Figure 28 shows the average number of test cycles for different values of redundancy. Increasing the size of crossbars used in the device, i.e. increasing redundancy, results in higher number of test cycles because there will be more product lines in each crossbar PLA that must be searched for finding

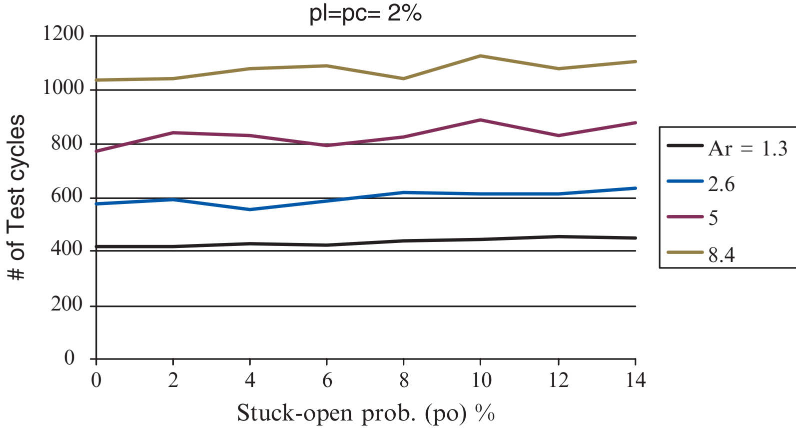


Fig. 28. Average number of test cycles for implementing the benchmarks under different values of redundancy when $pl = pc = 2\%$

defect-free implementation of functions into it. Therefore, higher redundancy results in higher number of test cycles and higher number of switch configurations and as a result, run time of the method will increase.

4 Conclusions

In this chapter an efficient test and defect tolerance method is proposed for crossbar-based nanoscale reconfigurable devices. An important advantage of the proposed method compared to previously proposed methods is that it does not require defect information of the device to be stored in a defect map. Also based on this method, placement and routing task will be performed only once in design phase. The proposed method is based on searching for possible implementations of a logic function in a crossbar PLA and finding its fault-free implementation. A probabilistic analysis was presented to demonstrate the high possibility of finding defect-free implementations of logic functions in defective crossbars. Two different implementations are presented and several experiments were performed on MCNC benchmarks under different defect probabilities and redundancy. The results obtained from the two methods show that very high defect rates can be tolerated.

References

1. ITRS 2005, *International Technology Roadmap for Semiconductors, Emerging Research Devices*, http://public.itrs.net/Common/2005ITRS/
2. A. DeHon, S. C. Goldstein, P.J. Kuekes and P. Lincoln, “Nonphotolithographic Nanoscale Memory Density Prospects,” *IEEE Transactions on Nanotechnology*, vol. 4, no. 2, pp. 215–228, March 2005

3. A. DeHon and M. J. Wilson, "Nanowire-Based Sublithographic Programmable Logic Arrays" *Int. Symp. on Field Programmable Gate Arrays (FPGA'04)*, pp. 123–132, 2004
4. R. M. P. Rad and M. Tehranipoor, "A New Hybrid FPGA With Nanoscale Clusters and CMOS Routing," submitted to *Design Automation Conf. (DAC'06)*, pp. 727–730, 2006
5. A. DeHon, "Nanowire-Based Programmable Architectures," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 1, no. 2, pp. 109–162, July 2005
6. D. Bhaduri and S. Shukla, "NANOLAB – A Tool for Evaluating Reliability of Defect-Tolerant Nano-Architectures," *IEEE Transactions on Nanotechnology*, vol. 4, no. 4, pp. 381–394, July 2005
7. P. Kuekes, W. Robinett, G. Seroussi and R. S. Williams, "Defect-Tolerant Interconnect to Nanoelectronic Circuits: Internally Redundant Demultiplexers Based on Error-Correcting Codes," *Inst. Phys. Nanotechnology*, issue 16, pp. 869–882, 2005
8. B. Culbertson, R. Amerson, R. Carter, P. Kuekes and G. Snider, "Defect Tolerance on the Teramac Custom Computer," in Proc. *IEEE Symp. on FPGA's for Custom Computing Machines (FCCM'97)*, pp. 116–123, 1997
9. A. DeHon and H. Naeimi, "Seven Strategies for Tolerating Highly Defective Fabrication," *IEEE Design & Test of Computers*, vol. 22, Issue 4, pp. 306–315, 2005
10. S. C. Goldstein and M. Budiu, "NanoFabric: Spatial Computing using Molecular Electronics," in *Proc. Int. Symp. on Computer Architecture*, pp. 178–189, 2001
11. J. G. Brown and R. D. S. Blanton, "CAEN-BIST: Testing the Nanofabrics," in *Proc. Int. Test Conf. (ITC'04)*, pp. 462–471, 2004
12. M. Mishra and S. C. Goldstein, "Defect Tolerance at the End of the Roadmap," in *Proc. Int. Test Conf. (ITC'03)*, pp. 1201–1210, 2003
13. M. Tehranipoor, "Defect Tolerance for Molecular Electronics-Based NanoFabrics Using Built-In Self-Test Procedure," in *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT'05)*, pp. 305–313, 2005
14. Z. Wang and K. Chakrabarty, "Using Built-In Self-Test and Adaptive Recovery for Defect Tolerance in Molecular Electronics-Based Nanofabrics," in *Proc. Int. Test Conf. (ITC'05)*, pp. 477–486, 2005
15. M. B. Tahoori, "Defects, Yield, and Design in Sublithographic Nanoelectronics," in *Proc. Defect and Fault Tolerance in VLSI Systems (DFT'05)*, pp. 3–11, 2005
16. R. M. P. Rad and M. Tehranipoor, "SCT: An Approach For Testing and Configuring Nanoscale Devices," in *Proc. VLSI Test Symp. (VTS'06)*, pp. 370–377, 2006
17. J. L. Kouloheris and A. E. Gamal, "PLA-based FPGA Area versus Cell Granularity," in *Proc. IEEE Custom Integrated Circuits Conference*, pp. 4.3.1–4.3.4, 1992
18. R. M. Rad and M. Tehranipoor, "Fine-Grained Island Style Architecture for Molecular Electronic Devices," submitted to *Int. Symp. on Field Programmable Gate Arrays (FPGA'06)*, 2006
19. C. Stroud, S. Konala, P. Chen and M. Abramovici, "Built-In Self-Test of Logic Blocks in FPGAs (Finally, A Free Lunch: BIST without overhead!)," in *Proc. 14th VLSI Test Symposium*, pp. 387–392, 1996

Chapter 4: A Built-In Self-Test and Diagnosis Strategy for Chemically-Assembled Electronic Nanotechnology

J.G. Brown and R.D. (Shawn) Blanton

1 Introduction

Very recently, researchers have achieved revolutionary advances that may radically change the future of computing. By controlling the transfer of energy between molecules, molecular-scale structures can be used to perform computational tasks. As we approach the economic and physical limits of current solid-state electronics, traditional semiconductor devices become increasingly difficult to manufacture. Advances in physics, chemistry, and biology have exposed new research opportunities for "bottom-up" fabrication techniques [1–8]. These bottom-up techniques are referred to as chemical self-assembly. Unlike photolithographic and etch techniques used in CMOS technologies, bottom-up fabrication techniques rely on molecules assembling themselves into regular patterns to create a computing system. Molecular electronics will not only address the ultimate limits of miniaturization but also provide promising methods for novel manufacturing techniques.

Chemically-assembled electronic nanotechnology (CAEN) is under intense investigation as a possible alternative or complement to CMOS-based computing [9–15]. CAEN, also referred to as the nanoFabric, is a form of molecular electronics, which uses directed self-assembly and self-alignment to construct electronic circuits from nanometer-scale devices that exploit quantum-mechanical effects. CAEN-based systems consist of devices that are two-dimensional arrays of nano-scale wires that can be electronically configured for memory, logic, or signal routing applications [16, 17]. Although expected to have densities greater than 10^8 gate-equivalents/cm^2, the nanoFabric may possibly exhibit defect densities of up to ten percent. These highly-defective circuits will therefore require a completely new approach to manufacturing computational devices. In order to achieve any level of significant yield, it will no longer be possible to discard a chip once a defect is found. Instead, a method of using defective chips must be devised that will most likely focus on post-fabrication reconfiguration to determine the properties of the device in order to avoid or tolerate defects.

Testing and diagnosis of CAEN are required to achieve fault tolerance through reconfiguration. In this work, a built-in self-test (BIST) strategy called CAEN-BIST is described that uses a test and diagnosis methodology to identify faulty blocks in the nanoFabric. This methodology allows a defective nanoFabric to be utilized despite the presence of defects. A behavioral model for the nanoFabric is created in order to understand its behavior and to explore the effectiveness of various test strategies. This model provides an environment for fault simulation that allows diagnostic accuracy to be investigated. An analysis of previously published test strategies and our proposed algorithm is presented. The test strategy proposed is intended for the nanoFabric but can be applied to other, regular circuit architectures as well (e.g., FPGAs).

2 Related Work

An integrated circuit is discarded after manufacturing if some part of it is defective. One form of defect tolerance involves the ability to diagnose defective sections of a chip and map the design to a usable set of resources, thus avoiding defective components. Defect tolerance provides increased yield and therefore reduced manufacturing costs. A reconfigurable device consists of a set of resources, some of which may be defective. Once the defects have been located, a reconfigurable architecture can be programmed to avoid them.

Modern RAM chips are able to tolerate defects by using built-in redundancy. For example, if a memory cell is found to be defective, a spare column of cells can be used to replace the defective cell. First, testing is performed to identify and locate any failures in the memory. Next, a redundancy analysis is performed to determine which memory cells can be repaired with redundant resources. There are two types of repairs that can be made. A “hard” repair uses fuses, antifuses, or laser programming to disconnect rows or columns with faulty memory cells. On the other hand, a “soft” repair uses an address-mapping procedure to bypass faulty address locations. The primary reason for redundancy within memories is to enhance yield. Studies have shown that yield can be enhanced 5–20% [18]. However, in CAEN-based systems, it is highly unlikely that a similar approach is viable. Only a small percentage of resources are typically available for replacement, so if a large portion of the system is defective, replacement will not be possible. Also, there is no way to ensure that the components used for replacement are defect-free. Therefore, simple row/column replacement will not be a viable approach for defect tolerance in the nanoFabric.

Since the nanoFabric has the advantage of reconfiguration, a natural solution to the defect density problem is likely suggested by studying other reconfigurable fabrics [19] such as Field-Programmable Gate Arrays (FPGAs). An FPGA consists of an array of programmable logic blocks (PLBs) interconnected by a programmable routing network. The function of the

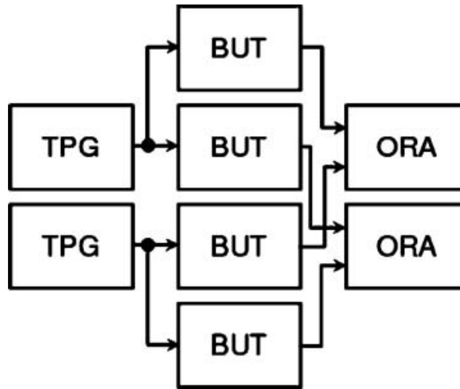


Fig. 1. Logic blocks are programmed as test pattern generators (TPGs), blocks under test (BUTs), or output response analyzers (ORAs) for FPGA-BIST [20]

device is determined by the programming bits, or *configuration*. Application-independent testing of such a device is complicated by the fact that all possible modes of operation must be covered.

FPGA-BIST [20, 21] is a test and diagnosis strategy that makes use of the reconfigurability of an FPGA to provide a complete test for all PLBs. In this BIST approach, the logic blocks of the FPGA are divided into three groups. As shown in Fig. 1, the PLBs are configured as either test pattern generators (TPGs), blocks under test (BUTs), or output response analyzers (ORAs). The TPG applies exhaustive test patterns to the BUTs, which send output responses to the ORA, and the ORA compares responses from multiple BUTs to determine if there is a defect. Since the PLBs of an FPGA are identical, there is no need to store output responses for testing. Defect-free PLBs have the same output responses, so the ORA only needs to compare the responses received. As shown in Fig. 1, each ORA compares the responses of BUTs that have different TPGs. This strategy is used to avoid the case in which a faulty TPG applies an incomplete test which would lead to a faulty BUT escaping detection. After the exhaustive test set is applied to the BUTs, the configurations of the blocks are then interchanged to ensure that every PLB is tested.

FPGA-BIST also takes advantage of reconfigurability for diagnosis of faulty PLBs. By configuring an FPGA according to the floorplans shown in Fig. 2, each TPG is configured to test different sets of BUTs for each configuration. Therefore, the results of FPGA-BIST provide increased diagnostic resolution. The reconfigurable nature of the nanoFabric will surely play a pivotal role in developing effective test strategies. However, FPGA test and diagnosis strategies [20–23] do not account for an extremely high defect density, nor a large numbers of components. Test and diagnosis of a CAEN-based system requires a new approach.

The EasyPath Solution [24] from Xilinx is an effort towards defect tolerance for FPGAs. Chips that are manufactured with defects cannot be used for

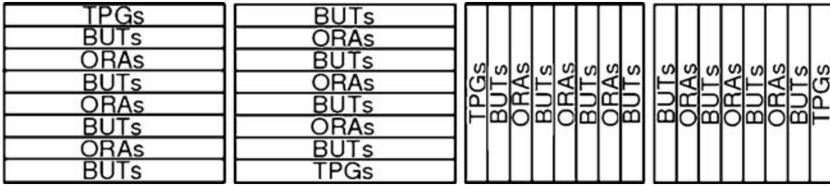


Fig. 2. Multiple test configurations of an FPGA provide increased diagnostic resolution [21]

all possible applications. However, defective chips may be usable for designs that do not make use of the defective portions of the chip. By testing the defective chip for the customer's specific design, it is possible to salvage these chips for use. Although these defective FPGAs are no longer reconfigurable, the ability to use them in this limited fashion improves yield and thus reduces manufacturing cost.

Teramac [17, 25] is an FPGA-based system capable of running user designs at one megahertz despite a defect density of nearly 3%. This system consists of 864 FPGAs, with over 650,000 gate equivalents. Testing is performed by downloading designs called *signature generators*. These are sets of circuit components, including switches, wires, and logic blocks, that are configured as linear feedback shift registers (LFSRs). Long, psuedo-random number strings are generated and communicated throughout the set of components. If the final bit stream generated by the LFSR is correct, all the components are assumed to be defect-free. If the bit stream is incorrect, there must be a defective component, so these components are used to create new signature generators. The resources found at the intersection of the defective LFSR configurations are recorded as defective in a defect database. The database is used to ensure that the system is programmed to use only defect-free components. Teramac provides empirical evidence that reliable systems can be created from unreliable components. To utilize a nanoFabric, defective components must be tolerated in a similar fashion.

Since the introduction of CAEN-BIST [26], Wang and Chakrabarty [27] have introduced a nanoFabric BIST strategy. Similar to FPGA-BIST, their approach configures each block as either a TPG, BUT, or ORA. Their strategy provides 100% fault coverage for stuck-at, stuck-open, bridge, and connection faults. Moreover, since every BUT is tested in parallel, the complexity of the algorithm is independent of the size of the nanoFabric.

3 NanoFabric

The CAEN-based system analyzed here is an architecture called the nanoFabric [9–14]. In this section, we describe the nanoFabric architecture (Sect. 3.1), defect tolerance for the nanoFabric (Sect. 3.2), our simulation model of the nanoFabric (Sect. 3.3), and algorithms for testing the nanoFabric (Sect. 3.4).

3.1 Architecture

The nanoFabric is a regular architecture that consists of an array of interconnected nanoBlocks (squares) and switchBlocks (circles) as shown in Fig. 3. Unlike typical CMOS circuits, the nanoFabric does not have a dependence on the underlying substrate. Therefore, as presented in [28, 29], the nano-scale circuitry can be fabricated on top of a conventional CMOS circuit. This capability enables an interface between nano-scale components and CMOS circuitry. The interface is used to apply configuration bits and supply power and ground to each component in the nanoFabric.

A nanoBlock is the fundamental unit of the nanoFabric and is analogous to a PLB of an FPGA. A set of configuration bits establishes the logic behavior for each block and the interconnections between blocks. A nanoBlock consists of a molecular logic array (MLA) of switches to store the configuration of the block, latches used for sequential circuit implementation, and I/O for forming connections to adjacent nanoBlocks within the nanoFabric. The MLA is a mesh of wires, commonly referred to as a crossbar architecture, as illustrated in Fig. 4. Carbon nanotubes, silicon nanowires, and other nano-scale wires have been used in this architecture. The regularity of the crossbar architecture provides an inherent defect tolerance since any row or column can be selected to implement a logic function.

At each intersection of wires, there is a carbon-based molecule called rotaxane that acts as a switch. The structure of the molecule resembles a ring on

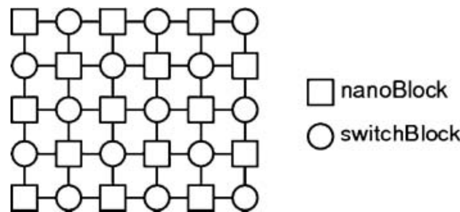


Fig. 3. The nanoFabric is an array of interconnected nanoBlocks and switch-Blocks [9]

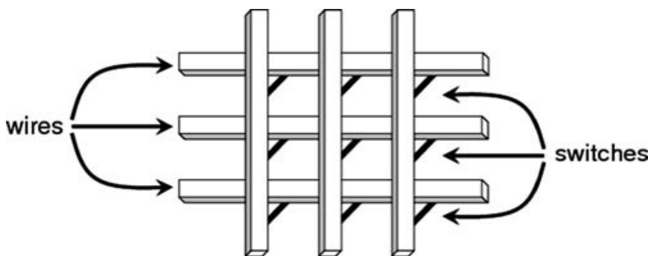


Fig. 4. The crossbar architecture is comprised of a mesh of nano-scale wires with molecular switches at each intersection [9]

a rod. When a voltage is applied to the rod, the position of the ring changes. The position of the ring determines the resistance of the molecule. The current for a closed switch has been reported to be 60–80 times greater than that of an open switch [30]. Therefore, the rotaxane switch can be configured to behave as a diode-connection ("on") or a highly-resistive open ("off").

As shown in Fig. 5a, the switches shown in black are configured as diode connections and the remaining switches are configured as highly resistive opens. By applying inputs A and B to the first two horizontal wires of the nanoBlock and configuring the first vertical wire as output Z, the diode-resistor logic implementation shown in Fig. 5b is created. This circuit performs the logic-AND function. NanoBlocks can be used to create simple logic gates, as well as more complex logic functions.

The region between a set of nanoBlocks is known as a switchBlock, as depicted in Fig. 6. A switchBlock is also reconfigurable and serves to connect wire segments of adjacent nanoBlocks. The configuration bits of the switchBlock determine the direction of data flow through the block. The switchBlocks of the nanoFabric provide the interconnect between each of the nanoBlocks.

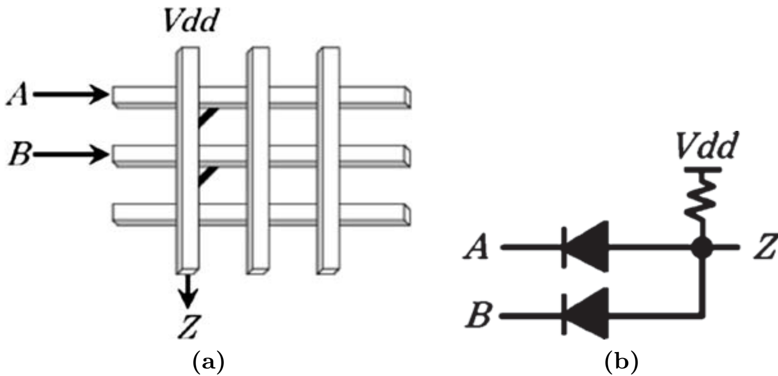


Fig. 5. In (**a**), the connections shown in black are configured as diodes while the remaining connections are configured as opens. This nanoBlock configuration produces the diode-resistor logic implementation shown in (**b**)

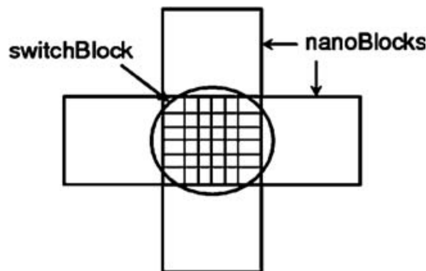


Fig. 6. A switchBlock is the region between a set of nanoBlocks used to determine the direction of data flow [9]

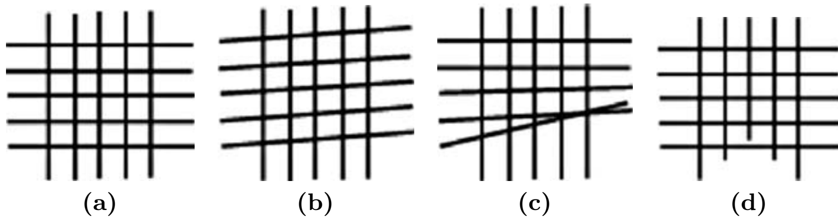


Fig. 7. Non-deterministic fabrication methods may lead to (**a**) a perfectly aligned mesh of wires, (**b**) alignment problems, (**c**) unwanted overlaps, or (**d**) missing overlaps [12]

Unlike traditional CMOS technologies, CAEN involves self-assembly and self-alignment methods. This approach is most effective at creating random or very regular structures. Aperiodic structures produced through photolithography will not be an option for this technology. Furthermore, like any manufacturing process, the fabrication of CAEN is imperfect. Unlike the ideal mesh of wires (Fig. 7a) expected in a crossbar architecture, a nanoBlock may have alignment problems (Fig. 7b), unwanted overlaps (Fig. 7c), or missing overlaps (Fig. 7d) [12]. The challenges involved with the nanoFabric are unique and require novel test and diagnosis strategies in order to utilize a defective fabric.

3.2 Defect Tolerance

Reconfigurability has provided a new outlook for test strategies and defect tolerance. However, in order to take advantage of the nanoFabric's reconfigurable architecture, new test strategies must be explored to create a *defect map*. As shown in Fig. 8, the shaded shapes represent defective components in an example nanoFabric. However, rather than discarding this device, we desire a test phase that results in a defect map that indicates which components are defective. Figure 8 shows a defect map that has defective components marked with an ×. The defect map provides an opportunity for defect tolerance. If the defective components can be located, the configuration of the chip can be adjusted through defect-aware place-and-route [31] to ensure that only defect-free components are used to program the application functionality. The defect map is specific to each particular fabric and is used to route around the defective components. Creating the defect map for a nanoFabric, however, requires new test and diagnosis strategies.

First, the defect density of CAEN is higher than standard CMOS-based circuits so many test and diagnosis algorithms intended for CMOS will not be effective. Among other reasons, these algorithms either assume a single instance of a defect [32], a small number of defects [33], or that the misbehavior of a defective circuit can be captured by conventional fault models [32]. However, the high defect density of the nanoFabric makes such algorithms ineffective.

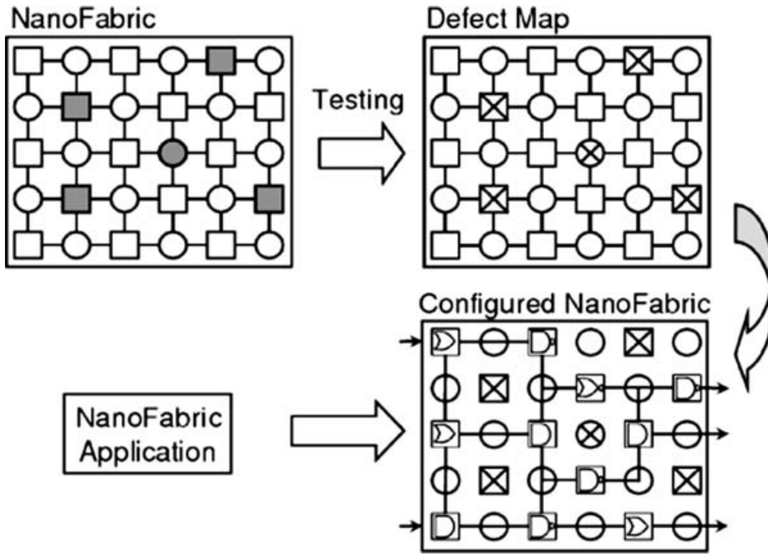


Fig. 8. Testing of the nanoFabric provides a defect map that can be used to avoid defective components when programming the application functionality [13]

Next, a nanoFabric may contain billions of components. Therefore, the complexity of the test algorithm is a very important issue. Separately testing each component is obviously not viable. Similar to what was done in [17, 25], creating signature generators that contain a large number of circuit components would improve test time, but would also increase the complexity of the test required to achieve comparable diagnostic resolution. Given the expected defect density, it is highly unlikely than any reasonably large set of components would be defect-free. Therefore, signature generators are not likely to be useful. For a nanoFabric, the large number of components and high defect density provide hurdles for the creation of a defect map. NanoFabric test requires a unique strategy to overcome these challenges.

A metric for describing the quality of a defect map is referred to as *recovery* [13]. Recovery is the percentage of defect-free components that are correctly diagnosed. This metric measures the diagnostic accuracy of a particular test strategy, but assumes there are no undetected defects. The ideal recovery of 100% means every component in the nanoFabric is diagnosed correctly. However, since the actual number of defect-free components cannot be known, this metric is meant solely for assessing simulation results.

3.3 Simulation Model

We created a behavioral model of the nanoFabric using C++ to determine the effectiveness of various test strategies. The nanoFabric model consists of

a set of interconnected nanoBlocks and switchBlocks as shown in Fig. 3. The number of nanoBlocks and switchBlocks in the nanoFabric (n), the number of wires in each nanoBlock ($k \times k$), and defect density (d) are all configurable quantities. The model created for the nanoFabric is a generalized model and is intended to simulate the high-level behavior of the nanoFabric. Therefore, the characteristics of diodes were not included in the nanoBlock configurations. Moreover, issues such as gain and isolation were not taken into account.

Each nanoBlock has input and output connections to adjacent nanoBlocks. Although the CMOS-CAEN interface [28, 29] may provide I/O for each of the nanoBlocks in the nanoFabric, in order to truly take advantage of nanometer-scale components, the nanoFabric interface should be minimized. Therefore, test algorithms should not assume that there is an unlimited amount of I/O. A limited I/O however makes the internal blocks of the fabric more difficult to control and observe. Moreover, since the defect rate in the fabric is extremely high, controlling and observing internal blocks becomes even more of a challenge.

The model created for the nanoBlock is similar to that of a programmable logic array (PLA). As shown in Fig. 9, a PLA has a set of input lines $I_1, \ldots, I_l$, a set of product lines $P_1, \ldots, P_m$, and a set of output lines $Z_1, \ldots, Z_q$.

Logic values are applied to the input lines and based on the connections between the input and product lines, the logic-AND function is performed on the input lines connected to a particular product line. For example, if connections are made from the input lines I_1 and I_2 to product line P_1, the value of P_1 equals $(I_1 \cap I_2)$. The set of connections between input lines and

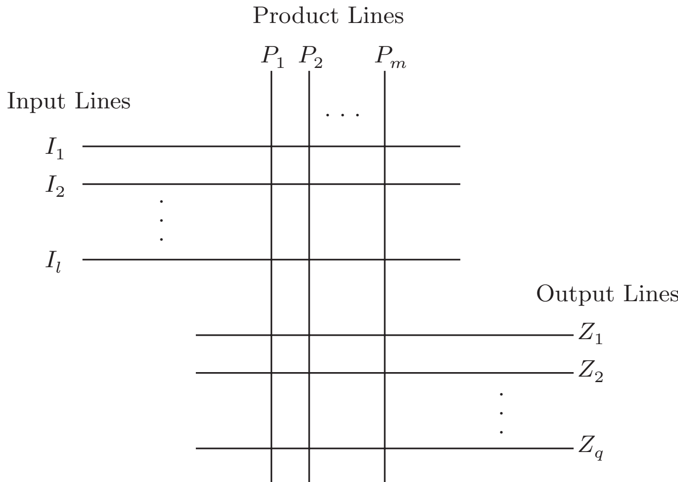


Fig. 9. A PLA consists of a set of input lines $I_1, \ldots, I_l$, a set of product lines $P_1, \ldots, P_m$, and a set of output lines $Z_1, \ldots, Z_q$

product lines is therefore known as the AND plane. The connections in the AND plane are defined in a personality matrix

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \vdots & \vdots & \dots & \vdots \\ A_{l1} & A_{l2} & \dots & A_{lm} \end{pmatrix}.$$

A_{ij} equals zero if there is a connection between input line I_i and product line P_j. Otherwise, A_{ij} equals one. If there are no connections between a product line and any input line, the nanoBlock model assigns floating values to the product line. Otherwise, the logic value of the product line is

$$P_j = \prod_{i=1}^{l}(I_i + A_{ij})]. \tag{1}$$

Connections between product lines and an output line form a logic-OR function. For example, if product lines P_1 and P_2 are connected to output line Z_1, the value of Z_1 equals $(P_1 \cup P_2)$. The set of connections between product and output lines is therefore known as the OR plane. The connections in the OR plane are defined in a personality matrix

$$\begin{pmatrix} B_{11} & B_{12} & \dots & B_{1m} \\ B_{21} & B_{22} & \dots & B_{2m} \\ \vdots & \vdots & \dots & \vdots \\ B_{q1} & B_{q2} & \dots & B_{qm} \end{pmatrix}.$$

B_{ij} equals one if there is a connection between product line P_j and output line Z_i. Otherwise, B_{ij} equals zero. If there are no connections between an output line and any product line, the nanoBlock model assigns floating values to the output line. Otherwise, the value of the output line is

$$Z_p = \sum_{j=1}^{m}(B_{pj}P_j)]. \tag{2}$$

In the nanoBlock model, horizontal and vertical wires can be configured as either input, product, or output lines. The configuration inputs of the nanoBlock, shown as diode connections in Fig. 10, determine the functionality of the block. Based on (1) and (2), an output Z_p is assigned the appropriate values according to

$$Z_p = \sum_{j=1}^{m}[B_{pj}\prod_{i=1}^{l}(I_i + A_{ij})], \tag{3}$$

where l is the number of input lines and m is the number of product lines. For example, in Fig. 10, the top two horizontal wires are configured as inputs I_1

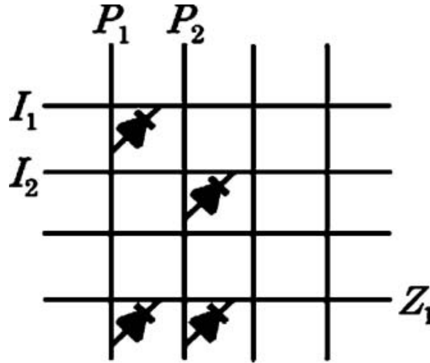


Fig. 10. A nanoBlock configured as a two-input OR gate. Active switches are shown as diode connections

and I_2. The bottom horizontal wire is configured as output Z_1. Therefore, the connections along the top two wires are A connections, while the connections along the bottom wire are B connections. The product lines P_1 and P_2 assume the values of I_1 and I_2, respectively, due to the A connections, and the B connections create a logic-OR function of the two product lines. Therefore, the output Z_1 equals $(I_1 \cup I_2)$. The wires of the nanoBlock can be reconfigured as either inputs or outputs, which can change the behavior of these connections and therefore change the logic function of the block.

A nanoBlock consists of a mesh of k horizontal wires and k vertical wires with k^2 configuration bits. A $k \times k$ nanoBlock can be used to implement a logic function with $(2k-1)$ inputs and one output, $(2k-2)$ inputs and two outputs, $(2k-3)$ inputs and three outputs, etc. The selection of wires in the block is arbitrary, however. Any of the wires, horizontal or vertical, can be configured as inputs or outputs to implement the desired logic function. In the model, a nanoBlock can also be configured as a tester. As a tester, a nanoBlock applies a test pattern, receives a response, and stores a binary value to indicate if the BUT has passed all tests.

The model for a switchBlock is quite simple, consisting of four ports (north, south, east, and west) and twelve control bits used to determine the direction of data flow as shown in Fig. 11. Each control bit enables data to flow from one port of the switchBlock to another. For example, if control bit zero is asserted, data flows from the west to the north.

A set of nanoBlocks uses the adjacent switchBlocks to create complex logic functions. Figure 12 is an example in which nanoBlocks 1 and 2 are used to create AND gates and the adjacent switchBlock transfers the outputs of these gates to nanoBlock 3, which is configured as an OR gate. This configuration creates a sum-of-products

$$Z_2 = I_5 + I_4 = Z_1 + Z_0 = I_3 I_2 + I_1 I_0.$$

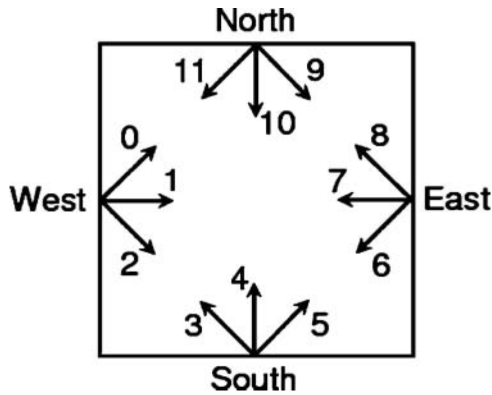


Fig. 11. The control bits of the switchBlock determine the flow of data between its four ports

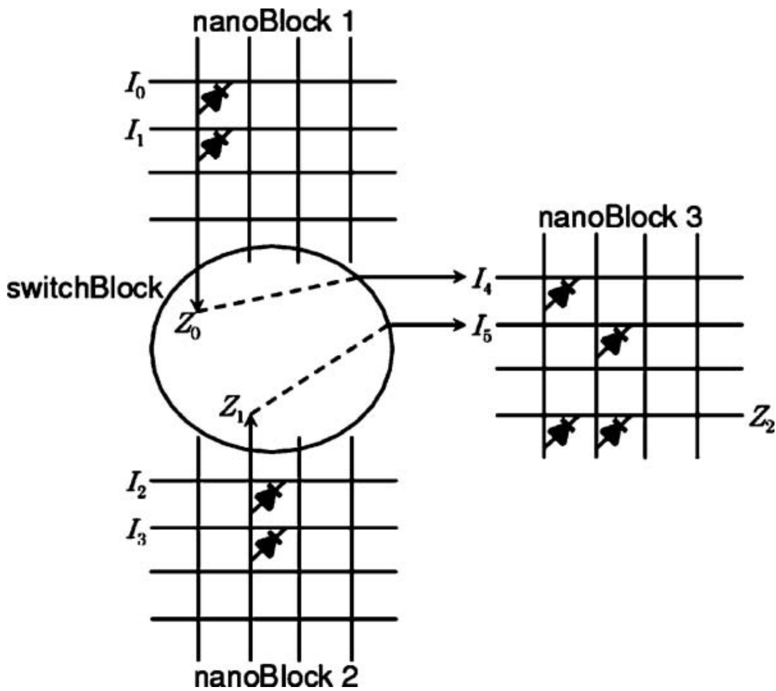


Fig. 12. Three nanoBlocks and a switchBlock used to create a sum-of-products function

In order to simulate a faulty nanoFabric, fault injection is performed. A nanoBlock or switchBlock is selected randomly from the nanoFabric and a randomly selected fault type is applied to the model for that block. Fault types include traditional PLA fault models such as stuck-at, connection, and bridge

faults. A stuck-at fault is injected by randomly selecting a wire in a nanoBlock or switchBlock and driving it to a specific value. Likewise, connection faults are injected by randomly selecting a bit from the personality matrix of a nanoBlock or a control bit of a switchBlock and driving it to a specific value. Additionally, bridge faults are injected by inserting logic functions between wires (AND-type, OR-type bridge fault) or overriding one signal with another (dominant bridge fault). The probability of a nanoBlock or switchBlock being defective is determined by the defect density d, and a block may have any number of faults. Therefore, the average nanoFabric would have $d\%$ defective blocks and $(100 - d)\%$ defect-free blocks. Once the faults are injected, an exact defect map is created based on the locations of the injected faults. This "golden" defect map is used to determine the diagnostic accuracy.

Based on the specified values of n, k, and d, a defective nanoFabric is created and simulated for a given test strategy. A second defect map is derived from the test-simulation results. A comparison between the derived and golden defect maps allows recovery to be calculated. To robustly examine the effectiveness of test strategies, thousands of simulations are performed for different sizes and defect densities of the nanoFabric.

3.4 Test Algorithms

In order to understand the effectiveness of testing, some notation is defined. First, the time required to apply a test pattern is defined as t_{test}. Second, the time required to reconfigure the entire nanoFabric is defined as t_{config}. Next, the time required for an LFSR signature generator to generate a final signature is defined as t_{LFSR}. Although no assumptions have been made as to the values of these parameters, one can speculate that the configuration time is substantial compared to t_{test}. Also, the time required for an LFSR to generate a signature is substantial compared to t_{test}. The values of these parameters are dependent on the number of nanoBlocks and switchBlocks in the fabric (n) and the number of wires in a nanoBlock ($k \times k$). The values of these three timing parameters are used to characterize the complexity of each nanoFabric test algorithm.

We explored several test algorithms using the nanoFabric model. The first algorithm creates signature generators (LFSRs, counters, etc.) from rows and columns of nanoBlocks. By creating large signature generators, test time can be reduced. The circuit generates a signature, which can be used to determine if there are defects in any of the nanoBlocks. If the tests pass, all the blocks in the signature generator are marked as defect-free. However, if any of the tests fail, all of the blocks are marked as defective. Therefore, one defective block in a large signature generator forces all the blocks in the circuit to be diagnosed as defective. By creating signature generators from rows of nanoBlocks and then repeating the process for each of the columns, diagnostic accuracy is improved. For example, if the first row of nanoBlocks is found to be defective and the first column of nanoBlocks is found to be defective, then

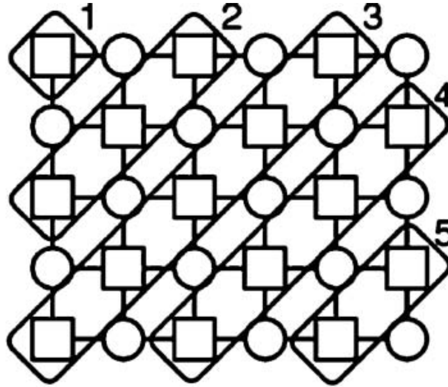


Fig. 13. Signature generators are created using diagonals of components

the nanoBlock that is located at the intersection of these two circuits must be defective. This algorithm requires two configuration phases and therefore two signature generation phases. Therefore, the time required for this algorithm to execute is $2t_{config} + 2t_{LFSR}$.

A similar algorithm involves diagonal signature generators. This strategy creates signature generators from diagonals of nanoBlocks. In a manner similar to row-column testing, if a defect is detected, the entire diagonal is diagnosed as defective. As shown in Fig. 13, the first three diagonals from the upper-left corner are labeled 1, 2, and 3, respectively. In this approach, the signature generators are of varying sizes, which allows for improved diagnostic resolution in some cases. For example, in Fig. 13, diagonal 1 has only one nanoBlock, while diagonal 3 has five. Smaller signature generators allow higher diagnostic resolution. Similar to the row-column approach, diagonal testing requires two configuration phases and therefore two signature generation phases. Therefore, the time required for this algorithm to execute is also $2t_{config} + 2t_{LFSR}$.

As shown in Fig. 14, for a defect density $d = 10\%$, row-column testing achieves an average recovery just greater than 60% while diagonal testing results in an average recovery of approximately 95%. However, these results assume a fabric size of $n = 25$ blocks. Although the time complexity is reasonable, these algorithms prove ineffective for larger circuit sizes. As shown in Fig. 15, the average recovery for these algorithms drops below 10% for fabrics that are larger than $n = 2{,}000$ blocks. Since the nanoFabric is expected to consist of millions of nanoBlocks, it is highly unlikely that either of these approaches are viable.

The "none-some-many" strategy is described in [13]. This approach creates LFSR-based signature generators from a random selection of nanoBlocks. Based on the results of the test set, each nanoBlock is assigned a probability of being defective. NanoBlocks that exceed a certain probability threshold are marked as defective in the defect map and removed from the fabric. New signature generators are created and the test is performed again for the remaining

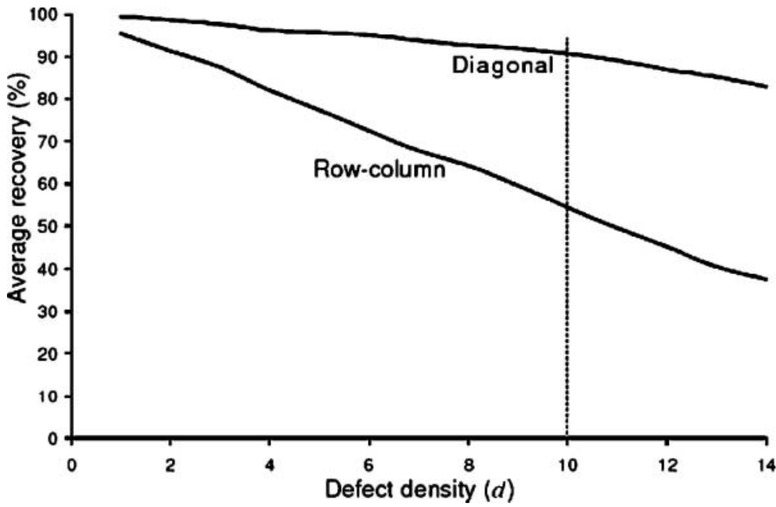


Fig. 14. Average recovery for row-column and diagonal testing as the defect density d increases, $n = 25$, $k = 10$

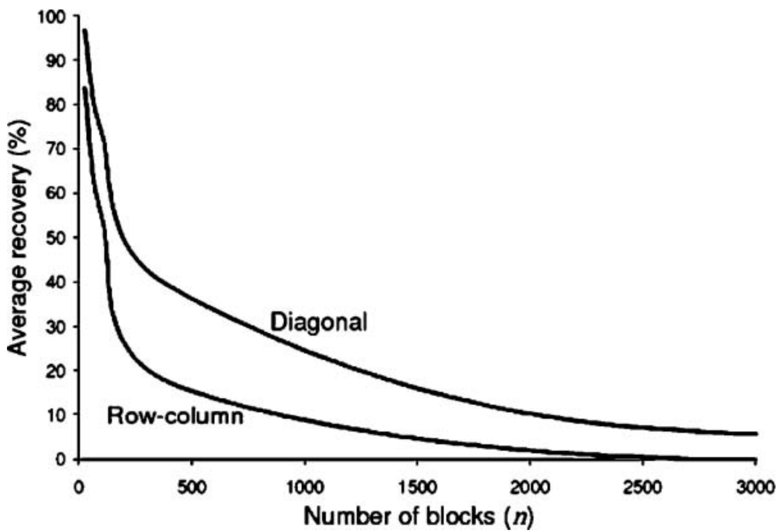


Fig. 15. Average recovery for row-column and diagonal testing as the number of components n increases, $k = 10$, $d = 10$

nanoBlocks. This process repeats until all the nanoBlocks are diagnosed as defective or defect-free. The average recovery is reported to be approximately 55% for $d = 10\%$. This approach however assumes an unlimited amount of interconnect between nanoBlocks in order to create signature generators consisting of randomly selected nanoBlocks.

3.5 CAEN-BIST

The test strategy presented in this chapter is a BIST method for the nanoFabric we call CAEN-BIST. This method not only enables the nanoFabric to test itself, but also stores the results of the tests internally. Since standard BIST approaches require only a small percentage of the circuit components, many algorithms assume that the additional hardware implemented for BIST is defect-free. However, due to the high defect density, no such assumptions can be made for the nanoFabric. Therefore, all blocks of the nanoFabric must be tested. CAEN-BIST is unlike traditional BIST since test pattern generation is not accomplished within the nanoFabric. The test patterns and configurations are created externally and delivered to the fabric. However, the output response analysis and diagnosis are accomplished internally.

The process of testing a nanoBlock requires a set of test patterns and a set of configurations. Since a nanoBlock can be configured to implement virtually any logic function, the block must be tested completely for any possible configuration. A nanoBlock has similar structure and behavior to that of a PLA and therefore should make use of similar fault models. These fault models include stuck-line, connection, and bridge faults [32].

The nanoBlock is configured for testing as shown in Fig. 16. A walking binary sequence is applied to the inputs of the nanoBlock. For a nanoBlock with 4 inputs, for example, the following test patterns are applied: 1000, 0100, 0010, 0001 as shown in Fig. 16. This configuration causes the walking sequence to appear as outputs on the vertical wires. If a walking sequence does not appear correctly, a fault is detected in the nanoBlock. After the test sequence is applied, the configuration is shifted, as shown in Fig. 17. The walking sequence is repeated until all connections in the nanoBlock have been tested. For a nanoBlock with k^2 connections, k test patterns are applied to k different configurations for a total of k^2 tests. This test set ensures that every connection in the nanoBlock is tested in both the "on" and "off" configurations and therefore provides 100% fault coverage for single stuck-line, connection, and bridge faults [34, 35].

The switchBlocks of the nanoFabric are not directly tested in this approach. However, a defective switchBlock is detected during the testing of

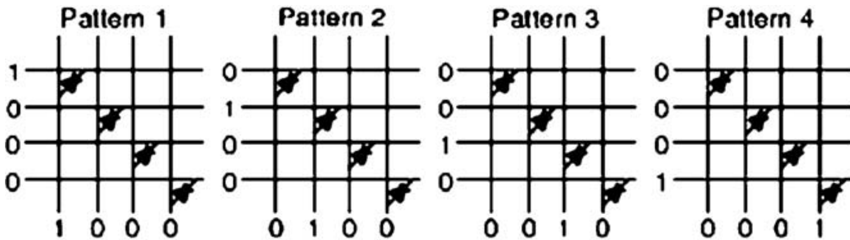


Fig. 16. A walking sequence of ones is applied to the horizontal wires and the output response is transmitted along the vertical wires

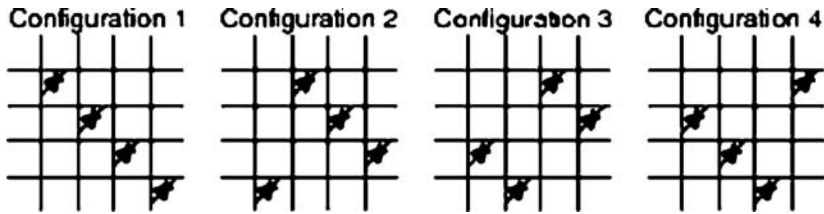


Fig. 17. The configuration of the nanoBlock is shifted after all test patterns have been applied and the process is repeated until all connections have been tested

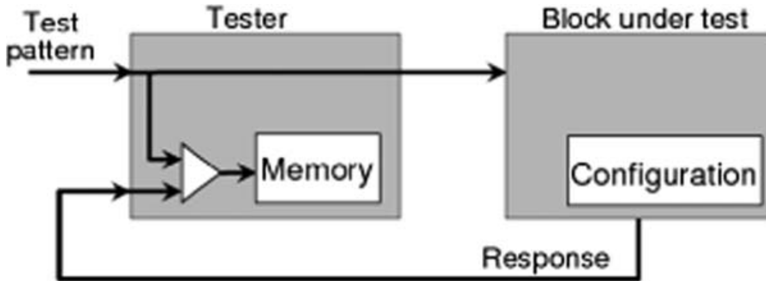


Fig. 18. A nanoBlock configured as a tester applies a test pattern to its neighboring nanoBlock, the block under test (BUT), compares the output response to the test pattern applied, and stores the status of the tested neighbor

its adjacent nanoBlock. Since the CAEN-BIST approach tests a nanoBlock multiple times, the diagnosis of a switchBlock is made accurately.

In CAEN-BIST, nanoBlocks are used to test neighboring nanoBlocks [36]. When configured as a tester, the nanoBlock passes a test pattern to the block under test (BUT) that sends an output response back to the tester as depicted in Fig. 18. The BUT is configured to receive the test pattern and generate an output response that matches the test pattern. Therefore, if the nanoBlock is defect-free, the output response matches the test pattern. Therefore, there is no need to store the correct response. The tester compares the test pattern to the response received to determine if the BUT is defective. Unlike standard BIST techniques, there is no signature-based compaction, so there are no aliasing problems.

Assuming the tester has memory storage [37], it stores a binary test outcome to indicate whether the neighbor is defective. Each nanoBlock has a single bit of storage for each of its three neighbors, which represents the status of the neighboring nanoBlock. Rather than storing the status of a nanoBlock within itself, the neighbors of a nanoBlock indicate whether it is defective. This approach is used to avoid the problem of defective nanoBlocks marking themselves as defect-free. As a result, if a nanoBlock is defective, the neighbors, when configured as testers, store a binary value to indicate that the neighboring nanoBlock is defective and that it should not be used. Each neighbor

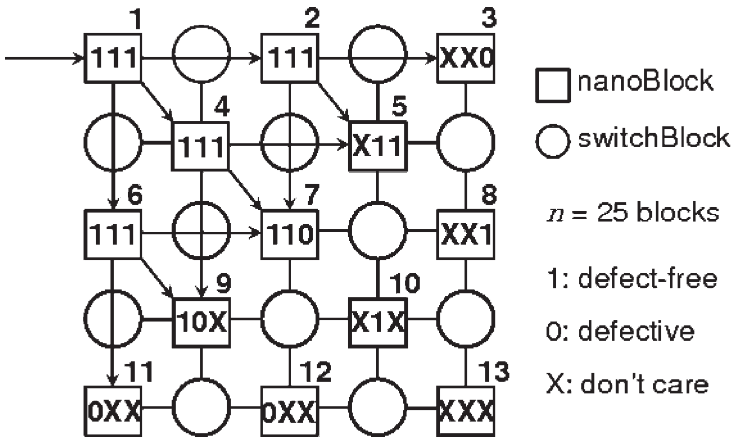


Fig. 19. Each nanoBlock stores three bits to indicate the condition of its three neighbors

is tested individually by a tester, but an entire diagonal of nanoBlocks is simultaneously configured as testers in order to reduce test time.

CAEN-BIST is performed in a wave-like manner [13]. First, the corner nanoBlock of the fabric is tested by an external tester and if this nanoBlock is defect-free, it is configured as a tester and used to test its three neighbors. In Fig. 19, for example, nanoBlock 1 is configured as a tester to test its three neighbors, 2, 4, and 6. After these three nanoBlocks are tested, nanoBlock 2 tests nanoBlock 3, nanoBlock 4 tests nanoBlock 5, and nanoBlock 6 tests nanoBlock 7, simultaneously. Each of these testers (2, 4, and 6) then test their next neighbors (5, 7, and 9) and once all three of their neighbors have been tested, the next diagonal of nanoBlocks becomes testers. Each diagonal is configured as testers until the entire fabric has been tested.

There are $\sqrt{n}$ diagonals to be tested, each tester tests three neighboring nanoBlocks, and k^2 test patterns are applied to each nanoBlock. Therefore, the total number of test patterns for CAEN-BIST is $3k^2\sqrt{n}$. The number of reconfigurations required is $3k\sqrt{n}$. Therefore, the time for this algorithm to execute is $(3k\sqrt{n})t_{config} + (3k^2\sqrt{n})t_{test}$.

Once the entire fabric has been tested, each nanoBlock has three bits stored internally to indicate the condition of its three neighboring nanoBlocks as shown in Fig. 19. The symbol "1" indicates a defect-free neighbor, "0" indicates a defective neighbor, and "X" indicates that there is no neighbor. In this particular example, nanoBlocks 7, 9, and 11 have all tested nanoBlock 12 and stored a "0" value in memory to indicate that nanoBlock 12 is defective. Therefore, nanoBlock 12 is diagnosed as defective since each tester was found to be defect-free by other testers. Although nanoBlock 12 has stored a "0" to indicate that nanoBlock 13 is defective, this result is arbitrary and cannot be used for diagnosis since a defective nanoBlock cannot be trusted as a reliable

tester. Also, nanoBlock 3 indicates that nanoBlock 8 is defective. However, nanoBlocks 5 and 7 indicate that the same nanoBlock is defect-free. Since nanoBlocks 3, 5, and 7 were all found to be defect-free, there must be a defect in the interconnect between nanoBlock 3 and nanoBlock 8. Therefore, the switchBlock between these two nanoBlocks is diagnosed as defective, but nanoBlock 8 is diagnosed as defect-free. In the case where there is no tester available to test a nanoBlock, the untested nanoBlocks are assumed to be defective.

The corner nanoBlocks are tested in an initial phase and their test outcomes are stored externally. All other nanoBlocks are assumed to be defective until a defect-free nanoBlock marks that block as defect-free. Since the defect map is kept internal during test, the defects are transparent to the outside world, and testing the fabric becomes less complicated since the nanoBlocks are aware of their defective neighbors. Furthermore, the need to observe effects of defects is eliminated. As long as the nanoBlocks are controllable, they can be tested effectively.

Controlling the nanoBlock is a challenge however. Since there is a large number of nanoBlocks and a high defect density, finding a path of defect-free nanoBlocks through the fabric is a non-trivial task. In Fig. 20, testers are labeled with a “1” while the BUT is labeled with a “2”. In order to pass data from the corner nanoBlock to the testers, there is a defective nanoBlock (marked with an ×) that must be avoided. However, since each nanoBlock is aware of its defective neighbors, the data is only passed along the solid lines shown. Based on the bit stored to indicate whether a neighbor is defective, a nanoBlock is configured to only communicate with a neighbor that is defect-free. Defect-free nanoBlocks disable the ports (shown as dotted lines) of the switchBlocks that are adjacent to the defective nanoBlock. This terminates

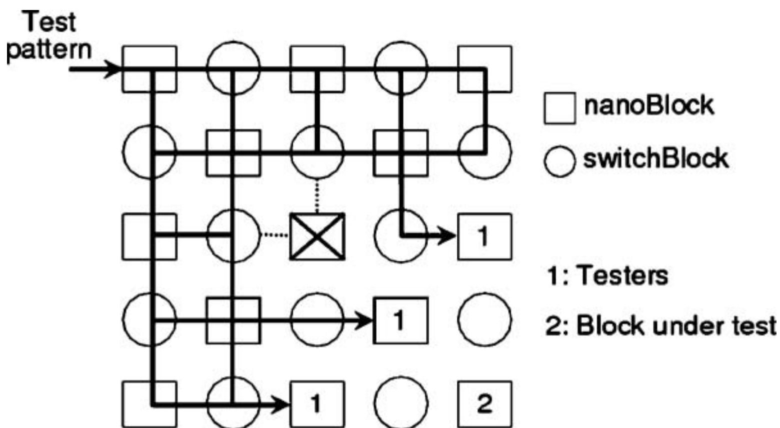


Fig. 20. Defective nanoBlocks must be avoided to ensure a defect-free path to the testers

communication to and from the defective block so that no corrupted data can be sent through the fabric. Therefore, a defect-free path can be assured for the testers, if one exists. If a path from the corner nanoBlock to a tester does not exist, a set of nanoBlocks may go untested and are assumed to be defective.

CAEN-BIST requires an external tester to apply configurations and test patterns. However, the test patterns and configurations being applied to the nanoFabric are independent of the defect map. The external tester has a standard sequence of test patterns and configurations. The only requirements of the external tester are to test the corner nanoBlocks of the fabric, configure each diagonal of the nanoFabric as testers and BUTs sequentially, and repeat a sequence of test patterns to the corner nanoBlock. The corner nanoBlock tests its neighbors with the test patterns received and then configures the adjacent switchBlocks to avoid any defects found. Next, the corner nanoBlock passes the test patterns to the next set of testers and the process repeats until the entire fabric has been tested. Since the nanoBlocks configure around their defective neighbors by enabling and disabling their adjacent switchBlocks, the external tester does not need to have any knowledge of the defect map. The sequence of test patterns and configurations are consistent for any nanoFabric under test. This external tester can therefore be implemented with a simple finite state machine, perhaps in the underlying CMOS.

A problem arises when corner nanoBlocks are defective. If a corner nanoBlock is defective, there is no defect-free path into the nanoFabric and so testing cannot be carried out effectively. However, since the selection of the corner nanoBlock is arbitrary, any corner nanoBlock can be used as the entry point for CAEN-BIST. Also, if the defect map is downloaded from the fabric, the process can be repeated from all the defect-free corners in order to provide higher diagnostic resolution based on four defect maps. If a nanoBlock is diagnosed as defect-free during any of the iterations, the nanoBlock is defect-free. Unfortunately, for a nanoFabric that has four defective corner nanoBlocks, the algorithm has no entry point and therefore diagnoses the entire nanoFabric as defective. Once testing is complete, the circuitry used during testing can be reprogrammed for application functionality.

4 Simulation Experiments

For the following experiments, 1,000 different versions of the nanoFabric were created for each combination of values for the parameters n, k, and d. A defect map is created for the nanoFabric based on test-simulation results and it is compared to the golden defect map to determine the recovery. Recovery is averaged over 1,000 nanoFabric simulations and is therefore referred to as *average recovery*.

CAEN-BIST has been found to provide an average recovery greater than 92% for defect densities reaching as high as 10%, and average recovery remains nearly 80% for defect densities up to 20%. The average recovery can

Table 1. Average recovery for increased iterations of CAEN-BIST for $n = 1$ million, $k = 10$, and $d = 10\%$

No. of iterations	Average recovery (%)
1	92.1
2	97.9
3	99.6
4	99.9

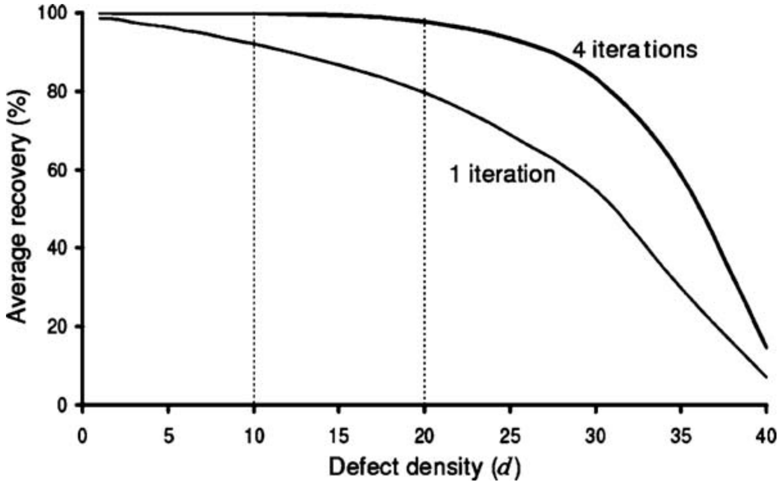


Fig. 21. CAEN-BIST provides high average recovery for increasing defect densities. Four iterations of CAEN-BIST provides nearly 100% average recovery for defect densities up to 20%

be improved however by repeating the algorithm from all the corners of the fabric. Although one corner provides acceptable results, the average recovery can be improved to nearly 100% by repeating the process for each corner of the fabric. As shown in Table 1, the average recovery for four iterations of CAEN-BIST is 99.9%.

Moreover, the average recovery remains high for increasing defect densities. As shown in Fig. 21, for a defect density of 20%, one iteration of CAEN-BIST achieves an average recovery of 79.5% and four iterations achieves an average recovery of 97.8%. If the defect density of CAEN exceeds expectations, CAEN-BIST will remain an effective test strategy.

CAEN-BIST also scales with the size of the nanoFabric. As shown in Table 2, the average recovery remains greater than 99% for large fabric sizes. The effectiveness of CAEN-BIST appears to be independent of the number

Table 2. Average recovery for four iterations of CAEN-BIST as fabric size increases for $k = 10$ and $d = 10\%$

No. of blocks (n)	Average recovery (%)
100	99.86
400	99.91
2,500	99.88
10,000	99.88
40,000	99.72
250,000	99.85
1,000,000	99.82
4,000,000	99.85
25,000,000	99.83
100,000,000	99.92

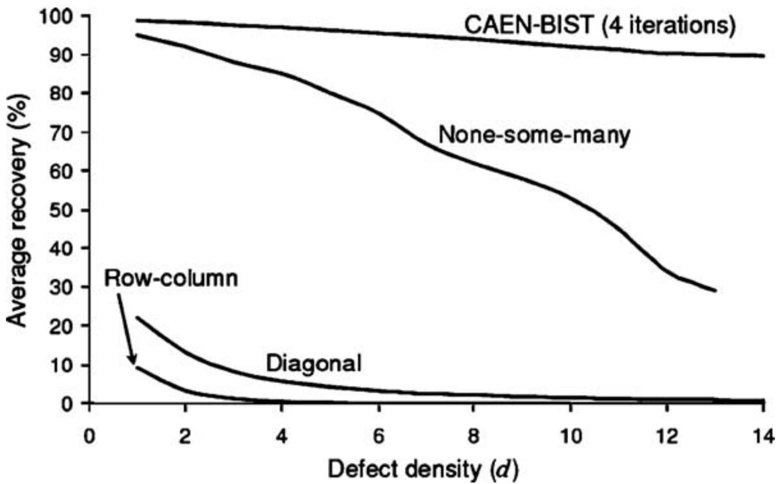


Fig. 22. Four iterations of CAEN-BIST achieves superior average recovery for increasing defect densities for $n = 1$ million and $k = 10$

of blocks in the nanoFabric. If the nanoFabric reaches or exceeds billions of nanoBlocks, we believe CAEN-BIST would remain an effective test algorithm.

CAEN-BIST achieves higher levels of average recovery than the other algorithms discussed in Sect. 3.4. As shown in Fig. 22, the row-column, diagonal, and the none-some-many algorithms do not achieve comparable levels of average recovery.

For the aforementioned experiments, faults are injected at the rate of $d\%$ using a random distribution. However, another set of experiments is performed to determine the effects of clustered faults within the nanoFabric. The

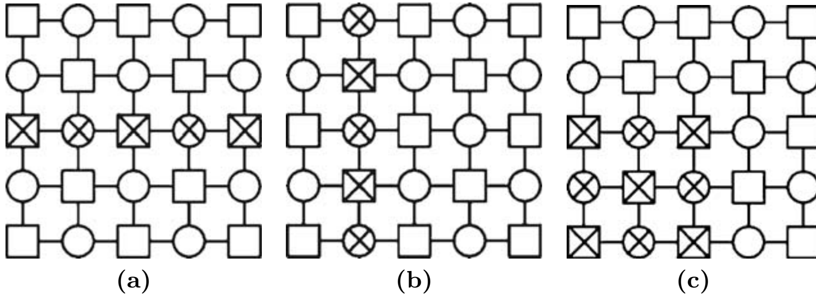


Fig. 23. Examples of clustered faults: (**a**) faulty row, (**b**) faulty column, and (**c**) faulty group

location, size, and type of clusters are chosen randomly. Examples of clustered faults are shown in Fig. 23. Clustered faults include faulty rows, faulty columns, or large groups of faulty nanoBlocks in one location. Fault clustering has a significant effect on the average recovery. For $d = 10\%$, $n = 1$ million, and $k = 10$, after one iteration of CAEN-BIST, the average recovery is only 12.3%. However, after testing from every corner of the fabric, the average recovery is greater than 98%. Therefore, if fault clustering becomes likely for CAEN, the repeated iterations of CAEN-BIST will be absolutely necessary to achieve a high level of recovery.

5 Implementation Challenges

There are several implementation challenges that must be addressed for CAEN-BIST to be effective. First, the time required for the nanoFabric to be configured is a major factor in this algorithm. Also, since the results of tests are stored internally, the defect map must be downloaded after each iteration of CAEN-BIST, which may also require a significant amount of time. If this required time becomes substantial, testing will become a major bottleneck in the manufacturing process. The next challenge involves the storage of test results. Since each nanoBlock uses three bits to store results for its neighboring nanoBlocks, this may become significant for larger fabric sizes. Memory storage becomes even more substantial when there are increased iterations of CAEN-BIST since intermediate defect maps must also be stored. Finally, CAEN-BIST creates a defect map that diagnoses each nanoBlock as defective or defect-free. However, the ideal defect map would have information regarding the condition of each wire and switch within a nanoBlock. Therefore, diagnostic resolution needs to be increased in order to provide the most ideal recovery for the nanoFabric.

6 Summary

We have developed a test and diagnosis strategy called CAEN-BIST that diagnoses faulty blocks in the nanoFabric. Our CAEN-BIST approach exploits the reconfigurability of the nanoFabric by configuring nanoBlocks as testers for their neighboring nanoBlocks. A behavioral model of the nanoFabric was created to simulate the behavior of the architecture and to provide an environment for different test algorithms. This environment allowed us to empirically show that an algorithm such as CAEN-BIST is viable for achieving accurate diagnostic results. CAEN-BIST achieves 100% coverage for all single stuck-line, connection, and bridge faults and average recovery of 92% for defect densities up to 10%. Moreover, diagnostic accuracy is improved significantly when the algorithm is performed on all four corners of the nanoFabric with the additional cost of storing the intermediate defect map. CAEN-BIST approaches 100% recovery for increased iterations and scales well with the size of the nanoFabric and defect density.

CAEN-BIST has provided promising results for diagnosis of the nanoFabric. The efficient creation of a defect map provides stepping stones toward defect tolerance in CAEN-based systems and an outlook toward the utilization of other defective circuit fabrics.

References

1. Y. Cui et al., "Diameter-controlled Synthesis of Single Crystal Silicon Nanowires," *Applied Physics Letters*, vol. 78, pp. 2214–2216, 2001.
2. T. Kamins et al., "Chemical-vapor Deposition of Si Nanowires Nucleated by $TiSi_2$ Islands on Si," *Applied Physics Letters*, vol. 76, 2000.
3. J. Mbindyo et al., "DNA-directed Assembly of Gold Nanowires on Complementary Surfaces," *Advanced Materials*, vol. 13, pp. 249–254, 2001.
4. D. J. Pena et al., "Electrochemical Synthesis of Multi-Material Nanowires as Building Blocks for Functional Nanostructures," in *MRS Symposium Proceedings*, vol. 636, 2001.
5. R. Service, "Assembling Nanocircuits from the Bottom Up," *Science*, vol. 293, 2001.
6. H. Soh et al., "Integrated Nanotube Circuits: Controlled Growth and Ohmic Contacting of Single-walled Carbon Nanotubes," *Applied Physics Letters*, vol. 75, 1999.
7. E. Winfree et al., "Design and Self-Assembly of Two-Dimensional DNA Crystals," *Nature*, vol. 394, pp. 539–544, 1998.
8. Y. Xia et al., "Unconventional Methods for Fabricating and Patterning Nanostructures," *Chemical Review*, 1999.
9. S. C. Goldstein and M. Budiu, "NanoFabrics: Spatial Computing Using Molecular Electronics," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp. 178–191, 2001.
10. S. C. Goldstein and D. Rosewater, "Digital Logic Using Molecular Electronics," in *Proceedings of the IEEE International Solid State Circuits Conference*, pp. 204–205, 2002.

11. M. Butts, A. DeHon, and S. C. Goldstein, "Molecular Electronics: Devices, Systems and Tools for Gigagate, Gigabit Chip," in *Proceedings of International Conference on Computer Aided Design*, pp. 440–443, 2002.
12. M. R. Stan et al., "Molecular Electronics: From Devices and Interconnect to Circuits and Architecture," in *Proceedings of the IEEE*, vol. 91, pp. 1940–1957, November 2003.
13. M. Mishra and S. C. Goldstein, "Defect Tolerance at the End of the Roadmap," in *Proceedings of International Test Conference*, 2003.
14. M. A. Reed and T. Lee, ed., *Molecular Electronics*, ch. 13. American Scientific Publishers, 2003.
15. M. Mishra and S. C. Goldstein, "Scalable Defect Tolerance for Molecular Electronics," in *Proceedings of International Symposium on High-Performance Architecture*, Feb. 2002.
16. Y. Luo et al., "Two-Dimensional Molecular Electronics Circuits," *CHEMPHYSCHEM*, vol. 3, pp. 519–525, 2002.
17. J. R. Heath, et al., "A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology," *Science*, vol. 280, pp. 1716–1721, June 1998.
18. R. Rajsuman, "Design and Test of Large Embedded Memories: An Overview," in *IEEE Design and Test of Computers*, vol. 18, pp. 16–27, May-June 2001.
19. S. K. Sinha et al., "Tunable Fault Tolerance for Runtime Reconfigurable Architectures," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 185–192, April 2000.
20. C. Stroud, et al., "Built-In Self-Test of Logic Blocks in FPGAs (Finally, A Free Lunch: BIST Without Overhead!)," in *Proceedings of IEEE VLSI Test Symposium*, pp. 387–392, 1996.
21. C. Stroud, E. Lee, and M. Abramovici, "BIST-based Diagnostics for FPGA Logic Blocks," in *Proceedings of IEEE International Test Conference*, pp. 539–547, 1997.
22. C. Metra et al., "Novel Technique for Testing FPGAs," in *Proceedings of Design, Automation and Test in Europe*, pp. 89–94, February 1998.
23. S. J. Wang and T. M. Tsai, "Test and diagnosis of faulty logic blocks in FPGAs," in *IEE Proceedings of Computers and Digital Techniques*, vol. 146, pp. 100–106, March 1999.
24. F. Toth, "Get the EasyPath Solution," *Xcell Journal*, Summer 2003.
25. B. Culbertson et al., "Defect Tolerance on the Teramac Custom Computer," in *Proceedings of 5th IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 140–147, 1997.
26. J. G. Brown and R. D. Blanton, "CAEN-BIST: Testing the NanoFabric," in *International Test Conference*, pp. 462–471, Oct. 2004.
27. Z. Wang and K. Chakrabarty, "Built-In Self-Test of Molecular Electronics-Based Nanofabrics," in *Proceedings of European Test Symposium*, pp. 168–173, May 2005.
28. M. M. Ziegler and M. R. Stan, "A Case for CMOS/Nano Co-design," in *Proceedings of the International Conference on Computer-Aided Design*, November 2002.
29. M. M. Ziegler and M. R. Stan, "The CMOS/Nano Interface from a Circuits Perspective," in *Proceedings of the International Symposium on Circuits and Systems*, May 2003.
30. C. P. Collier et al., "Electronically Configurable Molecular-Based Logic Gates," *Science*, vol. 285, July 1999.

31. S. Goldstein et al., "Reconfigurable Computing and Electronic Nanotechnology," in *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pp. 132–142, 2003.
32. M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design.* Piscataway, NJ: IEEE Press, 1990.
33. S. Venkataraman and S. B. Drummonds, "Poirot: Applications of a Logic Fault Diagnosis Tool," in *IEEE Design and Test of Computers*, vol. 18, January 2001.
34. J. E. Smith, "Detection of Faults in Programmable Logic Arrays," *IEEE Transactions on Computers*, vol. C-28, no. 11, pp. 845–853, 1979.
35. M. M. Ligthart and R. J. Stans, "A Fault Model for PLAs," *IEEE Transactions on Computer-aided Design*, vol. 10, no. 2, pp. 265–270, 1991.
36. L. Durbek and N. J. Macias, "Defect Tolerant, Fine-Grained Parallel Testing of a Cell Matrix," in *Proceedings of SPIE ITCom*, vol. 4867, 2002.
37. T. Rueckes et al., "Carbon Nanotube-Based Nonvolatile Random Access Memory for Molecular Computing," *Science*, vol. 289, pp. 94–97, 2000.

Chapter 5: Defect Tolerance in Crossbar Array Nano-Architectures

M.B. Tahoori

1 Introduction

Conventional lithography-based CMOS technology down-scaling faces serious challenges at both the device and system levels. Some of the challenges at the device level are manufacturing variability, sub-threshold leakage, power dissipation, increased circuit noise sensitivity, and cost/performance improvement. At the system level, some of the challenges are effective utilization of over-a-billion gates, system integration issues, power, and performance. While temporary solutions to these challenges will continue to be found, alternative devices need to be explored for possible replacement of or integration within CMOS. Some of the emerging candidates include *carbon nanotubes* (CNTs) [1–3], *silicon nanowires* (NWs) [4, 5], *resonant tunneling diodes* (RTDs) [6], *single electron transistors* [7], and *quantum-dot cellular automata* (QCA) [8].

Today's integrated circuits are designed using a top-down approach where lithography imposes a pattern. Unnecessary bulk material is then etched away to generate the desired structure. An alternative bottom-up approach, which avoids the sophisticated and expensive lithographic process, utilizes *self-assembly*, in which nanoscale devices can be self-assembled on a molecule-by-molecule basis. Examples of such devices are carbon nanotubes and silicon nanowires [2, 5, 9, 10]. Chemically self-assembled structures, as the building blocks for molecular-scale computing, are by their nature very regular and therefore well suited to the implementation of regular arrays similar to Field Programmable Gate Arrays (FPGAs). Reprogrammable nano-architectures are currently being investigated [11–14].

Self-assembly processes promise to considerably lower manufacturing costs, but at the expense of reduced control of the exact placement of these devices. Without fine-grained control, these devices will certainly exhibit higher defect rates. Moreover, nanofabrication process yields nanowires which are a few atoms long in the diameter. For instance, the contact area between nanowires contains only a few tens of atoms. With such small cross-section and contact areas, fragility of these devices is orders of magnitude more than

devices currently being fabricated using conventional lithography techniques. The need for defect and fault tolerance for this technology has been already stressed [9, 11, 13, 15].

The reconfigurability of this nanotechnology is well suited for the implementation of defect and fault tolerant mechanisms. For instance, after identifying defective resources in the chip using test and diagnosis, they can be bypassed by post-fabrication configuration. This approach is similar to conventional memory defect tolerant scheme which utilizes spare rows and columns. Nevertheless, the efficient implementation of this flow is not straightforward. Applying per-chip-customized configuration in the entire physical design flow causes serious problems for high-volume production. These include prohibitively large amount of information to be stored for the location of defect-free resources within the chip (defect map) and excessively increased post-fabrication efforts per chip (including test, diagnosis, and design).

In this chapter, we present a methodology to analyze the manufacturing yields of molecular crossbars in the presence of defects. We also present an alternative defect tolerant flow, the so-called *defect-unaware design flow*, in which most physical design steps (and also all higher level design steps) are unaware of the existence and the location of defects in the chip. The key idea is to identify universal defect-free subsets within the partially-defective chip to be used in the design flow. These subsets are called "universal" because these defect-free subsets are supposed to be identical for all manufactured chips with the same level of defect density. As a result, the size of the defect map as well as the amount of per-chip customized design time can be drastically reduced. Since this defect-tolerant flow is *application-independent*, i.e. it is not tailored for each particular mapped design, the amount of per chip design effort is minimized. Therefore, it is applicable for high-volume production. The only part of the physical design that needs to be aware of the location of the defects within the fabric is the *final mapping* in which the used resources in the universal subsets are mapped to the actual defect-free resources in the partially-defective fabric using defect map information.

The amount of excessive (spare) resources, also called *redundancy*, is a key factor that affects both the manufacturing yield and the cost of nano-scale systems. The amount of required redundancy is a function of manufacturing defect density, the required manufacturing yield, the size of the device, the architecture, and how the device will be used at the design steps. We try to analyze the amount of required redundancy in nano-architectures by examining two-dimensional (2D) programmable crossbar structures.

We also present and compare two different approaches to identify the maximum defect-free subsets and construct the compact defect map. One of the algorithms is an exact method whereas the other one is a heuristic approach for the same problem with faster run-time. These algorithms are also utilize in yield analysis method.

Since the molecular crossbar is the main building block in CNT-based crossbar array nano-architectures to implement logic, programmable

interconnects, and memory arrays, the main focus of the chapter is on the analysis of these crossbars for defect and fault tolerance. Nevertheless, we model and analyze the interconnect (nanowire) defects in our analysis by considering the crossbar defects and its surrounding nanowires.

The organization of this chapter is as follows. In Sect. 2 some background on CNT-based crossbar array nano-architectures is presented. The definitions and preliminaries are described in Sect. 3. The proposed design flow is described in Sect. 4. The yield metric and estimation method are presented in Sect. 5. The algorithms for yield estimation and mapping are explained in Sect. 6. The yield results as well as a comparison of the proposed heuristic algorithm vs. the exact method are discussed in Sect. 7. Finally, Sect. 8 concludes the chapter.

2 Programmable Crossbar Array Architectures

As explained in the introduction, bottom-up approaches used in the fabrication of nano-scale devices rely on self-assembly for defining feature size and may offer opportunities to drastically reduce the number of steps required to produce a circuit. However, the biggest impact in going from top-down designs to bottom-up is the inability to arbitrarily determine placement of devices or wires. Without fine control of the design, devices made from self-assembly techniques tend to be restricted to simple structures, such as two-terminal devices. Since these devices are usually non-restoring, one design challenge would be providing signal restoration between nanoscale logic stages.

Two dimensional (2D) crossbars are the building blocks of reconfigurable crossbar array architectures. In these architectures, two layers of orthogonal nanowires or carbon nanotubes form the crossbars [3, 16]. These nanowires or carbon nanotubes are aligned in parallel rows using bottom-up self assembly process. At each intersection, also called *crosspoint*, there is a programmable switch which is non-volatile. A one-time programmable switch consists of a monolayer of redox-active rotaxanes sandwiched between metal electrodes is demonstrated in [15]. In the *closed* state, the switch acts as a diode. It can also be irreversibly *opened* by applying a oxidizing voltage across the device. In these crossbars, configuration of crosspoint is performed by applying a higher voltage (programming voltage) to the intersecting nanowires. In other words, the same signal lines can be used as configuration circuitry. Unlike programmable crosspoints in conventional VLSI which are an order of magnitude larger than wire crossing area, crosspoint switches in this nanotechnology take the same area as of a wire crossing.

In [13], a chemically assembled electronic nanotechnology FPGA-like architecture called *NanoFabric* has been proposed. Nano logic arrays, also called *Nanoblocks*, implement a diode-resistor logic (DRL) since crosspoints act as programmable diodes. Since only AND and OR logic can be implemented by DRL, i.e. no inversion, inputs and their complements are given to nanoblocks

and the output function and its complement are generated. Signal restoration is performed by using a molecular latch at the output of crossbars.

DeHon has presented an array based nano-architecture using *Programmable Logic Arrays* (PLAs) [11, 12, 17]. The main building block, called the nano programmable logic array (nanoPLA), is built from a crossed set of N-type and P-type nanowires. This architecture allows inversion by using nanowire Field Effect Transistor (FET) devices as buffers. Logic functionality is achieved in the form of two stages of programmable crossbars. The first stage defines the logical product terms (pterms) by creating a wired-OR of appropriate inputs. The outputs of this wire-OR plane are restored through field-effect controlled nanowires that invert the outputs (thus creating the logical NOR of the selected input signals). These restored signals are then sent to the inputs of the next stage of programmable crosspoints. Each nanowire in this plane computes the wired-OR of one or more restored pterms. The outputs of the stage are then restored in the same manner as the first stage. The two stages together provide NOR-NOR logic (equivalent to a conventional PLA) The nanoPLA is programmed using lithographic-scale wires along with stochastically-coded nanowire addressing [17].

The *molecular CMOS* (CMOL) circuits proposed in [18] are designed using the same crossbar array structure as the nanoPLA design consisting of two levels of nanowires. The main difference with CMOL is how the CMOS/nanodevices are interfaced. Pins are distributed over the circuit in a square array, on top of the CMOS stack, to connect to either lower or upper nanowire levels. The nano crossbar is turned by some angle less than 90° relative to the CMOS pin array. By activating two pairs of perpendicular CMOS lines, two pins together with the two nanowires they contact are connected to the CMOS lines. Each nanodevice may be uniquely accessed using this approach. By angling the nanoarray, the nanowires do not need to be precisely aligned with each other and the underlying CMOS layer in order to be able to uniquely access a nanodevice. The most straightforward application of CMOL would be for memories (embedded or standalone). The CMOL circuits have also been proposed for building FPGA-like architectures for implementing random logic [19].

A CMOS-compatible crossbar memory array, called *NRAM*, has been proposed by Nantero Inc. [20]. In this architecture, everything but nanoelectromechanical switches are implemented in CMOS using conventional lithography processes (CMOS-compatible fabrication). The programmable switches are realized by a belt of carbon nanotubes (monolayer fabric of nanotubes). The same technology can be also used to implement programmable logic and interconnection network.

Defect tolerance of nano-architectures has been discussed in the literature. Hierarchical sparing is used in [11] to achieve defect tolerance. In [12], manufacturing yield of the proposed array-based architecture is analyzed based on the nanowire defect density, the yield of the stochastic decoder, and the stochastic buffering. In [21], the problem of logic mapping in a defective crossbar

is modeled by matching in a bipartite graph and a greedy matching algorithm with linear time complexity for logic mapping is presented. It is shown that when 20% of devices (i.e., crossbar diodes) were defective, only a 10% overhead in devices was needed to correctly configure the array around the defects.

3 Definitions

In CNT nanotechnology, the molecular crossbar is the main building block. An example of a 4×4 crossbar is shown in Fig. 1a. The crossbar consists of two sets of orthogonal nanowires. The horizontal nanowires are the inputs whereas the vertical nanowire are the outputs. There is a programmable switch at each intersection (*crosspoint*).

An $n \times n$ 2D crossbar can be represented by a *bipartite* graph $G = (U, V, E)$, as illustrated in Fig. 1b. A bipartite graph is a special graph where the set of vertices can be divided into two disjoint sets U and V such that no edge has both end-points in the same set. U represents the set of input nanowires and V represents the output nanowires; E, the set of edges, models the programmable switches in the crossbar. In a defect-free crossbar, there is a switch at the intersection of each input and output nanowire, i.e. $|E| = |U| \times |V|$. Therefore, a defect-free crossbar can be modeled by a *complete* bipartite graph.

In the presence of defects, some nanowires or switches become unusable and the corresponding bipartite graph is no longer complete. However, it might be possible to find a maximum defect-free subset of the crossbar which is complete. A subgraph of a bipartite graph is a *biclique* if this subgraph is a complete bipartite graph. The maximum complete subset of a bipartite graph is called the maximum biclique.

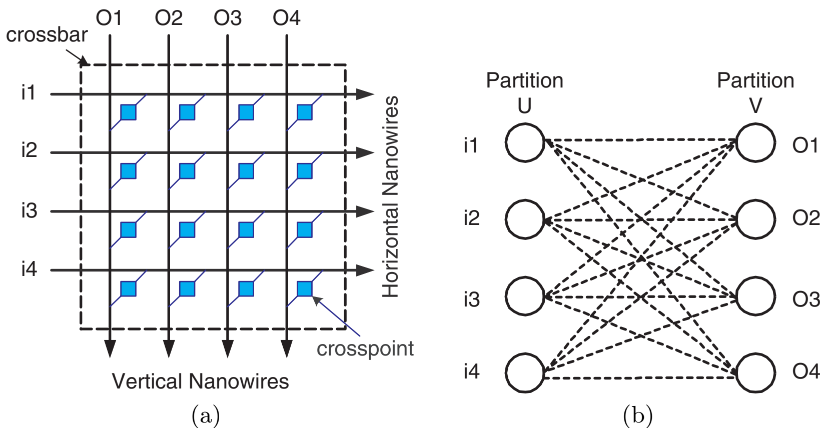


Fig. 1. (**a**) 4×4 2D nano-scale crossbar; (**b**) bipartite graph representation

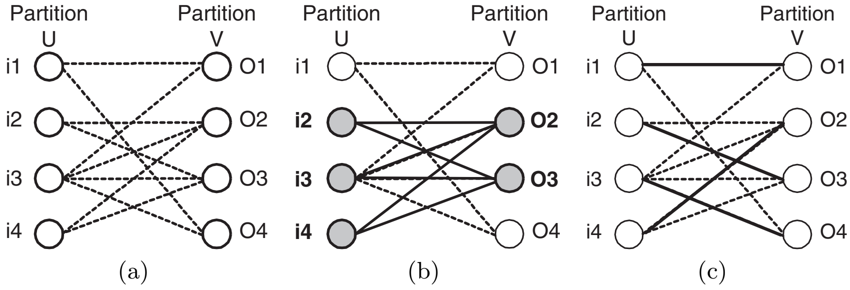


Fig. 2. (**a**) A bipartite graph $G(U, V, E)$; (**b**) biclique $(\{i_2, i_3, i_4\}, \{o_2, o_3\})$ in G; (**c**) a perfect matching in G

In some applications, such as logic mapping and some special routings, it is required to find a one-to-one mapping between input nanowires and output nanowires through crosspoints. A *matching* T is a set of edges such that no two edges share the same vertex. If an edge (v_i, v_j) is in the matching, then vertices v_i and v_j are said to be matched. A *perfect* matching of a graph is a matching such that all vertices are matched.

Figure 2 shows an example of a 4×4 bipartite graph. It also shows the maximum biclique in this graph. This maximum biclique is 3×2. Every node is the set $\{i_2, i_3, i_4\}$ is connectable to each and every node in $\{o_2, o_3\}$. A perfect matching of this bipartite graph is also shown.

4 Defect-Unaware Design Flow

Thorough testing and high-resolution diagnosis play major roles in the implementation of defect and fault tolerant systems. Test and diagnosis techniques for crossbar array architectures have been presented in [22–25], which exploit the reconfigurability of these nano-architectures to achieve a very high coverage test and high resolution diagnosis. These techniques are mainly categorized as *Built-in Self-test* (BIST) since test vector generation and output response analysis are performed on-chip. Such techniques are described in more details in the other chapters of this book, since test and diagnosis for crossbar array architectures are out of the scope of this chapter.

In the conventional adaptive defect tolerant flow, the existence and the location of defective elements are identified using test and diagnosis steps. After these steps, each individual resource in the crossbar array (e.g. nanowire, crosspoint, buffer) is precisely identified as defect-free (usable) or defective (unusable). The fault location information is stored in the so-called *defect map* which identifies the usability of the (programmable) elements of each manufactured chip. Defect tolerance is achieved by avoiding defective resources in the physical design flow using the defect map. Particularly, placement and routing phases of the physical design use the defect map in order to map the

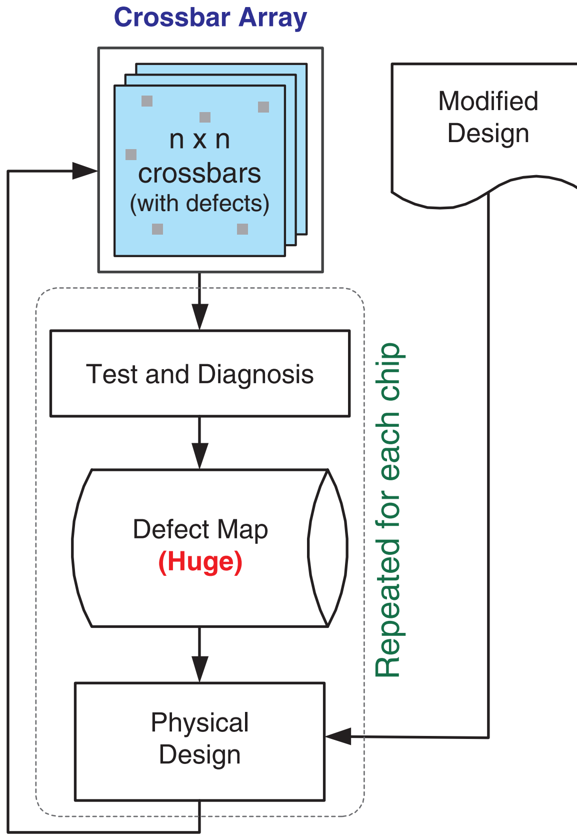


Fig. 3. Defect-aware design flow

design to the crossbar array by using only defect-free resources. This flow is shown in Fig. 3. We call this flow *defect-aware* or *application-dependent* since design phases use the defect map information. In addition to physical design phases, higher-level design steps, such as logic and architecture design, might need to be modified to incorporate defect map information. An example of an application-dependent defect tolerance flow for crossbar array nano-architectures is presented in [21].

However, there are several problems associated with this approach:

- First, the size of the defect map for the entire chip can be prohibitively large. Note that defect map stores the usability of each and every individual resources in the crossbar array and 10^{12} to 10^{14} devices are anticipated per chip at the nano-scale integration. Moreover, it would be impossible to store and use this defect map for online testing and fault tolerance purposes.

- Second, the entire design flow has to be customized per chip. This is because the defect map is unique for each manufactured chip (different chips have different defect maps). Since the design phases use defect map information, all defect-aware design phases have to be repeated for each manufactured chip. This adds the design time (∼hours) to the post-fabrication test time (∼seconds to minutes) per chip. Note that in today's design flow (i.e. without defect tolerance), the design effort is independent of the number of manufactured chips. However in this defect-aware flow, most design steps have to be performed in a per-chip basis.
- Third, per-chip design customization results in large and unacceptable parametric variations for a same design mapped into different chips. As the defect maps of different manufactured chips could be be considerably dissimilar and the outcome of the physical design is highly dependent on the defect map, the parametric characteristics (including performance and power) of a same design mapped into various chips might be completely different.
- Fourth, this defect-aware scheme results in a radical shift from conventional design methods such that all logic and physical design tools have to re-developed. The design steps need to be modified such that they can accept the defect map as one of the design inputs.

Due to these fundamental problems, this traditional approach cannot be used for high-volume production of nano-chips.

Most drawbacks of the defect-aware flow are due to the fact that this method is *application dependent*, i.e. defects are handled in a per-application basis. In contrast to the defect-aware design flow, we propose a *defect-unaware* design flow to tolerate defects in crossbar arrays. Our proposed defect tolerant approach is an *application independent* flow, which means that the defects are bypassed before mapping any particular application to the nano-architecture. In other words, defect tolerance is performed once and the same recovered array can be used for all applications mapped to the nano-architecture. In the proposed flow, almost all design steps, from high-level architecture design to the last step of the physical design, are unaware of the existence and the location of defects within the nano-architecture. In other words, all design steps work on a *design view* of the nano-chip which is defect-free. There is a *final mapping* phase, as the very last step of the physical design flow that makes the connection between the defect-free design view and the actual *physical view* of the nano-chip which contains the actual defects.

The key idea in this defect-unaware flow is to identify *universal* defect-free subsets of resources within the original partially-defective nano-chip. All design steps use these universal defect-free subsets as the physical implementation devices (design view of the nano-chip). Hence, conventional architectural, logic, and physical design flows and tools can be reused. These defect-free subsets are called "universal" because the size of these subsets is identical for all nano-chips fabricated in the same process environment (similar defect

densities). Also, these universal defect-free subsets remain unchanged for different applications mapped into the same nano-chip, making this approach application-independent. An additional step, after placement and routing, is required to map the used resources within the universal subset (design view) into the actual resources within the original device (physical view). This is the only defect-aware step in the entire design flow which has to be performed for each manufactured nano-chip. The two major tasks of defect-tolerance is this defect-unaware flow is to

1. Identify and extract the defect-free universal subsets from partially-defective nano-chip
2. Perform the final mapping phase, which consists of mapping the used resources in the design view to the actual resources in the physical view.

For molecular crossbars, the goal is to identify defect-free $k \times k$ crossbars within the original partially-defective $n \times n$ crossbars (task 1). These defect-free subsets of the crossbars are complete, which means that every k input nanowire in the defect-free subset is connectable to every k output nanowire through a defect-free crosspoint. In general, the defect-free subset is smaller than the original crossbar ($k < n$). The final mapping step maps the used resources within $k \times k$ crossbars into the actual defect-free resources within the original $n \times n$ crossbars (task 2).

Figure 4 shows the defect-unaware physical design flow. Test and diagnosis steps identify the size and the location of defect-free $k \times k$ crossbars within

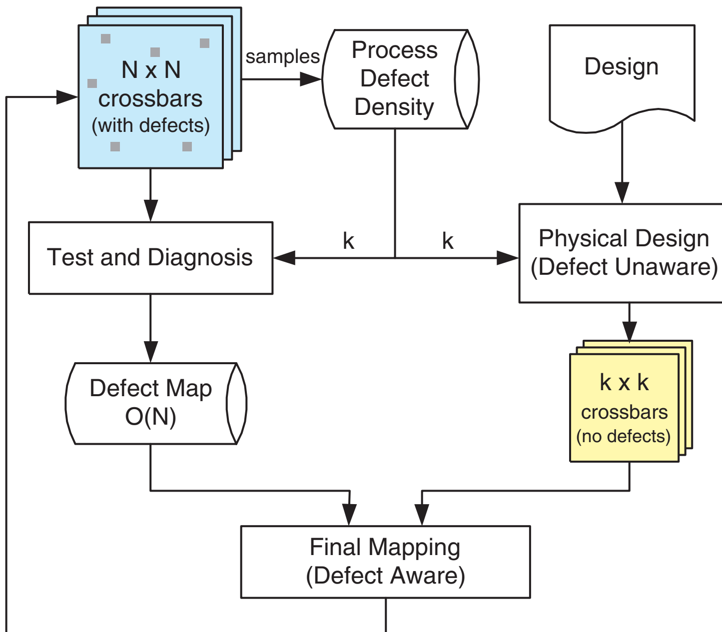


Fig. 4. Defect-unaware design flow

partially-defective fabricated $n \times n$ crossbars. The size of the maximum defect-free crossbar, k, can be estimated using a yield analysis method (presented in Sect. 5) based on a sample of fabricated devices and this value will be used for all chips manufactured in the same process environment. All crossbars with a defect-free subset not smaller than $k \times k$ are considered as "good" (pass) devices, otherwise they are marked as "bad" (fail) devices. The information regarding the location of defect-free subsets of crossbars is stored in a compact defect map. Compared to the defect map in the defect-aware flow which stores the location of each defective switch within each crossbar, this one stores only the position of the defect-free $k \times k$ crossbar within the original crossbar.

During the physical design, the original design will be mapped (placed and routed) into an array of $k \times k$ crossbars. A final defect-aware mapping step is required to re-map the used resources within $k \times k$ crossbars into the actual defect-free resources within partially-defective $n \times n$ crossbars using the defect map information.

Note that the value of k cannot be determined in a per-chip based otherwise some design steps will have to be customized for each chip. Therefore, the value of k is estimated based on the defect density level of the fabrication process. This is the critical point to make the design efforts independent of the number of manufactured chips. Moreover, the value of k is fixed for all manufactured parts in the same fabrication environment. For example, if the value of k is set to 16, then, all crossbars for which $k \geqslant 16$ are considered as good (passing) devices, and those with $k < 16$ are considered as failing devices. It needs to be mentioned that the desirable value of k is set such that a large (and acceptable) fraction of manufactured devices passes the test. The yield analysis method required for this step is presented in the next section.

Another advantage of this approach is the reduction in the size of the defect map. In the conventional defect-aware flow, the size of the defect map is $O(n^2)$ for an $n \times n$ crossbar. This is because one bit information is required for each switch to specify whether it is defect-free or not. In the proposed defect-unaware flow, it is only needed to identify the location of $k \times k$ defect-free crossbar within the partially-defective $n \times n$ crossbar. Therefore, only two binary vectors of size n, the vertical and horizontal vectors, are needed to be stored. Each bit position in the vertical (horizontal) vector corresponds to a row (column) in the original crossbar and specifies whether this row (column) participates in the defect-free biclique ($k \times k$ crossbar). These two $O(n)$ vectors are sufficient to precisely locate the defect-free subset within the original crossbar. Consequently, the size of the defect map is reduced from $O(n^2)$ to $O(n)$.

Figure 5 shows and compares defect maps in defect-aware and defect-unaware flows. In Fig. 5 (a), the bipartite graph model of a defective 6×6 crossbar is shown. This crossbar is defective since the corresponding bipartite graph is not complete. Each missing edge corresponds to a defective switch. The complete defect map, as used in the original defect-aware flow, is shown in Fig. 5 (b) in which a "1"("0") entry represents a usable (unusable) switch.

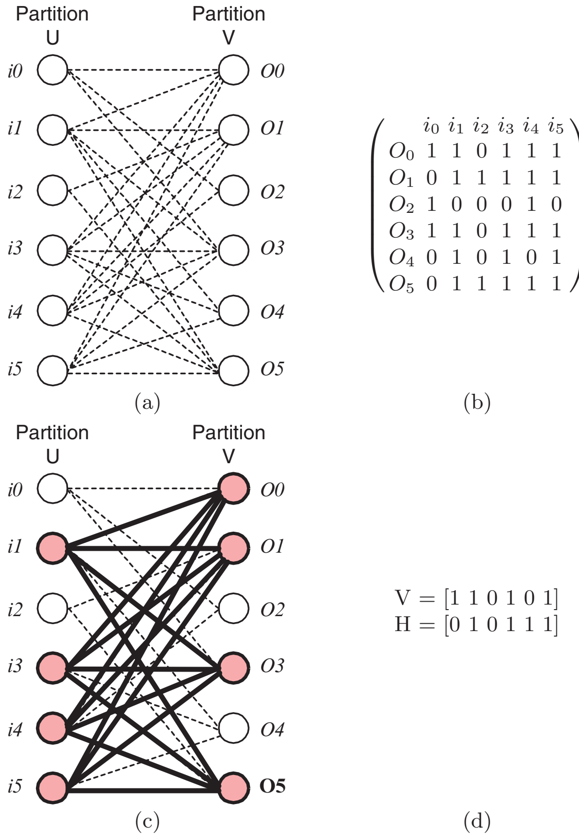


Fig. 5. (**a**) 6×6 Crossbar (**b**) complete $O(n^2)$ defect map (**c**) defect-free 4×4 subset of crossbar (**d**) reduced defect map: two vectors of $O(n)$

The defect-free 4×4 subset of the crossbar is shown in Fig. 5 (c). The reduced defect map containing only of two vectors of size n is shown in Fig. 5 (d). Using these two vectors, the defect-free 4×4 subset, corresponding to the complete 4×4 subgraph can be uniquely identified. The horizontal vector identifies the nodes in the input partition (U) participating in the defect-free subset whereas the vertical vector corresponds to the output partition (V).

The nano-architecture can be viewed as an array of crossbars. The *global view* of this architecture considers only the interconnection of crossbars whereas the *local view* deals with the individual resources within each crossbar. The main focus of this chapter is the local view of the nano-architecture in which the structure of a crossbar is fully analyzed. Considering the local view (used in detailed placement and routing), the problem of performance variation due to detailed mapping, placement, and routing can be solved with the proposed design flow. Nevertheless, we discuss some issues related to defect tolerance in the global view of the nano-architecture in Sect. 7.3.

In the next section, we present a yield metric based on this design flow in which we explain how to set the value of k such that a large fraction (high yield) of manufactured crossbars contain defect-free subsets of minimum size k. In Sect. 6, two algorithms for identifying and extracting the defect-free subset within the fabricated crossbar, based on the information obtained from test and diagnosis steps, are presented. These algorithms can also be used to estimate the value of k from process defect density information using a sample of manufactured chips. The reduced $O(n)$ defect map will be generated as one of the outputs of these algorithms. The generated defect map is also used in the final mapping phase. Since this step has to be performed for each manufactured chip, its run time has to be minimized. Therefore, we propose a fast greedy algorithm for this purpose.

5 Yield Metric

In this section we present a yield analysis framework which is compatible with the defect tolerant flow presented in the previous section. Specifically, this framework is used to analyze the yield and the expected size of the defect-free subsets within the partially-defective crossbars.

We have addressed the matching problem in defective crossbars in [26]. In matching problem, the goal is to find a routing between m distinct input signals to m output signals through an $n \times n$ crossbar using m non-incident defect-free switches ($m \leqslant n$). However, defect-free matching which is sufficient for logic mapping, is not adequate for signal routing in the general form since selective connectivity is lost. From algorithmic point of view, there is an exact solution for defect-free matching problem based network flow algorithm with polynomial time complexity, $O(n^{\frac{5}{2}})$, [26]. The goal in this chapter is to find a defect-free $k \times k$ crossbar in a partially-defective $n \times n$ ($k < n$) crossbar at a specified defect density level. Based on the design flow presented in Sect. 4, the following yield metric is defined:

Yield metric $Y^d_{n,k}$. The probability of finding a biclique (defect-free crossbar) of size $k \times k$ in an $n \times n$ crossbar when the defect density is d.

Based on this yield metric $Y^d_{n,k}$, we can identify the minimum size of a fabricated (partially defective) crossbar, $n \times n$, such that a defect-free crossbar of size $k \times k$ can always be found with the yield of y when the defect density is d. In other words, for desirable values of k and y, and given value of d, the minimum value for n can be obtained such that $Y^d_{n-1,k} < y$ but $Y^d_{n,k} \geqslant y$.

5.1 Defect Distribution Models

The distributions of defects in the crossbar can be modeled using uniform or non-uniform probability distributions. We consider both *clustered* and *unclustered* defects distribution models. In the unclustered model, as the simplest model, the defect probability distribution over the crossbar is uniform. In other

words, the probability of defect at each location (e.g. crosspoint or nanowire) is p, the defect density, and independent of the locations of other defects in the crossbar. Figure 6 describes the unclustered defect injection method.

In real cases, defects tend to cluster together and the distribution in not uniform [27]. In this model, the probability of defects in the regions near already existing defects is higher. We can consider this clustering effect by a model in which the defect probabilities in the regions near defect clusters are higher than other regions. This probability is also inversely proportional to the distance from the existing defect clusters. Formally, if N_d defects already exists (are injected) in the crossbar, the defect probability at location i, P_i, is calculated as

$$P_i \propto \sum_{j=1}^{N_d} \left(\frac{1}{d_{ij}}\right)^{\alpha}$$

where d_{ij} is the distance between switch (nanowire) i and j, and α is the clustering parameter. Note that a larger value of α corresponds to more clustering.

In defect injection based on this clustering model, if the defect density is d, and there are N resources (crosspoints and nanowires) as candidates for defect injection, total of $N.d$ defects will be injected. The algorithm for defect injection based on the clustering defect distribution model is given in Fig. 7. Figure 8 shows an example of defect injection patterns in a 32×32

Unclustered Defect Injection(N, d)
  $N_d \leftarrow 0$
  while $N_d < N.d$ **do**
    $i \leftarrow \text{random}(0, N)$
    $j \leftarrow \text{random}(0, N)$
    if location (i, j) is not defective **then**
      Inject defect at location (i, j)
      $N_d \leftarrow N_d + 1$

Fig. 6. Unclustered defect injection algorithm

Clustering Defect Injection(N, d, α)
  $N_d \leftarrow 1$
  Inject the first defect randomly
  while $N_d < N.d$ **do**
    for each location i **do** (defects already injected in locations j)
      $P_i \leftarrow \sum_{j=1}^{N_d} \left(\frac{1}{d_{ij}}\right)^{\alpha}$
    Normalize P_i s.t. $\sum_{i=1}^{N} P_i = 1$
    $r \leftarrow \text{random}(0,1)$
    choose k s.t. $\sum_{i=1}^{k} P_i < r \leqslant \sum_{i=1}^{k+1} P_i$
    Inject defect at location k
    $N_d \leftarrow N_d + 1$

Fig. 7. Clustering defect injection algorithm

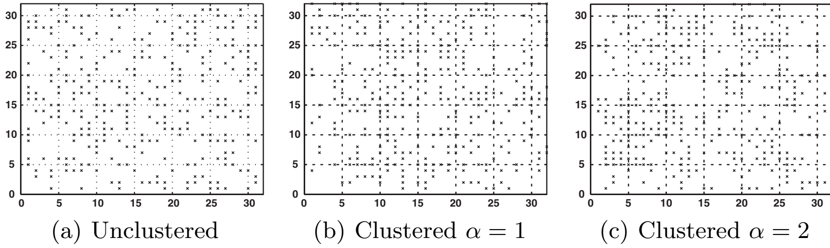


Fig. 8. Defect injection in a 32 × 32 crossbar using different defect distribution models

crossbar with 30% defect density using unclustered and clustered models. As can clearly be seen in this figure, defects are more clustered with $\alpha = 2$ compared to unclustered or $\alpha = 1$.

5.2 Fault Injection Method for Yield Estimation

We use a fault injection platform to obtain the the value of the yield metric $Y^d_{n,k}$. Once the locations of the defects in the crossbar are identified based on defect density and distribution model, faults are introduced in the complete bipartite graph $G = (U, V, E)$ representing the defect-free crossbar resulting in a new bipartite graph $G' = (U', V', E')$ corresponding to the defective crossbar. The problem of finding a defect-free crossbar (of size $k \times k$) within a partially-defective one can be expressed as finding a biclique of the desirable size (k) in bipartite graph G'.

In this work we consider the faults within the crossbar as well as the interconnect faults surrounding the crossbar (outside crossbar). By considering the interconnect (nanowire) faults surrounding each crossbars, faults in the entire crossbar array can be modeled.

Switch stuck-open faults (within crossbar). A switch stuck-open fault corresponds to a missing switch at the crosspoint. Hence, the faulty bipartite graph G' can be obtained by simply deleting from G the edges corresponding to the faulty switches.

Switch stuck-closed faults (within crossbar). If there is a stuck-closed fault on a switch corresponding to edge (u, v), the horizontal nanowire $u \in U$ and the vertical nanowire $v \in V$ are shorted together and both become unusable. Therefore, G' can be obtained by removing the nodes u and v from G along with all edges incident to these nodes ($U' = U - \{u\}, V' = V - \{v\}$). In this case, some fault-free switchings connected to u or v become unusable as well. Note that this situation is different from the effect of switch stuck-closed faults when defect-free matching is required [26]. We can relax this constraint by just removing one node (either u or v) along with all edges incident to that node. This reduces the number of defect-free switches that become unusable

due to a stuck-closed fault. This relaxed case is similar to the situation for nanowire open faults described below.

Nanowire open fault (within and outside crossbar). In some architectures the precharge/evaluation logic is used at the output stage of molecular crossbars [12]. As a result, broken nanowires are of no use at all and cannot be used to carry signals. In this case, all switches connected to a nanowire with an open fault become unusable. So G' is obtained by deleting the faulty node (broken nanowire) from G.

Nanowire bridging fault (within and outside crossbar). In case of a nanowire bridging fault, two (or more) nanowires are shorted together, and both (all) of them become unusable. When constructing the faulty bipartite graph G', all nodes corresponding to the shorted nanowires and all incident edges to these nodes are removed from the original graph G.

Figure 9 shows the graph model of a 4×4 crossbar in the presence of some faults. As can be seen from this example, short faults (nanowires and switches) have clearly more severe effect on the usability of crossbars than switch open faults.

The fault injection method is as follows. Depending on the defect density d and the particular fault model, there are *FP fault patterns* each corresponds to a different G'. For example, in the case of switch stuck-open faults, there are $\binom{n^2}{\lceil n^2 d \rceil}$ possible fault patterns. Since the number of possible fault patterns can be quite large, we only consider a statistical sample of them. For each fault pattern, one defective crossbar is generated with defect density d, fault models, and defect distribution model. The yield metric $Y^d_{n,k}$ is estimated as the ratio of the number of simulated fault patterns with a minimum biclique of size $k \times k$ over the total number of simulated fault patterns. The fault injection approach for yield estimation is shown in Fig. 10. In order to improve the accuracy of this fault injection method, it can be implemented using the monte-carlo simulation in which the number of simulated fault patterns can be adjusted such that the variance of the estimated yield metric falls within predefined ranges.

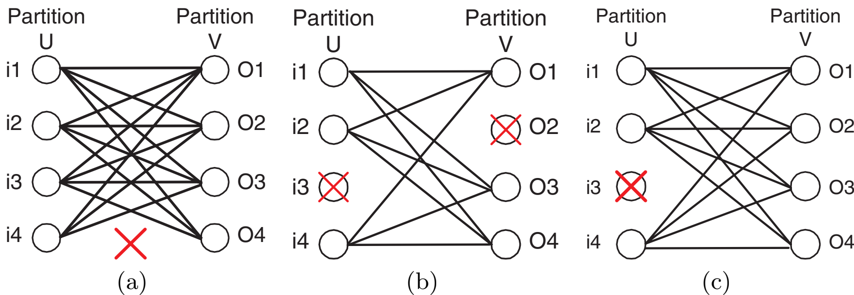


Fig. 9. (a) Switch Open $(i4, O4)$ **(b)** Switch Short $(i3, O2)$ **(c)** Nanowire $i3$ Open

Fault Injection Approach for Yield Estimation(n, k, d)
 $N_{success} \leftarrow 0$
 for each fault pattern i, $1 \leqslant i \leqslant FP$, **do**
  Inject $n^2.d$ defects in crossbar G based on fault models and defect distribution model to obtain G'
  if Biclique$(G') \geqslant k \times k$ **then**
   $N_{success} \leftarrow N_{success} + 1$
 $Y^d_{n,k} = \frac{N_{success}}{FP}$

Fig. 10. Fault injection method

6 Biclique Algorithms

In this section we present two different approaches to find the maximum defect-free crossbar within a partially defective crossbar. The first approach is a variation of the exact method in which the optimal solution is guaranteed at the expense of exponential run time. The second one is a greedy heuristic method in which the optimal solution is not guaranteed whereas its run time complexity is $O(n \log n)$. These algorithms can be used for the yield analysis step as explained in Sect. 5 as well as the extraction of the universal defect-free subsets. They are also the basis for the final mapping phase and the generation of the reduced defect map (Sect. 4).

6.1 Recursive Biclique Algorithm

Given the location of defect-free switches and nanowires in the crossbar, obtained from test and diagnosis procedures, the graph model of the defective crossbar can be formed. The goal is to find (and locate) the maximum defect-free $k \times k$ crossbar within the original crossbar. This problem corresponds to finding the maximum biclique in a bipartite graph.

Finding the maximum biclique in a bipartite graph is an NP-complete problem [28]. However, we solve a *decision* version of this problem which is less complex compared to the original *optimization* version. Instead of finding the maximum biclique in $G(U, V, E)$, we solve the following decision (Yes/No) problem: "*Does* $G(U, V, E)$ *have a biclique of size* $k_1 \times k_2$?"

Note that in the above problem $k_1 \times k_2$ is not necessarily the size of the maximum biclique of G. Using the fault injection approach presented in Sect. 5.2, by setting $k_1 = k_2 = k$, the desirable probability of $Y^d_{n,k}$ can be obtained. The recursive algorithm shown in Fig. 11 is developed for solving the decision problem outlined above.

Proof of Correctness

Since this algorithm is recursive, we provide a proof of the algorithm based on induction hypothesis. The induction basis is checked in lines 2–5. Bicliques of

```
1 Function HasBiclique(G(U, V, E), k1, k2)
2     if k1 ⩽ 0 ∨ k2 ⩽ 0 then
3       return TRUE
4     if |U| ⩽ k1 ∨ |V| ⩽ k2 then
5       return FALSE
6     U1 ← {u|u ∈ U, d(u) < k2}
7     V1 ← {v|v ∈ V, d(v) < k1}
8     U2 ← {u'|u' ∈ U, d(u') = |V|}
9     V2 ← {v'|v' ∈ V, d(v') = |U|}
10    U ← U − (U1 ∪ U2); V ← V − (V1 ∪ V2)
11    k1 ← k1 − |U2|; k2 ← k2 − |V2|
12    u ←node with minimum degree in U
13    V' ←{nodes connected to u in V}
14    return HasBiclique(G(U − {u}, V, E),k1,k2) ∨
             HasBiclique(G(U − {u}, V', E),k1 − 1,k2)
```

Fig. 11. Recursive biclique algorithm

sizes $0 \times k_2$ and $k_1 \times 0$ can be always found (lines 2–3), so the function returns "true". Similarly, the size of any biclique cannot be larger than the size of the original graph (lines 4–5), and as a result, the return value is "false".

If there is a node in U whose degree in less than k_2, it cannot participate in the biclique and hence, can be excluded from the graph during the search. The set of these nodes are represented by $U1$. Moreover, if there is a complete node (i.e. connected to all nodes in V), it can be always included in the biclique. Hence, we can remove this node from U and decrease k_1 by one. The set of such nodes is represented by $U2$. The similar cases for the nodes in the second partition, V, are also considered (lines 6–11).

The induction step is performed as follows. Let $|U|+|V| = n$. The induction hypothesis is that *HasBiclique* function works correctly for any graph of node size smaller than n. Consider an arbitrary node $u \in U$. This node is either included in the biclique or not.

- If it is not included, we need to find a biclique of size $k_1 \times k_2$ in G($U - \{u\}, V, E$). Since the node size of this graph is $n - 1$, based on induction hypothesis, the *HasBiclique* function gives a correct answer for this graph.
- If this node is included in the biclique, the nodes of V that can participate in the biclique are those connected to u. So, if we call this set V', we should be able to find a biclique of size $k_1 - 1 \times k_2$ in G($U - \{u\}, V', E$). In order to reduce the number of recursions, u is chosen as the node with the minimum degree. This way, the size of V', which directly affects the search space, is minimized. In other words, if the answer is supposed to be "false", we will be able to find it out with the minimum number of iterations. Since $|V'| \leqslant |V|$, the node size of G($U - \{u\}, V', E$) is smaller than or equal to $n - 1$. Again, based on the induction hypothesis, the *HasBiclique* function will return a correct answer for this graph.

There is a biclique of size $k_1 \times k_2$ in $\mathrm{G}(U, V, E)$ if at least one of the above cases is true (line 14). Therefore, based on the induction hypothesis, the function works correctly on graphs of node size n, and the proof is complete.

Run Time Analysis

The worst case run time complexity of this recursive algorithm is exponential. Let $|U| + |V| = n$ and $f(n)$ be the run time of this algorithm on $\mathrm{G}(U, V, E)$. In the worst case $|V'| = |V|$ and hence, $f(n) = f(n-1) + f(n-1) + O(1) = 2f(n-1) + O(1)$. Also, $f(1) = O(1)$. This makes $f(n) = O(2^n)$ for the worst case. However, since the removed node is chosen as the node with the minimum degree, $|V'| < |V|$, otherwise the algorithm terminates in the next iteration (lines 8–9 followed by lines 2–3). In general, since this is an exact algorithm (if there is a solution, it guarantees to find that solution), its average case complexity is still exponential.

6.2 Greedy Mapping Algorithm

The recursive algorithm presented above is an exact algorithm, however its worst case complexity is exponential. Here we present a heuristic greedy algorithm for finding the maximum biclique. This algorithm is not exact, i.e. its solution is not guaranteed to be optimum, however, its run time is completely tractable. We will compare the accuracy and run time of this greedy algorithm vs. the exact algorithm in Sect. 7.

Our approach here is to convert this problem to the problem of finding the maximum independent set in the *complement* graph. The outline of the proposed greedy algorithm is shown in Fig. 12. The complement of a graph G is a graph $\overline{G}$ with the same set of vertices such that two vertices of $\overline{G}$ are adjacent if and only if they are not adjacent in G. An *independent set* S in a graph G is a subset of nodes that are disconnected, i.e. there are no edges between any two nodes in an independent set: $\forall u, v \in S,\ (u, v) \notin E(G)$. The maximum independent set is an independent set with the maximum number of nodes.

Even in the presence of defects (defect density $< 30\%$), the corresponding bipartite graph model of the crossbar is still dense, i.e. $|E| = O(n^2)$. Consequently, the complement graph would be sparse and therefore, we can efficiently use a heuristic approach for finding the maximum independence set in the complement graph. This is the main motivation behind converting the maximum biclique problem into the maximum independent set problem in the complement graph.

The proposed heuristic works as follows. Nodes with zero *degree* (i.e. the number of edges connected to that node) can always participate in the independent set. Our objective is to remove some nodes along with their incident edges from the complement graph such that more nodes with zero degree can be found in the reduced graph. Our heuristic is based on removing the nodes

```
1 Function Biclique(G(U,V,E))
2     Obtain Ḡ(U,V,Ē), Ē = K_{|U|,|V|} − E
3     Find maximum independent set in Ḡ
4       Sort U based on d(u) in Ḡ (decreasing order)
5       Sort V based on d(v) in Ḡ (decreasing order)
6       U^b ← φ, V^b ← φ, flag ← TRUE
7       Repeat
8           U^b ← U^b ∪ {u|u ∈ U, d(u) = 0}, U ← U − U^b
9           V^b ← V^b ∪ {v|v ∈ V, d(v) = 0}, V ← V − V^b
10          if flag then
11            u ← node in U with maximum degree
12            U ← U − {u}
13            for each v' ∈ V such that (u,v') ∈ Ē do
14                d(v') ← d(v') − 1
15            Re-sort V accordingly
16          else
17            v ← node in V with maximum degree
18            V ← V − {v}
19            for each u' ∈ U such that (u',v) ∈ Ē do
20                d(u') ← d(u') − 1
21            Re-sort U accordingly
22          flag ← ¬flag
23      Until U = φ and V = φ
24    return U^b × V^b as the maximum biclique
```

Fig. 12. Greedy biclique algorithm

with the maximum degree since it is assumed that they will not participate in the maximum independent set. Deleting the nodes with the maximum degree allows us to remove a maximum number of edges with a minimum node removal. This increases the chance of finding a large independent set in the remaining nodes. Since the complement graph is sparse and nodes with the maximum degree in the reduced graph are removed, we hope that the set of independent nodes with zero degree in the reduced graph is very close to the maximum independent set.

First, the complement graph is formed (line 2). The nodes in each partition are sorted based on their degrees in the complement graph in the decreasing order (lines 4–5). At each iteration, nodes with zero degree are added to the solution list (lines 8–9). In this pseudo-code, the set of nodes from U and V participating in the maximum independent set are denoted as U^b and V^b, respectively.

Bicliques with square shapes are preferred, i.e. if the biclique obtained by this algorithm is $k_1 \times k_2$, it is preferred that k_1 is very close to k_2. The presented heuristic approach (removing nodes with the highest degrees) finds an independent set (biclique) with minimum node removal. In other words, this algorithm tries to maximize $k_1 + k_2$, the number of nodes in the biclique

(independent set). Since the number of edges (switches) in the biclique is $k_1.k_2$, the number of edges in the biclique is maximized when $k_1 \approx k_2$, for a fixed $k_1 + k_2$.

To obtain a maximum-edge biclique, our heuristic is to alternate between U and V when removing the nodes with the highest degrees. Hence, at every other iterations, nodes with the highest degree are removed from U (lines 10–15), whereas in the alternating iterations, nodes with the maximum degree are removed from V (lines 16–21). Since $|U| = |V|$, at the end of the execution of the algorithm $|U^b| = |V^b| \pm 1$. Consequently, the resulting biclique has a square shape and hence, contains a maximum number of edges.

Proof of Correctness

At each iteration of the repeat-until loop (lines 7–23), one node with the maximum degree will be removed either from U or V. Let's denote the set of nodes removed from U and V as U^r and V^r, respectively. The invariant of the algorithm can be defined as follows:

Invariant. *During the execution of the algorithm, $U^b \cup V^b$ forms an independent set of $\overline{G}$.*

Proof. At the beginning of the algorithm (line 6), U^b and V^b are initialized to the empty sets, and hence, the invariant holds. At each iteration of the repeat-until loop, nodes with zero degree from $U - U^r$ and $V - V^r$ are added to U^b and V^b, respectively. Let's denote the set of nodes removed from U by the end of ith iteration as U_i^r. We define V_i^r similarly. Since $U_i^r \subseteq U_{i+1}^r$ and $V_i^r \subseteq V_{i+1}^r$, $d(u)\ in\ \overline{G}(U - U_i^r, V - V_i^r, \overline{E}) \geqslant d(u)\ in\ \overline{G}(U - U_{i+1}^r, V - V_{i+1}^r, \overline{E}), u \in (U - U_{i+1}^r) \cup (V - V_{i+1}^r)$. Therefore, if $d(u) = 0$ in the ith iteration, it is independent from all nodes in $(U - U_i^r) \cup (V - V_i^r)$. Consequently, any node u' to be added to U^b or V^b in the $i + 1^{th}$ iteration is also independent from u. This proves the invariant.

Based on this invariant, at the end of the execution of the algorithm, $U^b \cup V^b$ forms an independent set in the complement graph, $\overline{G}$, and hence, a biclique in the original graph, G.

Run Time Analysis

Removing nodes with maximum degrees modifies the degrees of remaining nodes in the graph. Therefore, the ordering of nodes based on their degrees has to be refreshed at each step. By implementing U and V sets as *binary heaps*, deletion and reordering can be performed in a logarithmic time.

If $|U| + |V| = O(n)$, the initial sorting takes $O(n \log n)$ (lines 4–5). The repeat-until loop (lines 7–23) will be executed n times in the worst case, since at each step at least one node is removed from U or V. The execution of the body of the loop takes $O(\log n)$ assuming that the binary heap is used for the implementation of U and V sets (deletion and reordering). Therefore, the worst case time complexity of this algorithm is $O(n \log n)$.

7 Experimental Results

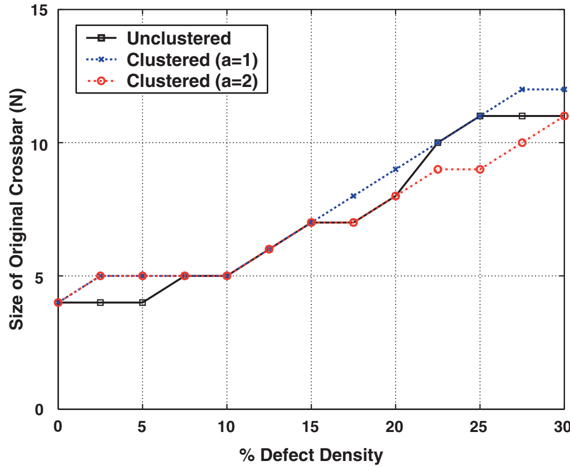
7.1 Yield Analysis Results

The objective in defect tolerance is to extract the maximum defect-free block from a partially-defective manufactured device. Alternatively, if a $k \times k$ defect-free crossbar is required, we find the smallest $n \times n$ crossbar ($n > k$) to be fabricated in a manufacturing environment with defect density d. For the fixed values of k and d, and desirable yield of y, we find an n such that $Y^d_{n,k} = y$ but $Y^d_{n-1,k} < y$. Typically, we set $y = 100\%$. Such evaluations can be performed based on the yield analysis framework, as discussed in Sect. 5, and are utilized in the proposed defect tolerance flow to make the connection between the design view and the physical view (Sect. 4).

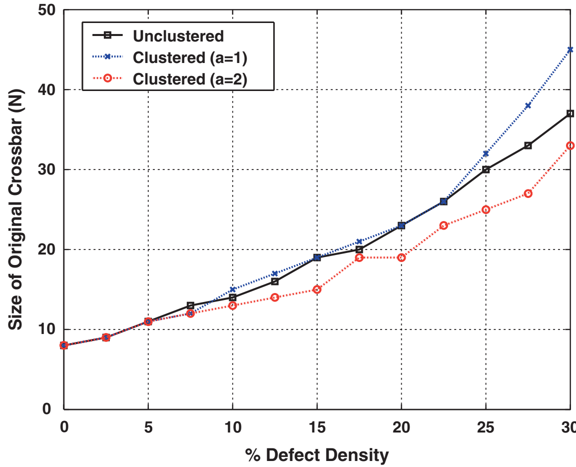
Experimental results for various sized defect-free crossbars in the presence of stuck-open faults are presented in Figs. 13 and 14. In these experiments, the number of simulated fault patterns, FP, ranges between 500 and 5000 depending on the size of the crossbar (n). The results are presented for three different defect distribution models, unclustered and clustered (with $\alpha = 1, 2$). For each defect density, the minimum size of the original crossbar (n) is given such that the desirable defect-free crossbar can be always found (yield = 100%).

Figure 15 shows the same data in a different format. In this figure, the amount of area overhead required to achieve the desirable defect-free subset is reported. The overhead is normalized to the area of the defect-free subset, i.e. n^2/k^2. Note that the y-axis is in the logarithmic scale. As can clearly be seen in this figure, the amount of redundancy required to obtain smaller defect-free subsets are much smaller. Also, the overhead associated with the clustered defect distribution model is much lower than unclustered one. This figure also shows that for any fixed defect density, the area overhead increases exponentially with the size of the required defect-free crossbar. Therefore, it might be more economical to tailor the defect-unaware steps of the design flow to map the design into a set of smaller crossbars rather than a set of larger crossbars. In other words, it costs less to use (map the design into) a large number of small crossbars instead of a small number of large crossbars. However, the impact on the routability of the crossbar array needs to carefully be considered, as well. Hence, there is a minimum (threshold) on the size of the crossbar such that below that threshold, the routability could negatively be affected. Figure 16 shows the size of minimum fabricated crossbar to achieve a defect-free 16×16 crossbar for various yields ($y = 80\%, 85\%, 90\%, 95\%, 100\%$). The defect distribution is unclustered. This figure suggests that the effect of the required yield on the amount of redundancy is almost linear.

The situation for short (bridging) defects is different from open defects. If there is a stuck-closed fault, both nanowires (or at least one nanowire in the relaxed model) intersecting at the faulty switch position become unusable. This corresponds to missing vertices in the graph model instead of missing



(a) 4 × 4 crossbar

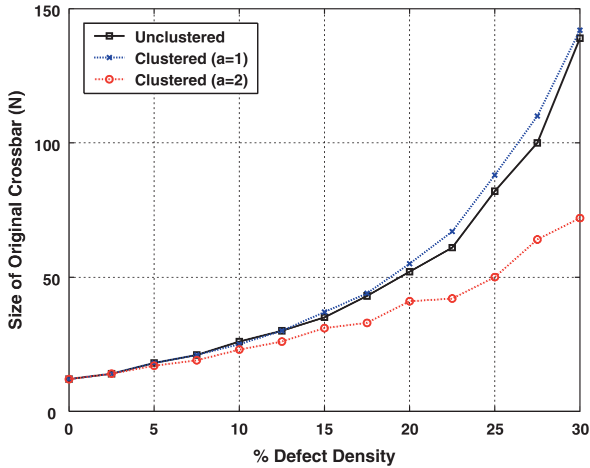


(b) 8 × 8 crossbar

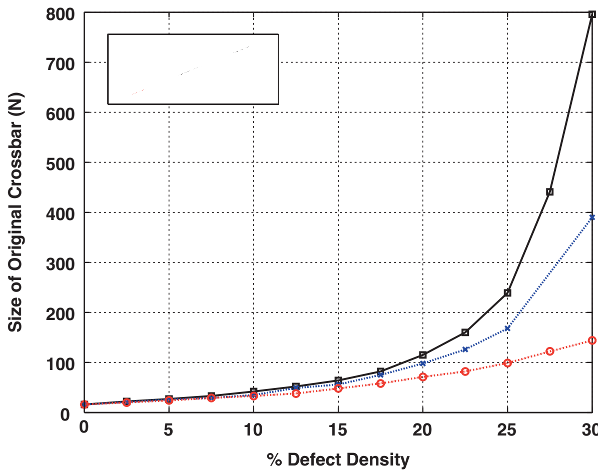
Fig. 13. The size of original crossbar (n) required for obtaining the desirable defect-free crossbar ($k = 4, 8$) in the presence of stuck-open faults (yield = 100%)

edge which is the case of open defects. For open defects, the probability of finding a defect-free crossbar is roughly the function of defect density. However, the probability of finding a defect-free crossbar in the presence of stuck-closed faults is a function of the number of faulty switches. This severely affects the yield metric in the presence of short defects (switch or nanowire).

The difference between stuck-open and stuck-closed faults is well depicted in Fig. 17a. In this figure, the probabilities of finding a defect-free 16 × 16 crossbar within various sized defective crossbars with defect density of 2.5%



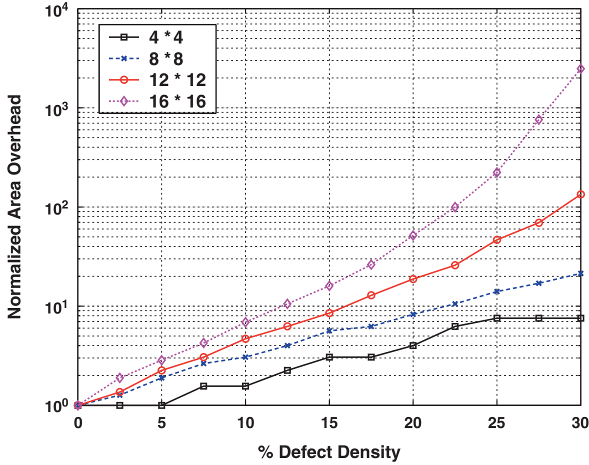
(a) 12 × 12 crossbar



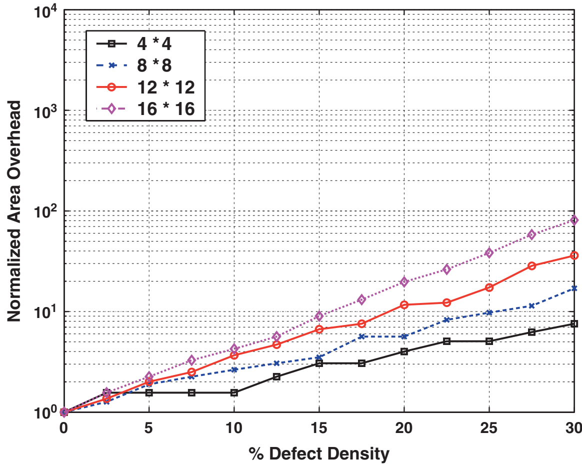
(b) 16 × 16 crossbar

Fig. 14. The size of original crossbar (n) required for obtaining the desirable defect-free crossbar ($k = 12, 16$) in the presence of stuck-open faults (yield = 100%)

for both stuck-open and stuck-closed faults are plotted. The defect distribution model is unclustered. The probability of finding a defect-free crossbar with a fixed size in the presence of stuck-open faults is monotically increasing given the size of the original defective crossbar is increasing as well (assuming that the defect density is also fixed). However, for stuck-closed faults, this probability decreases at some point. This is due to the fact that as the size of the original defective crossbar increases (with a fixed defect density), the number of faulty switches will also increase.



(a) Unclustered



(b) Clustered ($\alpha = 2$)

Fig. 15. Area overhead (redundancy) required to achieve defect-free crossbars with different sizes (yield = 100%)

Unlike for stuck-open faults, the effect of different defect distribution models on the probability of finding a defect-free crossbar in the presence of stuck-closed faults is quite significant. Figure 17b shows the probability of finding a defect-free 16 × 16 crossbar within defective crossbars with defect density of 2.5% using different defect distribution models. As can be seen in this figure, this probability for the clustered distribution model with $\alpha = 2$ is much higher than other models, and converges to more than 90% (while the other two converge to zero).

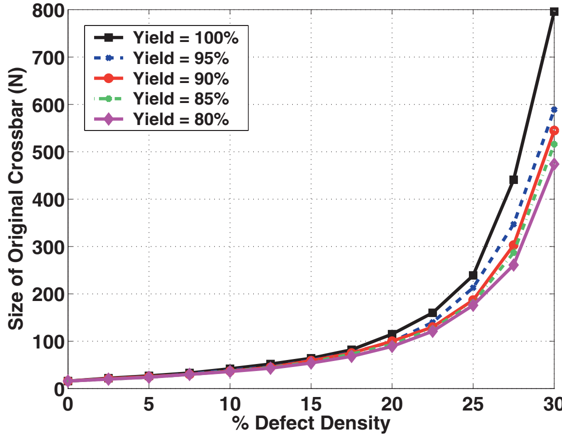


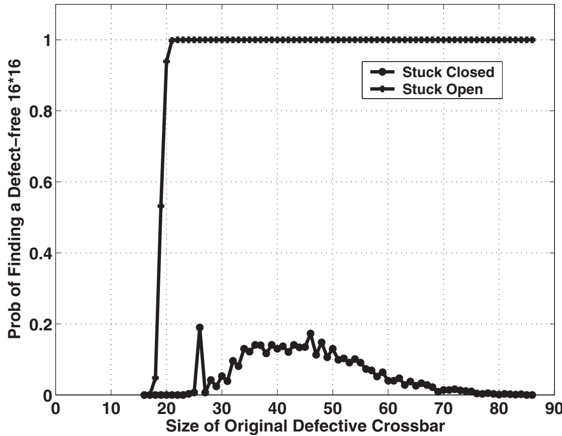
Fig. 16. Size of minimum fabricated crossbar for finding 16 × 16 crossbar with various yields (unclustered defect distribution)

Fortunately, the stuck-closed faults are much less likely than stuck-open faults in the self-assembly fabrication of molecular crossbars, as shown in [11]. Moreover, in the fabrication process, it is possible to tailor the chemical synthesis technique to decrease the probability of having switch stuck-closed faults at the expense of increasing the total number of switch faults [13]. In this case, we expect to have a much higher defect density of stuck-open faults than stuck-closed faults, thus resulting in a better defect tolerance in overall.

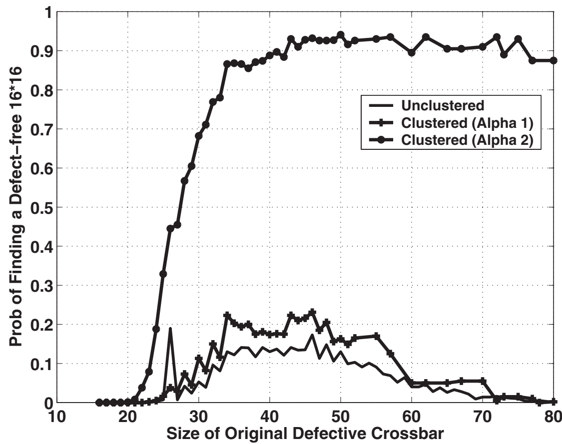
7.2 Comparison of Recursive and Greedy Algorithms

Here we compare the accuracy and run time of the two algorithms presented in Sect. 6. As explained in Sect. 6.1, the recursive algorithm is an exact approach which always gives the optimum solution. However, the greedy algorithm is a heuristic approach with much reduced run time in which the maximum biclique is not guaranteed. Figure 18 shows the results obtained by these two methods for 16 × 16 and 32 × 32 crossbars, using defect densities (stuck-open defects) up to 30%. For each data point, 1000 crossbars with the given defect density are randomly generated and the average sizes of the maximal bicliques obtained by these two methods are reported. As can be seen in this figure, the difference between the recursive (exact) method and the greedy algorithm is almost negligible.

Table 1 shows the run time comparison of these two approaches for three different sizes of the crossbars, 16×16, 24×24, and 32×32. The run time numbers are normalized to the run of the greedy algorithm for 16×16 crossbar with 5% defect density (the smallest run time number). With these crossbars, the greedy algorithm is up to 406 times faster than the recursive (exact) algorithm and this speedup exponentially increases for larger crossbars. As can



(a)



(b)

Fig. 17. Finding a defect-free 16 × 16 crossbar within defective crossbars (d = 2.5%): (**a**) open and short faults (**b**) short faults

be seen in this table, the run time of the recursive (exact) approach grows exponentially with the size of the crossbar. However, the run time of the greedy approach grows almost linearly with the crossbar size. Another observation is that the run time of the greedy algorithms grows monotically with the defect density. This is because the efficiency of the heuristic has a direct relation with the sparseness of the complement graph. In other words, as the defect density increases, the algorithm requires to remove more nodes before termination. However, the run time of the recursive approach does not necessarily increases with the defect density. When the defect density is very low, there are many nodes that are connected to all nodes in the other partition (i.e.

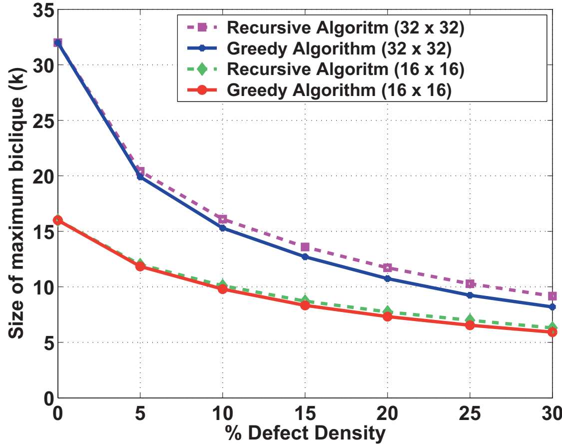


Fig. 18. Greedy Algorithm vs. Recursive Algorithm

Table 1. Normalized execution time for greedy and recursive algorithms

Defect density (%)	16×16		24×24		32×32	
	Greedy	Recursive	Greedy	Recursive	Greedy	Recursive
5	1.0	3.7	3.8	111.4	9.9	3,258.4
10	1.6	9.1	5.8	260.6	12.6	5,127.4
15	1.9	12.3	6.9	205.7	14.8	3,617.4
20	2.1	11.3	6.5	208.6	16.5	2,337.9
25	2.4	9.9	7.2	133.6	16.5	1,520.6
30	2.6	9.5	7.8	103.7	17.7	866.3

satisfying the conditions of lines 8–9 in the pseudo code of Fig. 11). on the other hand, when the defect density is large enough, there are many nodes connected to less than k nodes (i.e. satisfying the conditions of lines 6–7 in the pseudo code of Fig. 11). This is why the run time for very low and very high defect densities is smaller than that for medium cases.

Defect map is generated as a byproduct of these algorithms. As a result, no extra time is required to obtain the defect map (i.e. the execution times in Table 1 include defect map generation). As stated in Sect. 4, the size of the reduced defect map is only $2n$ (the vertical and horizontal vectors).

7.3 Defect Tolerant Crossbar Array

The nano-architecture is based on an array of molecular crossbars. The *global view* of this architecture considers the connections among crossbars and how the design is partitioned and mapped into crossbars. The *local view* focuses on individual crossbars, the resources within each crossbar and how the crossbar

resources are utilized for a particular mapped design. The floorplanning, global placement, and global routing phases of the physical design use the global view, whereas the detailed placement and routing are based on the local view. As stated in the introduction, the main focus of the chapter is defect tolerance in the local view of the nano-architecture. Nevertheless, here we discuss some defect tolerance issues related to the global view of the nano-architecture.

In order to achieve a global defect tolerant architecture, one option is to generate a defect-unaware (application-independent) global architecture which consist of an array of $k \times k$ defect-free crossbar. The size of this crossbar array needs to be the same for all the chip manufactured in the same fabrication environment. In other words, the value of k has to be constant for all crossbars in the chip. Therefore, k has to be chosen such that that all crossbars in the chip have defect-free subsets of minimum size k. This might result in excessive area overhead and smaller values for k. Moreover in this scheme, the chips containing crossbars with very small defect-free subset (smaller than expected k) will be thrown away and this affects the overall yield of manufactured chips.

The other option is to use an application-dependent flow for the global view and use the proposed defect-unaware flow only for the local view. In this scheme, the crossbars on the chip with defect-free subset smaller than k are marked as “unusable” and will not be used for application mapping. As a result, the size of “usable” crossbar array might vary from chip to chip. The advantage of this approach is its reduced area overhead (i.e. larger values of k) compared to the application-independent global defect tolerance, as explained above. This comes with the cost of using application-dependent global mapping per chip. Since the complexity of global mapping (placement and routing) is less than local mapping (detailed placement and routing), the amount of per chip customization of this mixed globally-defect-aware locally-defect-unaware defect tolerance flow is not high.

One major issue that needs to be resolved in the defect tolerant crossbar array is how to make connections between defect-free subsets of individual crossbars. Since defect-free subsets of individual crossbars are locally extracted, the inputs/outputs of defect-free subsets of adjacent crossbars might not fully match. In this situation, we use some crossbars just to make defect-free matching between the inputs and outputs of defect-free subsets of crossbars [29]. This $n \times n$ crossbar, which makes the connection between k particular input nanowires to k particular output nanowires using a one to one matching, is called a *permutation crossbar*. In the application-independent defect-tolerant architecture, the crossbars are divided into two sets of *user crossbars* (UCB) and *permutation crossbars* (PCB). UCBs are used for the implementation of logic, programmable switch block, or non-volatile memory array. The design view of an $n \times n$ partially-defective crossbar used as a UCB is a $k \times k$ defect-free complete crossbar (biclique). UCBs are connected through PCBs. PCBs are transparent to the mapped design and the design flow. In other words, only UCBs exist in the design view and can be used for application mapping.

PCBs provide defect-free matching between their input and output nanowires participating in the corresponding defect-free subsets of the adjacent UCBs to that PCB.

8 Conclusions

Defect densities in self-assembly enabled nanotechnology are significantly higher than those in the lithography-based CMOS technology due to non-determinism in self-assembly nano-fabrication and atomic device sizes. Defect tolerant techniques are therefore essential for the designs realized using this nanotechnology to achieve an acceptable level of manufacturing yield.

In this chapter, we have proposed a defect-unaware design flow for defect tolerance of reconfigurable crossbar array architectures. The presented application-independent flow has many advantages over the defect-aware adaptive defect tolerance (application-dependent), including a considerable reduction in the defect map size, negligible per-chip customized design effort, and predictability in the performance of mapped designs. We have also investigated defect tolerance of the two-dimensional crossbar, which is used as the basic building block for logic, interconnect and memory in various nano-architectures. We have presented a yield analysis method based on the proposed defect-unaware flow in which the probability of finding a defect-free subset within a partially-defective fabricated crossbar is identified. We have presented algorithms to identify the maximum defect-free crossbar with the partially-defective fabricated crossbar. These algorithms can be used in yield analysis as well as final mapping and defect map generation steps.

References

1. S. Iijima. Helical Microtubules of Graphitic Carbon. In *Nature*, volume 354, pages 56–58, 1991.
2. A. Bachtold, P. Harley, T. Nakanishi, and C. Dekker. Logic Circuits with Carbon Nanotube Transistors. In *Science*, volume 294, pages 1317–1320, 2001.
3. T. Rueckes, K. Kim, E. Joselevich, G. Y. Tseng, C. L. Cheung, and C. M. Lieber. Carbon Nanotube-Based Nonvolatile Random Access Memory for Molecular Computing. In *Science*, volume 289, pages 94–97, 2000.
4. T. Kamins, R. Williams, Y. Chen, Y. -L. Chang, and Y. Chang. Chemical Vapor Deposition of Si Nanowires Nucleated by TiSi2 Islands on Si. In *Applied Physics Letters*, volume 76, pages 562–564, 2000.
5. Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K. H. Kim, and C. M. Lieber. Logic Gates and Computation From Assembled Nanowire Building Blocks. In *Science*, volume 294, pages 1313–1317, 2001.
6. K. Chen, K. Maezawa, and M. Yamamoto. Inp-based High-Performance Monostable-Bistable Transition Logic Elements (MOBILE's) Using Integrated Multiple-Input Resonant-Tunneling Devices. In *IEEE Electron Device Letters*, volume 17, pages 127–129, 1996.

7. K. Likharev. Single Electron Devices and their Applications. In *Proc. IEEE*, volume 87, pages 606–632, 1999.
8. P. Tougaw and C. Lent. Logical Devices Implemented Using Quantum Cellular Automata. In *Applied Physics*, volume 75, pages 1818–1825, 1994.
9. M. Butts, A. DeHon, and S. C. Goldstein. Molecular Eletronics: Devices, Systems and Tools for Gigagate, Gigabit Chips. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 443–440, 2002.
10. Y. Cui and C. M. Lieber. Functional Nanoscale Electronics Devices Assembled Using Silicon Nanowire Building Blocks. In *Science*, volume 291, pages 851–853, 2001.
11. A. DeHon. Array-Based Architecture for FET-Based, Nanoscale Electronics. In *IEEE Trans. on Nanotechnology*, volume 2, pages 23–32, 2003.
12. A. DeHon and M. J. Wilson. Nanowire-Based Sublithographic Programmable Logic Arrays. In *Proc. ACM Int'l Symp. on Field-Programmable Gate Arrays*, pages 123–132, 2004.
13. S. C. Goldstein and M. Budiu. NanoFabrics: Spatial Computing Using Molecular Electronics. In *Proc. Int'l Symp. on Computer Architecture*, pages 178–189, 2001.
14. M. M. Ziegler and M. R. Stan. Design and Analysis of Crossbar Circuits for Molecular Nanoelectronics. In *Proc. IEEE Int'l Conf. on Nanotechnology*, pages 323–327, 2002.
15. C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams, and J. R. Heath. Electronically Configurable Molecular-Based Logic Gates. In *Science*, volume 285, pages 391–394, 1999.
16. Y. Chen, G. Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams. Nanoscale Molecular-Switch Crossbar Circuits. In *Nanotechnology*, volume 14, pages 462–468, 2003.
17. A. DeHon, P. Lincoln, and J. E. Savage. Stochastic Assembly of Sublithographic Nanoscale Interfaces. In *IEEE Trans. on Nanotechnology*, volume 2, pages 165–174, 2003.
18. X. Ma, D. Strukov, J. Lee, and K. Likharev. Afterlife for Silicon: CMOL Circuit Architectures. In *Proc. IEEE Conf. on Nanotechnology*, pages 175–178, 2005.
19. D. Strukov and K. Likharev. CMOL FPGA: A Reconfigurable Architecture for Hybrid Digital Circuits with Two-Terminal Nanodevices. In *Nanotechnology*, volume 16, pages 888–900, 2005.
20. Nantero. www.nantero.com, 2005.
21. H. Naeimi and A. DeHon. A Greedy Algorithm for Tolerating Defective Crosspoints in NanoPLA Design. In *Proc. Int'l Conf. on Field-Programmable Technology*, pages 49–56, 2004.
22. M. Mishra and S. C. Goldstein. Defect Tolerance at the End of the Roadmap. In *Proc. Int. Test Conf.*, pages 1201–1211, 2003.
23. M. B. Tahoori and S. Mitra. Defect and Fault Tolerance in Reconfigurable Molecular Computing. In *Proc. Field Custom Computing Machines*, pages 176–185, 2004.
24. Z. Wang and K. Chakrabarty. Using Built-In Self-test and Adaptive Recovery for Defect Tolerance in Molecular Electronics-Based Nanofabrics. In *Proc. Int. Test Conf.*, pages 477–486, 2005.
25. R. Rad and M. Tehranipoor. SCT: an approach for testing and configuring nanoscale devices. In *Proc. VLSI Test Symp.*, pages 370–377, 2006.

26. J. Huang, M. B. Tahoori, and F. Lombardi. On the Defect Tolerance of Nano-Scale Two-Dimensional Crossbars. In *Proc. IEEE Int'l Symp. on Defect and Fault Tolerance of VLSI systems*, pages 96–104, 2004.
27. F. J. Meyer and D. K. Pradhan. Modeling Defect Spatial Distribution. In *IEEE Trans. on Computers*, volume 38, pages 538–546, 1989.
28. F. C. Doherty, J. R. Lundgren, and D. J. Siewert. Bicliquecovers and Partitions of Bipartite Graphs and Digraphs and Related Matrix Ranks of 0,1-Matrices. In *Congressus Numerantium 136*, pages 73–96, 1999.
29. M. B. Tahoori. Application Independent Defect Tolerant Crossbar Nano-Architectures. In *Proc. IEEE Int'l Conf. on Computer Aided Design*, 2006.

Section 2: Test and Defect Tolerance for QCA Circuits

M. Tehranipoor

Traditionally, cellular automata (CA) were commonly implemented as software programs. However, a physical implementation of an automaton using quantum-dot cells was proposed about a decade ago. The automaton quickly gained popularity among researchers in academia and research laboratories. The discrete nature of both, cellular automata and quantum mechanics, was then combined to create nanoscale devices capable of performing computation at very high switching speeds and consuming extremely small amounts of power. Today, standard solid state quantum-dot cellular automata (QCA) cell design considers the distance between quantum dots to be about 15 nm, and a distance between cells of about 50 nm. Just like any CA, Quantum-dot Cellular Automata are based on the simple interaction rules between cells placed on a grid. A QCA cell is constructed from four quantum dots arranged in a square pattern. These quantum dots are the sites in which electrons can occupy by tunneling to them.

Complementary metal-oxide semiconductor (CMOS) technology has been the industry standard for implementing very large scale integrated (VLSI) devices for the last two decades, and for very good reasons mainly due to the consequences of miniaturization of such devices. QCA is only one of the many alternative technologies proposed as a replacement solution to the fundamental limits CMOS technology will impose in the years to come. Optimistic assumptions suggest that intrinsic switching time of a QCA cell is in the order of terahertz, however, switching speed is not limited by a cells intrinsic switching speed but by the proper quasi-adiabatic clock switching frequency setting. QCA technology resolves, in principle, the problems of current CMOS technology, and it is currently limited by the availability of its practical fabrication methods.

This section contains four chapters that deal with various challenges in test and defect tolerance for QCA devices. The first chapter (Chap. 6) in this section entitled “Reversible and Testable Circuits for Molecular QCA Design” describes that reversible logic design is a well-known paradigm in digital computation. While an extensive literature exists on its mathematical

characterization, little work has been reported on its possible technological basis. In this chapter, Quantum-dot Cellular Automata is investigated for testable implementations of reversible logic. In QCA, the basic combinational unit is the majority voter, so new criteria must be met in the logic design. Two new reversible gates (denoted as QCA1 and QCA2) are proposed. These gates are compared with other reversible gates (such as Toffoli and Fredkin) for QCA implementation. Figures of merit for the evaluation of area, delay and synthesis, are provided. As the bijective nature of reversibility makes testing significantly easier than in the general case, testing of the reversible gates is pursued in detail. As applicable to molecular QCA implementations, different array configurations are treated in detail under single and multiple faults. Issues related to controllability and observability are addressed for both fully exhaustive and reduced test sets.

"Cellular Array-based Delay-insensitive Asynchronous Circuits Design and Test for Nanocomputing Systems" is the second chapter (Chap. 7) in this section. This chapter presents the design, layout, and testability analysis of delay-insensitive circuits on cellular arrays for nanocomputing system design. In delay-insensitive circuits the delay on a signal path does not affect the correctness of circuit behavior. The combination of delay-insensitive circuit style and cellular arrays is a useful step to implement nanocomputing systems. In the approach proposed in this chapter the circuit expressions corresponding to a design are first converted into Reed–Muller forms and then implemented using delay-insensitive Reed–Muller cells. The design and layout of the Reed–Muller cell using primitives are described in detail. The effects of stuck-at faults in both delay-insensitive primitives and gates are analyzed. Since circuits implemented in Reed–Muller forms constructed by the Reed–Muller cells can be easily tested offline, the proposed approach for delay-insensitive circuit design improves a circuits testability. Potential physical implementation of cellular arrays and its area overhead are also discussed in this chapter.

Chapter 8, entitled "QCA Circuits for Robust Coplanar Crossing", proposes different circuits of QCA for the so-called coplanar crossing. Coplanar crossing is one of the most interesting features of QCA because it allows for mono-layered interconnected circuits, whereas CMOS technology needs different levels of metallization. However, the characteristics of the coplanar crossing make it prone to malfunction due to thermal noise or defects. The proposed circuits exploit the majority voting properties of QCA to allow a robust crossing of wires on the Cartesian plane. This is accomplished using enlarged lines and voting. A Bayesian Network (BN) based simulator is utilized for evaluation; results are provided to assess robustness in the presence of cell defects and thermal effects. The BN simulator provides fast and reliable computation of the signal polarization vs. normalized temperature. Simulation of the wire crossing circuits at different operating temperatures is provided with respect to defects and a quantitative metric for performance under temperature variations is proposed and assessed.

The final chapter in this section (Chap. 9) entitled "Reliability and Defect Tolerance in Metallic Quantum-dot Cellular Automata" focuses on the issue of robustness in the presence of disorder and thermal fluctuations. It examines the performance of a semi-infinite QCA shift register as a function of both clock period and temperature. The existence of power gain in QCA cells acts to restore signal levels even in situations where high-speed operation and high-temperature operation threaten signal stability. Random variations in capacitance values can also be tolerated.

Chapter 6: Reversible and Testable Circuits for Molecular QCA Design

X. Ma, J. Huang, C. Metra, and F. Lombardi

1 Introduction

Emerging technologies have been widely advocated to supersede the projected limitations of CMOS at the end of the roadmap [27]. Computation at nano regimes is substantially different from conventional VLSI. Extremely small feature size, high device density and low power are some of the attributes that emerging technologies must address, while implementing new computational paradigms [1]. One of these paradigm is *reversible computing*. Reversible computation is accomplished by establishing a one-to-one onto mapping between the input states and output states of the circuit [7]. This *bijective property* was initially investigated by Landauer who showed that $kT \ln 2$ joules of energy are generated for each bit of information lost due to non reversible computation [6]. But, if computation is performed in a reversible manner, it has been shown that $kT \ln 2$ energy dissipation would not necessarily occur. Due to the bijective property, testing of reversible logic is generally simpler than conventional irreversible logic [31].

An extensive literature exists on reversible computing [2, 4, 5], different gates as primitives for reversible logic computation have been proposed. In most cases, an elegant mathematical analysis (such as those based on the conservative property) of these gates has been provided to describe a technology independent characterization by which reversible computing (mostly at logic level) can be accomplished. However, little work has been reported on the capabilities of emerging technologies to perform reversible computation.

Among emerging technologies, Quantum-dot Cellular Automata (QCA) [9,18] relies on novel design concepts to exploit new physical phenomena (such as Coulombic interactions), and implement unique paradigms [10, 14] with desirable features such as low power. In QCA, gates (such as the inverter, INV and majority voter, MV) and other devices (such as the binary wire and the inverter chain) have been proposed as primitives for combinational circuit design [16]. It has been shown [25, 26] that for QCA, the functions with at

most three input variables (such as the MV) provides the basis for efficient combinational design.

As a combined methodology for computation and communication [11, 13, 14], different designs of logic circuits have been proposed for QCA implementation [14, 15, 17, 19, 20]. Recent developments in manufacturing involve molecular assembly of the QCA devices to supersede metal-based implementations [12]. At very small features sizes, self-assembly and large scale cell deposition on insulated substrates have been proposed.

QCA has also very low power consumption. Using an induced electric field mechanism for clocking (made of four phases), true power gain is possible in this technology. QCA has been used to quantitatively assess the relation between computation and energy dissipation [32]. In QCA, the information stored in the cell is "erased" when cell is released by clock. Two strategies have been considered for clocking QCA [32]: logically irreversible "erase" and logically reversible "copy-then-erase". The "copy" operation is performed by a nearby QCA cell (called the demon cell). It has been shown that erasure without copying requires an amount of dissipated energy in the order of the signal energy. However, the energy dissipation during a "copy then erase" process in which a copy of the bit is retained, can be made arbitrarily small. The operation of a binary wire (i.e. a QCA shift register) can be viewed as a sequence of "copy then erase" operations. It has been shown that the energy dissipated per switching event can be much less than $kT \ln 2$. However, it should be pointed out that the majority voting function is logically irreversible, because the information in the minority input is lost during computation. A novel clocking scheme referred to as *Bennett clocking* has been proposed for QCA in [33]. Bennett clocking offers a practical realization of reversible computing. The basic principle of Bennett clocking is that bit information is held in place by the clock until a computational block is complete, then an erasure takes place in the reverse order of computation. The process is reversible and has been shown by direct calculation that energy dissipation per switching event is much less than $kT \ln 2$ for QCA circuits containing MV and fanout [33]. Bennet clocking does not require any change in the QCA layout, i.e. no additional circuit complexity at QCA level, only clocking signals are modified.

Although it is possible in theory to design a system with virtually zero energy dissipation with Bennett clocking, the main focus of this chapter is to analyze the testability advantage of logically reversible QCA systems by considering different features such as fault model, test set cardinality, observability and controllability. In this chapter, new reversible logic gates are defined in QCA by their bijective property, i.e. the one-to-one onto mapping between inputs and outputs. However, QCA does not have the same logic and physical features of existing technologies, such as CMOS. Hence, the logic definition of a reversible gate must take into consideration the unique features of QCA, such as the majority voter based synthesis and timing requirements. Additionally, reversible circuits are amenable to efficient testing due to their bijective property. The one-to-one onto mapping of reversible logic improves

both controllability and observability of a circuit. As applicable to *molecular implementations*, different array structures of basic units (referred to as modules made of reversible gates) are analyzed under single and multiple fault models. It is shown that constant testability (C-testability) is achieved in these arrays under different conditions of controllability, observability, fault models and test sets.

This chapter is organized as follows. Section 2 presents a brief review of reversible computing. Section 3 deals with a brief review of QCA. Preliminaries inclusive of the defect model and assumptions are presented in Sect. 4. Reversible gates for QCA implementation are proposed in Sect. 5. Defect analysis and testing for the QCA gates are given in Sect. 6. Section 7 discusses testing of one-dimensional arrays made of reversible modules with controllability limited to the primary inputs and outputs of the array. The issue of array testability, controllability and observability are analyzed in Sect. 8. Section 9 presents the testing process for one-dimensional arrays in which controllability and observability are added to the intermediate modules. Section 10 concludes the chapter.

2 Reversible Computing

Reversible computing is based on invertible primitives and composition rules that preserve invertibility [2]. Reversible computing can be used in quantum computing, optical computing and nanotechnology and has attracted considerable research attention in recent years [5]. The basic principle of reversible computing is that no information is lost during the computation process and the entire computing process is invertible, i.e. given the final state of the system, a unique initial state can be determined.

In has been proved in [6] that the irreversibility of traditional logic gates inevitably leads to power dissipation in the order of kT for the erasure of each bit of information. Although currently the power dissipation of CMOS circuits is much higher than kT, in the future with low power dissipation and an increased level of integration of nano-based technologies this theoretical limit may become a major barrier. Bennett [4] has shown that to avoid energy dissipation reversible logic gates must be used. Using reversible logic, it is ideally possible to build sequential circuits with no internal power dissipation [2]. Furthermore, [2] has shown that even if the reversible circuit is interfaced with conventional irreversible logic gates, the power dissipation at the interface is in the worst case proportional to the number of inputs and outputs (instead of the number gates). A reversible logic function is a *one-to-one onto* mapping (bijection) between inputs and outputs, i.e. each input pattern is mapped to a unique output pattern, while each output pattern has a unique input pattern mapped to it. Traditional logic functions (such as AND and OR) are not reversible, because more than one input state is mapped to a common output state. In this case, given the output state, it is not possible to determine

the input state, thus information is lost during the computation process. INV is a simple example of a reversible logic gate. The most studied reversible logic gates are the Toffoli [2] and the Fredkin gates [3]. Quantum analysis of these two gates has been given in [7]. In the simplest form, the Toffoli gate has three inputs and three outputs and is *universal* [2], i.e. any combinational reversible logic circuit can be build with Toffoli gates. The Fredkin gate also has three inputs, three outputs and is universal. In addition, the Fredkin gate is also conservative, i.e. the number of "1"s in the outputs is the same as the number of "1"s in the inputs.

The bijective nature of reversible gates makes testing significantly *simpler* compared to the conventional case. Reversible logic gates are *information lossless*, i.e. the information output of a reversible circuit is maximized. Therefore according to [30], the probability of fault detection is maximized too. Reversible logic is inherently easier to test because the one-to-one onto property improves the controllability as well as the observability of the circuit. It has been proved [31] that any test set that detects all single stuck-at faults must detect all multiple stuck-at faults. Efficient test generation algorithms can be used [31] to obtain a test set of half the size of that generated by conventional ATPG. One bound presented in [31] shows that the size of the test set grows at most logarithmically with the size of the circuit. The testability problem for a subclass of reversible logic gates, namely the k-CNOT gates have been investigated in [35]. It is shown that n-wire reversible circuits have a universal test set of size $n^2 + 2n + 2$. Iterative Logic Arrays (ILAs) built with reversible logic gates have been also considered in [35] under a single module fault assumption. Assuming each module has h horizontal inputs (outputs) and v vertical inputs (outputs), it has been proved that a 1D array with a single faulty module is C-testable and fault detection requires 2^{h+v} test vectors, i.e. an exhaustive test set. Furthermore, it has been shown that any d-dimensional array is C-testable under a single-faulty-module assumption. However, multiple faulty modules are not addressed; in this case, masking may occur. Moreover, multiple faulty modules are likely to be present in nanotechnology, especially for biological and molecular based implementations. In this chapter, the test properties of reversible logic are investigated in detail. Test properties of 1D ILAs made of modules with reversible logic gates in QCA are considered under single and multiple faulty modules. It will be shown that a reversible 1D array has the attractive property of C-testability under specific fault assumptions.

3 Review of QCA

A QCA cell can be viewed as a set of four charge containers or "dots", positioned at the corners of a square [16]. The cell contains two extra mobile electrons which can quantum mechanically tunnel between dots, but not cells. The electrons are forced to the corner positions by Coulombic repulsion. The

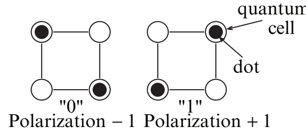


Fig. 1. QCA cell and polarization states

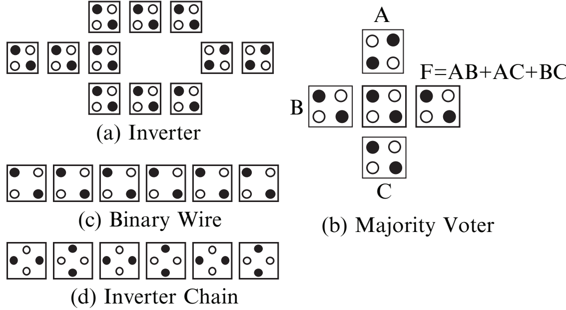


Fig. 2. Basic QCA devices

two possible polarization states represent logic “0” (polarization $P = -1$) and logic “1” (polarization $P = +1$), as shown in Fig. 1. Unlike conventional logic circuits in which information is transferred by electrical current, QCA operates by the Coulombic interaction that connects the state of one cell to the state of its neighbors. This results in a technology in which information transfer (interconnection) is the same as information transformation (logic manipulation).

One of the basic logic gates in QCA is the *majority voter* (MV) [16]. The majority voter with logic function $MV(A, B, C) = AB + AC + BC$, can be realized by only five QCA cells, as shown in Fig. 2b. Logic AND and OR functions can be implemented from the majority voter by setting one input (the so-called programming or control input) permanently to a 0 or 1 value. The inverter is the other basic gate in QCA and is shown in Fig. 2a. The binary wire and inverter chain (as interconnect fabric) are shown in Figs. 2c,d.

In VLSI systems, timing is controlled through a reference signal (i.e. the *clock*); however, timing in QCA is accomplished by clocking in four distinct and periodic phases [12] (as shown in Fig. 3). A QCA circuit is partitioned into serial (one-dimensional) zones, and each zone is maintained in a phase. The use of a *quasi-adiabatic switching* technique for QCA circuits requires a four-phased clocking signal which is commonly supplied by CMOS wires buried under the QCA circuitry for modulating the electric field. The four phases are Relax, Switch, Hold and Release. During the Relax phase, there is no interdot barrier and a cell remains unpolarized. During the Switch phase, the interdot barrier is slowly raised and a cell attains a definitive polarity under the influence of its neighbors. This is the phase in which the actual

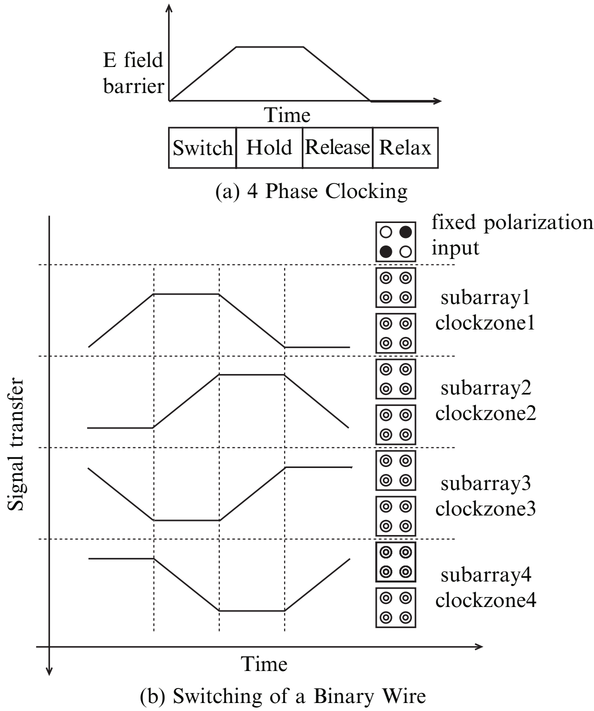


Fig. 3. Four-phased signal for clocking zones in QCA, adiabatic switching

computation takes place. In the Hold phase, barriers are high and a cell retains its polarity. Finally in the Relax phase, barriers are lowered and a cell loses its polarity. Timing zones of a QCA circuit or system are arranged by following the periodic execution of these four clock phases. Zones in the Hold phase are followed by zones in the Switch, Release and Relax phases, one after another. There is effectively a latch between two clocking zones. A signal is latched when one clocking zone goes into Hold phase and acts as input to the subsequent zone. This clocking mechanism provides inherent pipelining [23,24] and allows multi-bit information transfer for QCA through signal latching. Because a zone in the Hold phase is followed by a zone in the Switch phase (and preceded by a zone in the Relax phase), the computation in QCA is strictly one-dimensional (i.e. unidirectional and consistent with signal propagation). Designs are partitioned along one dimension (say the X-axis), thus effectively creating columns of clocking zones. The clocking signal is applied through an underlying CMOS circuitry that generates the required electric field to modulate the tunnelling barrier of all cells in the zones.

At logic level, two synthesis approaches are applicable to QCA-based design. The first approach referred to as AND/OR-based synthesis, has been recently proposed specifically for QCA combinational circuits [8]. This

approach reduces the number of MV gates required for computing three variable Boolean functions to facilitate the conversion of SOP expressions into QCA majority logic. Thirteen standard functions [8] are utilized to completely represent all three-variable Boolean functions. Using a three cube representation, an iterative procedure is proposed [8] to generate a reduced majority gate expression that is amenable to QCA. The generated expression however is not always optimal, because for some standard functions the expression contains three levels of MVs. It is know that any combinational function of at most three literals can be implemented with a two-level [1] MV network.

The second synthesis approach which can be applied to QCA, is the so-called MV-based synthesis [1]. This approach relies on the logic analysis of the MV function as an instance of threshold logic. Threshold logic has been extensively analyzed in the past and the majority threshold function of three variables (i.e. corresponding to the MV) is equivalent to a logic representation which can be easily implemented in QCA. Synthesis under the technique of [1] is based on identifying negated or permutated variables of a function such that restrictions can be generated to comply with the voting nature (such as agreement or disagreement) of this threshold function. An iterative process which can be extended to the general case of n variable voting functions, is described in [1].

This chapter presents reversible gates implemented by QCA. In addition to the Toffoli and the Fredkin gates, two new reversible gates (namely QCA1 and QCA2) are proposed. Area, number of QCA cells and delay of the four reversible logic gates are analyzed and compared.

4 Preliminaries

QCA gates used in this chapter contains MV, fanout and fixed polarization cells. These gates are therefore not strictly "reversible". However, the focus of this chapter is not constructing a system with zero energy dissipation, but rather exploring the testability advantage of logically reversible system and the one-to-one onto mapping. In this chapter, reversible logic gates are defined by their bijective property between inputs and outputs.

The following notation and assumptions are valid hereafter:

1. As applicable to molecular implementations, *missing/additional* QCA cell defects are considered in a gate. Only defects in the active devices (MV and INV) are considered. For a QCA interconnect, results under this defect model have been reported in previous manuscripts [21, 22]. A *single* fault per gate is assumed.
2. The basic function of a reversible logic gate is a one-to-one onto mapping of input to output patterns. Let a_i represent the 3 bit pattern whose decimal value is i, e.g. $a_0 = 000$, $a_5 = 101$. Then a reversible logic gate can be represented by an input/output mapping $a_i \rightarrow a_j$. Independently

of the status (defective or defect-free) of a gate, the number of distinct output patterns can not be greater than the number of distinct input patterns. The ith faulty mapping from inputs to outputs results in a fault pattern that is denoted by FP_i.
3. In all cases, simulation is performed using QCADesigner v1.4 and its coherence vector engine [36]. All cells used in the simulations have size of 10 nm×10 nm, and dot size of 2.5 nm. Cell to cell distance in a binary wire is set to 2.5 nm. Additional/missing cell defects are identified by the x and y coordinates in the layout of each device. In the layout, each MV is highlighted by a dotted grid.
4. The four phase clocking is assumed in all QCA designs; cells in the different clocking zones are represented by different colors.

5 Reversible Gates in QCA

In this section, the QCA implementations of four gates are presented. Two of these gates are new as applicable to QCA. As consistent with previous papers and practical considerations [2,5], each gate has three inputs and three outputs and is universal.

– *Fredkin gate.* The Fredkin gate [3] has a truth table shown in Table 1 with three output functions given as follows (where u' denotes the complement of u):

$$v = u,$$
$$y1 = u'x2 + ux1,$$
$$y2 = u'x1 + ux2.$$

The QCA implementation of the Fredkin gate is shown in Fig. 4 with its logic schematic diagram in Fig. 5. This corresponds to a two-level MV implementation. Altogether six MVs are used: four MVs are used as AND gates, while the other two MVs are used as OR gates.

Table 1. Truth table of the Fredkin gate

u $x1$ $x2$	v $y1$ $y2$
000	000
001	010
010	001
011	011
100	100
101	101
110	110
111	111

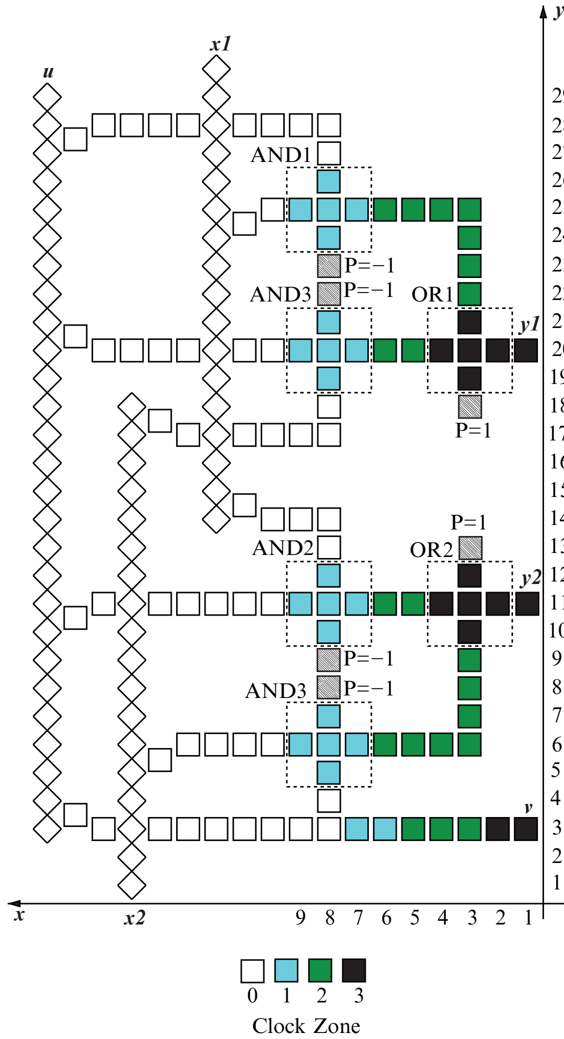


Fig. 4. QCA layout of the Fredkin gate

– *Toffoli gate.* The Toffoli gate [2] is defined by the truth table in Table 2. The output functions are

$$y1 = x1x2' + x1x3' + x1'x2x3,$$
$$y2 = x2,$$
$$y3 = x3.$$

A QCA implementation of the Toffoli gate is presented in Fig. 6 with its schematic in Fig. 7. This is also a two-level MV implementation with four MVs. One of the MV is used as AND gate and one is used as an OR gate.

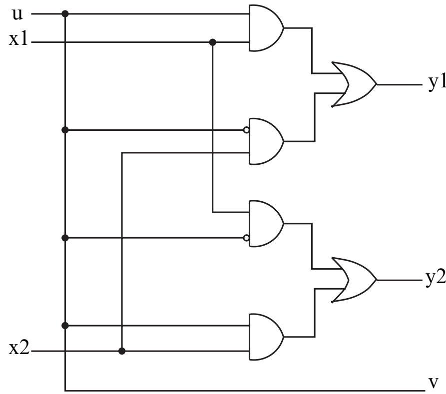


Fig. 5. Logic schematic diagram of the Fredkin gate

Table 2. Truth table of the Toffoli gate

$x1\ x2\ x3$	$y1\ y2\ y3$
000	000
001	001
010	010
011	111
100	100
101	101
110	110
111	011

- *QCA1*. The first gate designed for QCA implementation is referred to as QCA1. This reversible gate is defined by the truth table in Table 3. The output functions are

$$y1 = MV(x1, x2, x3),$$
$$y2 = MV(x1, x2, x3'),$$
$$y3 = MV(x1', x2, x3).$$

  A QCA implementation of the QCA1 gate is shown in Fig. 8 with its schematic diagram in Fig. 9. This gate requires only one-level MV implementation in QCA.
- *QCA2*. The second gate designed for QCA implementation is referred to as QCA2. This gate has a truth table given in Table 4. Its output functions are

$$y1 = MV(x1, x2, x3),$$
$$y2 = MV(x1, x2, x3'),$$
$$y3 = MV(x1', x2, x3').$$

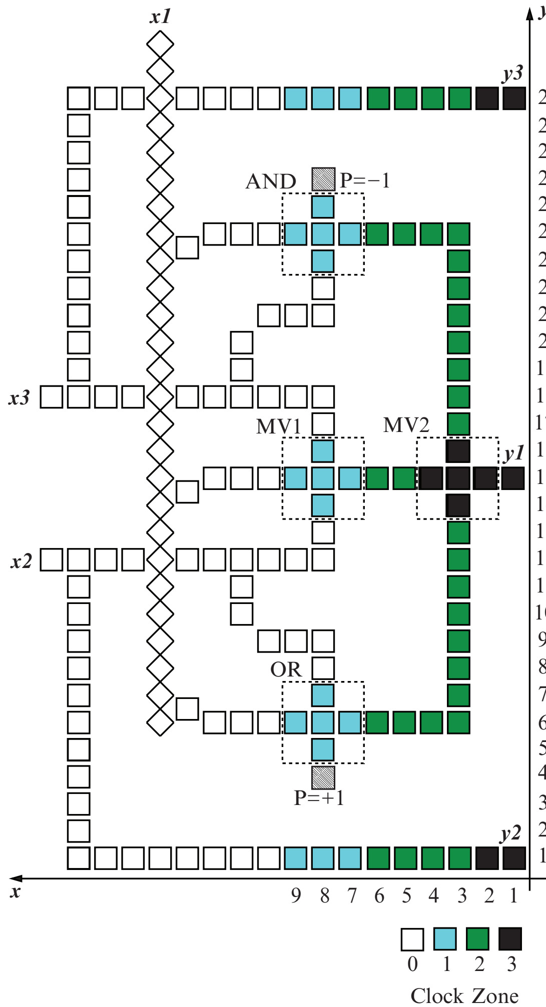


Fig. 6. QCA layout of the Toffoli gate

QCA2 has similar properties to QCA1. A QCA implementation of QCA2 is given in Fig. 10 with a schematic diagram in Fig. 11. The implementation requires only one-level MV (with three disjoint MVs).

QCA1 and QCA2 allow multiple MV functions to be embedded in a gate at the same time, thus they are amenable to threshold-based synthesis.

The implementations of the four reversible logic gates can be compared according to different measures as applicable to QCA. The results are shown in Table 5. The number of *clocking zones* in each reversible logic gate is presented to quantify the delay between inputs and outputs. Also, the *geometric area*

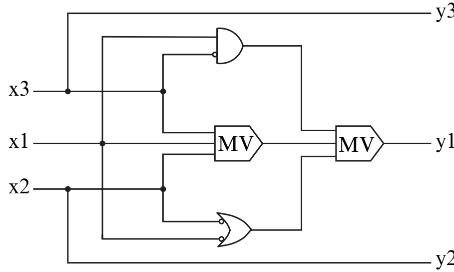


Fig. 7. Logic schematic diagram of the Toffoli gate

Table 3. Truth table of QCA1

$x1$ $x2$ $x3$	$y1$ $y2$ $y3$
000	000
001	001
010	011
011	101
100	010
101	100
110	110
111	111

occupied by each gate is provided. This is defined as the *rectangular area* occupied by the design. For example, the geometric area of a single MV is 3×3. The number of QCA cells used in each design is also compared. The *control cells* are the input cells with fixed polarization ($P = +1$ or $P = -1$), which are used to program the MV as either a 2-input OR, or AND gate. All other cells are referred to as *normal cells*. In these implementations, two types of normal cells are used, namely the non-rotated cells (for the MVs and binary wires) and the rotated cells (obtained by rotating the non-rotated cell by 45° for the inverter chains).

As QCA1 and QCA2 have one-level MV implementations and the Fredkin and Toffoli gates have two-level MV implementations, the delay (number of clocking zones) is smaller for the former gates. Also, QCA1 and QCA2 occupy a smaller area with a reduced number of QCA cells (and no control cell).

The 13 standard combinational functions proposed in [8] are initially utilized for comparison purposes. These functions represent all 256 three-variable Boolean functions. Let the three Boolean variables be denoted as a, b and c, then the 13 standard functions are shown in Table 6 (where A, B and C can be mapped to any one of a, b, c, a', b', c'). For example, $F = a'b + bc'$ and $F = bc + ca$ can both be represented by the same standard function $F = A'B + BC'$. For comparison, the 13 standard functions have been implemented using each of the four reversible gates. The number of gates and the number of clocking zones are reported in Table 6.

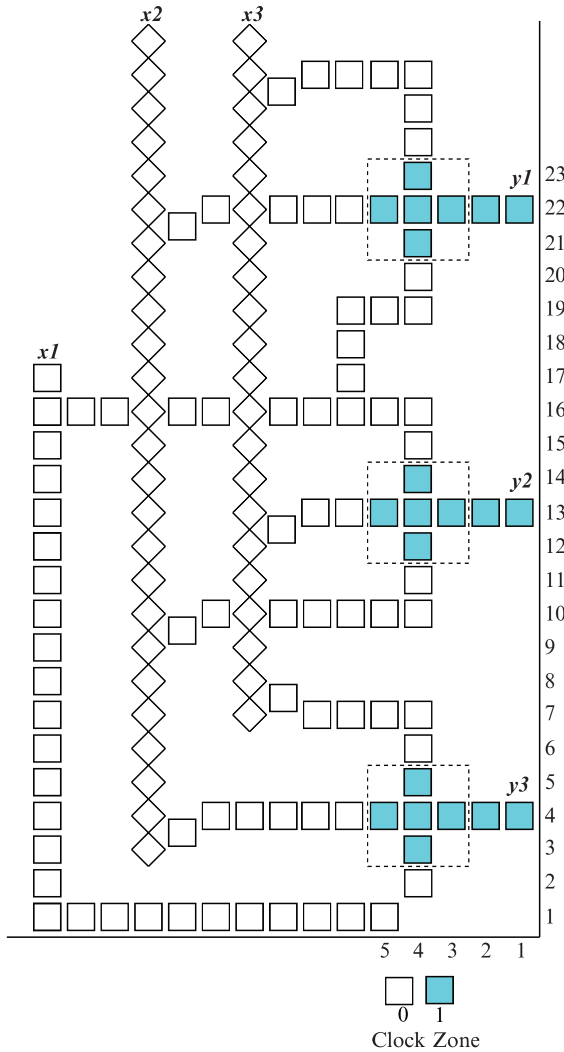


Fig. 8. QCA layout of QCA1

As in previous papers, an additional clocking zone is needed to connect two gates. For example in Table 6, the function F_1, implemented by Fredkin gates requires two gates and has a total delay of nine clocking zones, i.e. in addition to the four clocking zones of each Fredkin gate, gates are concatenated through a QCA interconnect (wire) requiring an additional clocking zone, so in total $4 + 1 + 4 = 9$ clocking zones are needed.

Synthesis results for a few benchmark circuits are provided and compared in terms of area and delay of their QCA implementations. Three benchmark circuits from [28] are chosen (their specifications can be found for completeness

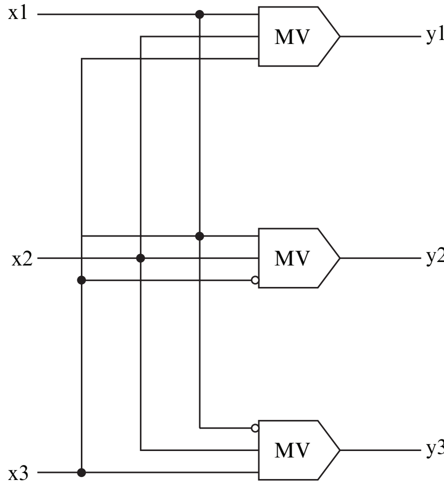


Fig. 9. Logic schematic diagram of QCA1

Table 4. Truth table of QCA2

$x1\ x2\ x3$	$y1\ y2\ y3$
000	001
001	000
010	011
011	101
100	010
101	100
110	111
111	110

in Appendix A). These circuits were synthesized using QCA1, QCA2 and Toffoli as gates with the addition of the so-called CNOT [28]. The CNOT gate implements the following logic functions

$$y1 = x1,$$

$$y2 = x1\ XOR\ x2 = x1x2' + x1'x2.$$

A QCA implementation of the CNOT gate is shown in Fig. 12. This implementation requires an area of 15×15 and has a delay of four clocking zones.

Two synthesis methods as applicable to reversible logic design, were used. The first method is a slight variation of the MV-based synthesis for threshold circuits proposed in [1]. The modified synthesis algorithm can be described as follows:

1. Step 1: Use the algorithm of [1] to synthesize the boolean function of the required circuit into a network of MVs and INVs.

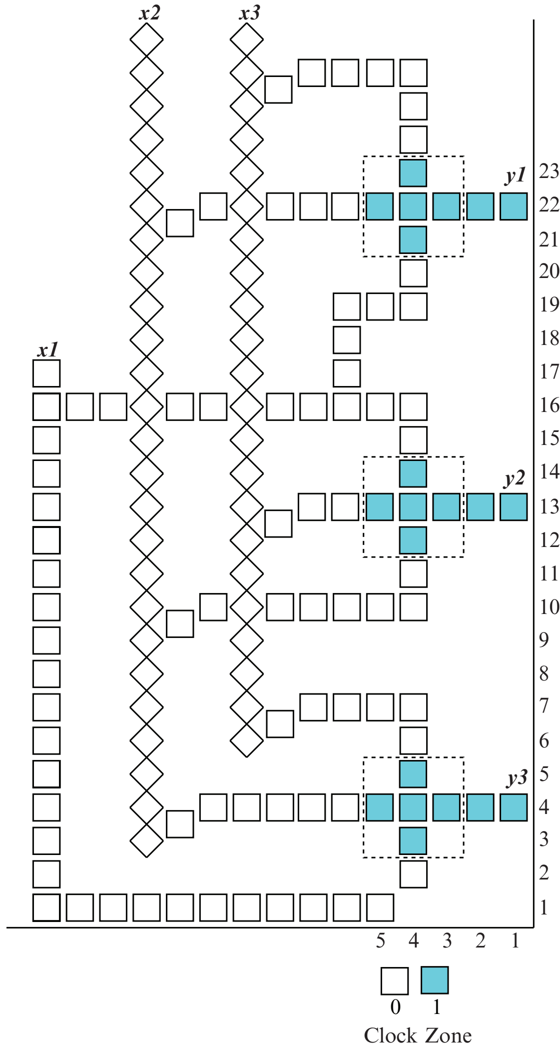


Fig. 10. QCA layout of QCA2

2. Step 2: Find the internal signals that appear more than once in the network from Step (1). These shared internal signals are implemented by fanout interconnections.
3. Step 3: Further simplify the circuit by reusing internal signals and substituting the appropriate primitives in the MV network with CNOT gates. For example, the benchmark *3_17* with three inputs (A, B and C) and three outputs (1, 2 and 3) is implemented as

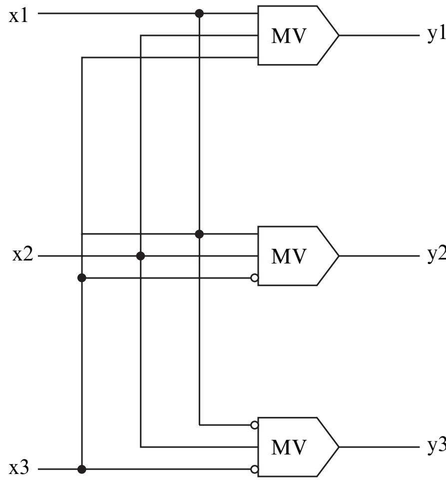


Fig. 11. Logic schematic diagram of QCA2

Table 5. Comparison between the four QCA reversible gates

	Fredkin	Toffoli	QCA1	QCA2
Clk zones	4	4	2	2
MVs	6	4	3	3
Area	30 × 18	31 × 18	27 × 15	27 × 15
Ctrl cells	6	2	0	0
Normal cells non-rotated rotated	185 122 63	167 140 27	146 100 46	147 100 47

$$\begin{aligned}1 &= (A' \; AND \; B') \; OR \; (B \; AND \; C)\\ &= MV[MV(A', B', 0), \; MV(B, C, 0), \; 1]\\ 2 &= A \; XOR \; B \; XOR \; C'\\ &= MV[C, \; MV(A, B', C'), \; MV(A', B, C')]\\ 3 &= (A' \; AND \; C') \; OR \; (A \; AND \; B)\\ &= MV[MV(A', C', 0), \; MV(A, B, 0), \; 1]\end{aligned}$$

Then, using the CNOT gate the function A XOR B XOR C' is implemented with a smaller number of gates (hence at a reduced area). Using the output signal 2 and CNOT gates, the output signals 1 and 3 can also be simplified. The simplified circuit is given by

$$\begin{aligned}1 &= (A' \; AND \; B') \; OR \; (B \; AND \; C)\\ &= C \; CNOT \; MV(B', \; 2, \; 0)\\ 2 &= A \; XOR \; B \; XOR \; C'\\ &= A \; CNOT \; B \; CNOT \; C'\end{aligned}$$

Table 6. Reversible gate implementation of 13 standard functions

Functions	Fredkin + INV			Toffoli + INV			QCA1		QCA2	
	# of Fre.	# of INV	clk zone	# of Tof.	# of INV	clk zone	# of QCA1	clk zone	# of QCA2	clk zone
$F_1 = AB'C$	2	0	9	2	0	9	2	5	2	5
$F_2 = AB$	1	0	4	1	0	4	1	2	1	2
$F_3 = A'BC + A'B'C'$	2	1	9	2	1	9	2	5	2	5
$F_4 = A'BC + AB'C'$	2	0	9	3	1	9	3	8	3	8
$F_5 = A'B + BC'$	2	0	9	2	0	9	2	5	2	5
$F_6 = AB' + A'BC$	2	0	9	3	0	9	3	5	3	5
$F_7 = A'BC$ $+ABC' + A'B'C'$	3	1	9	3	2	9	3	5	3	5
$F_8 = A$	1	0	4	1	0	4	1	2	1	2
$F_9 = AB + AC + BC$	3	1	9	4	0	14	1	2	1	2
$F_{10} = A'B + B'C$	1	0	4	3	0	9	3	5	3	5
$F_{11} = A'B$ $+BC + AB'C'$	3	1	9	1	0	4	4	5	4	5
$F_{12} = AB + A'B'$	1	1	4	1	0	4	2	5	2	5
$F_{13} = ABC' + A'B'C'$ $+AB'C + A'BC$	2	2	9	2	1	9	2	5	2	5

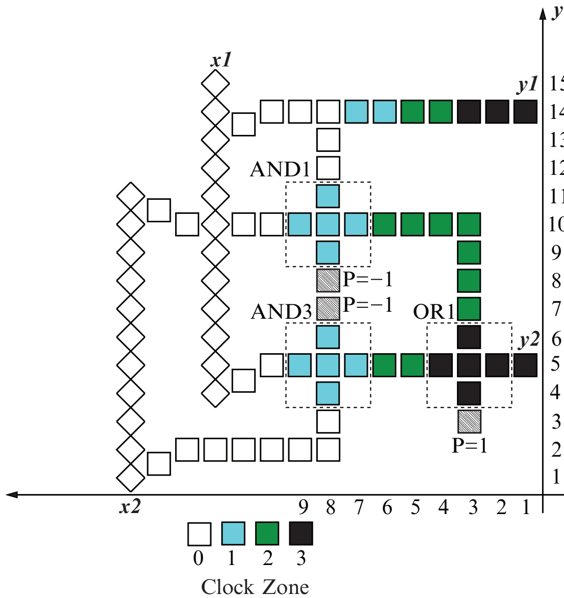


Fig. 12. QCA layout of the CNOT gate

$$3 = (A' \; AND \; C') \; OR \; (A \; AND \; B)$$
$$= B \; CNOT \; MV(A', 2, 0)$$

4. Step 4: Map all MVs with the same input signals into QCA1/QCA2 gates (as these gates can operate as three disjoint MVs).
   For example, benchmark *rd32* has three inputs (A, B and Cin) and two outputs (Sum and $Cout$); synthesis after the first three steps yields the following equations:

$$Sum = A \; XOR \; B \; XOR \; Cin$$
$$= MV[Cin', \; MV(A, B', Cin)$$
$$= MV(A', B, Cin)]$$
$$Cout = MV(A, B, Cin)$$

   Three MVs with input A, B and Cin (including the inverted signals) are required. These three MVs can be mapped into a single QCA1/QCA2 gate, e.g. for QCA1, the arrangement $x1 = A$, $x2 = Cin$, $x3 = B$ results in $y1 = MV(A, B, Cin)$, $y2 = MV(A, B', Cin)$ and $y3 = MV(A', B, Cin)$.
   It is in this final step that QCA1/QCA2 could be advantageous over the Toffoli/Fredkin gates, because in QCA1/QCA2 all MV outputs can be used (as disjoint devices differently from the Toffoli/Fredkin gates in which some of the outputs are simply a repetition of the inputs).

Note that this is not a true reversible logic synthesis method as in QCA fanouts and fixed polarization cells are allowed in the circuit.

For comparison purposes, synthesis results in [28] with the NCT library containing CNOT, Toffoli and INV is employed. The synthesis algorithm of [28] takes the number of so-called garbage signals as its major optimization goal, while the MV-based synthesis method of [1] is area-oriented. For fair comparison, benchmarks were also synthesized using the NCT library with the MV-based method. The modified version of the synthesis algorithm of [28] has an execution similar to the one described previously for QCA1/QCA2 and is given as follows:

1. Step 1: Synthesize the circuit using the algorithm of [28].
2. Step 2: There is no separate step involving the CNOT gate, because CNOT is already an integral part of the synthesis algorithm of [28].
3. Step 3: Use the fanout to parallelize the circuit (rather than the garbage-optimized, but serialized version) by decreasing the number of clocking zones.
   As an example, Fig. 13 shows the changes incurred in this step by benchmark *rd32*.

Note that in this algorithm there is no further adjustment as presented previously in the modified MV-based synthesis for QCA1/QCA2.

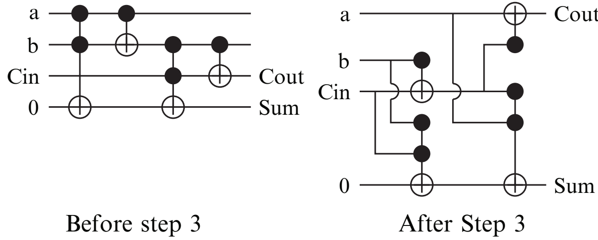


Fig. 13. Changes in rd32 due to step 3

Table 7. Benchmark synthesis results

Bench		3_17	mod5	rd32
[28] (Toff+CNOT)	Gar.	0	4	2
	gate	6	5	4
	A_{device}	1791	1570	1570
	A_{total}	31×90 =2790	34×81 =2754	37×77 =2849
	clk	21	17	16
Modified [28] (Toff+CNOT)	Gar.	4	4	6
	gate	4	3	4
	A_{device}	1566	1008	1566
	A_{total}	67×60 =4020	37×4 =1517	57×51 =2907
	clk	14	9	10
MV-based (QCA1+CNOT)	Gar.	12	8	4
	gate	7	5	2
	A_{device}	2835	2035	810
	A_{total}	62×86 =5332	32×35 =1120	33×41 =1353
	clk	18	7	6
MV-based (QCA2+CNOT)	Gar.	12	8	4
	gate	7	5	2
	A_{device}	2835	2035	810
	A_{total}	62×86 =5332	32×31 =992	33×41 =1353
	clk	18	7	6

The synthesis results are given in Table 7. As shown in Table 7, there are different figures of merit used in the comparison of the synthesis approaches: "Gar." denotes the number of garbage signals; "Gate" identifies the number of gates used in the QCA implementation; "A_{device}" is the rectangular area occupied by the active devices (QCA1, QCA2, Toffoli and CNOT), assuming the area of one QCA cell is 1; "A_{total}" denotes the rectangular area occupied by the whole QCA circuit, including interconnections between gates; "Clk" shows the number of clocking zone needed in the QCA implementation. Note

that INV is not considered as an active gate, but rather part of the interconnect, because in most cases the INV function can be achieved using an inverter chain, (as part of the interconnect).

The benchmark synthesis results confirm the findings established previously for the 13 standard functions. QCA1/QCA2 gates with CNOT have similar features as reflected in the figures of merit; compared with its original version, the modified synthesis algorithm of [28] results in a considerable reduction of clocking zones (albeit an increase in area and number of garbage signals occur). Noticeably, QCA1 and QCA2 show no improvement in terms of area but the number of clocking zones (and therefore operating speed of the synthesized circuit) is significantly lower than [28] with Toffoli gates.

6 Defect Analysis and Gate Testing

Recent developments in QCA manufacturing focus on molecular implementation, in which each QCA cell is a molecule. In manufacturing, defects can occur in both the *chemical synthesis* phase (in which the QCA cells are manufactured) and the *deposition* phase (in which the QCA cells are attached to a substrate) [29]. According to [29], defects are more likely to occur in the deposition phase than in the chemical synthesis phase, which results in perfectly manufactured cells imperfectly placed in the substrate. In this chapter, two types of defects are considered, namely the missing cell defect and the additional cell defect. The former represents the case where a cell fails to attach to the substrate while the latter represents the case of unwanted cell deposition.

The four reversible gates have been characterized in the presence of a *single* missing/additional cell defect. In QCADesigner an exhaustive test set (consisting of eight input test patterns) has been initially used in all cases:

- *Fredkin gate.* The results for the defects are shown in Table 8. Fourteen fault patterns are generated at the outputs.
- *Toffoli gate.* For this gate, the results are shown in Table 9. Thirteen fault patterns are generated at the outputs.
- *QCA1.* The results are shown in Table 10. Seven fault patterns are observed at the outputs.
- *QCA2.* The results are shown in Table 11. Again seven fault patterns are observed at the outputs.

Consider next the testing process of each reversible gate. The minimal test set that detects a single missing/additional cell defect with 100% coverage, has been established for each reversible gate using the simulation results.

- *Fredkin gate.* For a single Fredkin gate, the test has a cardinality of 3: $vy1y2 = a_1 = 001$; $vy1y2 = a_2 = 010$ and $vy1y2 = a_7 = 111$.
- *Toffoli gate.* The test set consists of three vectors: $x1x2x3 = a_3 = 011$, $x1x2x3 = a_4 = 100$ and $x1x2x3 = a_7 = 111$.

Table 8. Single missing/additional cell defect results for the Fredkin gate

Fault free outputs: v=0000,1111 y1=0101,0011 y2=0011,0101				
Missing cell	Faulty pattern	v	y1	y2
9,6		Correct	Correct	Correct
8,7	FP_1	Correct	Correct	0111,0101 (AND4 acts as horizontal wire)
8,6	FP_2	Correct	Correct	1111,1111 (AND output sa1)
8,5	FP_1	Correct	Correct	0111,0101 (AND4 acts as horizontal wire)
7,6		Correct	Correct	Correct
9,11		Correct	Correct	Correct
8,12	FP_3	Correct	Correct	1111,0101 (AND3 acts as horizontal wire)
8,11	FP_2	Correct	Correct	1111,1111 (AND2 output sa1)
8,10	FP_3	Correct	Correct	1111,0101 (AND3 acts as horizontal wire)
7,11		Correct	Correct	Correct
4,11		Correct	Correct	Correct
3,12	FP_4	Correct	Correct	0011,0000 (OR2 acts as horizontal wire)
3,11	FP_4	Correct	Correct	0011,0000 (OR2 acts as horizontal wire)
3,10	FP_4	Correct	Correct	0011,0000 (OR2 acts as horizontal wire)
2,11		Correct	Correct	Correct
3,6	FP_5	Correct	Correct	1111,1010 (extra INV AND4→OR2)
8,3	FP_6	Correct	Correct	0111,0000 (extra INV u→AND4)
8,14	FP_7	Correct	Correct	1100,0101 (extra INV x1→AND3)
9,20		Correct	Correct	Correct
8,21	FP_8	Correct	1111,0011 (AND2 acts as horizontal wire)	Correct
8,10	FP_9	Correct	1111,1111 (AND2 output sa1)	Correct
8,19	FP_8	Correct	1111,0011 (AND2 acts as horizontal wire)	Correct
7,20		Correct	Correct	Correct
9,25		Correct	Correct	Correct
8,26	FP_{10}	Correct	0111,0011 (AND1 acts as horizontal wire)	Correct
8,25	FP_9	Correct	1111,1111 (AND1 output sa1)	Correct
8,24	FP_{10}	Correct	0111,0011 (AND1 acts as horizontal wire)	Correct

Table 8. Continued

Fault free outputs: v=0000,1111 y1=0101,0011 y2=0011,0101				
Missing cell	Faulty pattern	v	y1	y2
7,25		Correct	Correct	Correct
4,20		Correct	Correct	Correct
3,21	FP_{11}	Correct	0101,0000 (OR1 acts as horizontal wire)	Correct
3,20	FP_{11}	Correct	0101,0000 (OR1 acts as horizontal wire)	Correct
3,19	FP_{11}	Correct	0101,0000 (OR1 acts as horizontal wire)	Correct
2,20		Correct	Correct	Correct
3,25	FP_{12}	Correct	1111,1100 (extra INV AND1→OR1)	Correct
8,18	FP_{13}	Correct	1010,0011 (extra INV x2→AND2)	Correct
8,28	Fp_{14}	Correct	0111,0000 (extra INV u→AND1)	Correct
Additional cell	Fault pattern	v	y1	y2
9,26		Correct	Correct	Correct
9,24		Correct	Correct	Correct
7,26		Correct	Correct	Correct
7,24		Correct	Correct	Correct
4,7		Correct	Correct	Correct
9,4		Correct	Correct	Correct

- *QCA1.* The test set consists of three vectors: $x1x2x3 = a_1 = 001$, $x1x2x3 = a_3 = 011$, $x1x2x3 = a_5 = 101$
- *QCA2.* The test set consists of three vectors: $x1x2x3 = a_0 = 000$, $x1x2x3 = a_2 = 010$, $x1x2x3 = a_4 = 100$.

Note that in cases of weak polarized outputs, signals are restored by QCA lines in the next clocking zone.

7 Reversible 1D Array Testing

In this section, testing of a 1D unilateral ILA made of N identical modules[1] is considered. In this array configuration, there is *no direct* controllability and observability of the intermediate modules. *Constant testability* (C-testability)

[1] Hereafter, the logic cell in the array is referred to as "module" to differ from the QCA cell in the previous discussion. Also in the following presentation, modules are made of reversible gates such as presented in previous sections.

Table 9. Single missing/additional cell defect results for the Toffoli gate

Fault free outputs: y1=0001,1110 y2=0011,0011 y3=0101,0101				
Missing cell	Faulty pattern	y1	y2	y3
9,24		Correct	Correct	Correct
8,25	FP_1	0001,1111(AND acts as horizontal wire)	Correct	Correct
8,24	FP_2	1111,1111(AND gate output sa1)	Correct	Correct
8,23	FP_1	0001,1111(AND acts as horizontal wire)	Correct	Correct
7,24		Correct	Correct	Correct
9,15		Correct	Correct	Correct
8,16	FP_3	0000,1110(MV1 acts as horizontal wire)	Correct	Correct
8,15	FP_4	1000,1110(MV1 performs MV(x2',x1,x3')	Correct	Correct
8,14	FP_3	0000,1110(MV1 acts as horizontal wire)	Correct	Correct
7,15		Correct	Correct	Correct
4,15		Correct	Correct	Correct
3,16	FP_5	0001,0111(MV2 acts as horizontal wire)	Correct	Correct
3,15	FP_5	0001,0111(MV2 acts as horizontal wire)	Correct	Correct
3,14	FP_5	0001,0111(MV2 acts as horizontal wire)	Correct	Correct
2,15		Correct	Correct	Correct
9,6		Correct	Correct	Correct
8,7	FP_6	0001,0010(OR acts as horizontal wire)	Correct	Correct
8,6	FP_7	0000,0010(OR output sa0)	Correct	Correct
8,8	FP_6	0001,0010(OR acts as horizontal wire)	Correct	Correct
7,6		Correct	Correct	Correct
3,24	FP_8	1111,0101(extra INV in AND→MV2)	Correct	Correct
2,5	FP_9	0000,0011(extra INV in OR→MV2)	Correct	Correct
8,9	FP_{10}	0001,0011(extra INV in x2→OR)	Correct	Correct
11,12	FP_{10}	0001,0011(extra INV in x2→OR)	Correct	Correct
8,12	FP_{11}	0100,1100(extra INV in x2→MV1)	Correct	Correct
8,18	FP_{12}	0010,1010(extra INV in x3→MV1)	Correct	Correct
11,18	FP_{12}	0010,1010(extra INV in x3→MV1)	Correct	Correct
8,21	FP_{13}	0001,0101(extra INV in x3→AND)	Correct	Correct
Additional cell	v	y1	y2	
11,9	FP_{10}	0001,0011(extra INV in x2→OR)	Correct	Correct
11,21	FP_{13}	0001,0101(extra INV in x3→AND)	Correct	Correct

in the presence of at most one faulty module (given by a reversible gate under the defect model presented previously) is initially considered. C-testability refers to the property by which the number of vectors for testing the 1D array is independent of N. The previously presented defect analysis of reversible QCA gates and corresponding testability can be extended to a 1D array whose basic module is one of the reversible gates.

Table 10. Single missing cell defect results for QCA1

Fault free outputs: y1=0001,0111 y2=0010,1011 y3=0111,0001				
Missing cell	Faulty pattern	y1	y2	y3
4,3	FP_1	Correct	Correct	0011,0011(y3=x2)
5,4		Correct	Correct	Correct
4,4	FP_2	Correct	Correct	0010,1011($x1' \rightarrow x1$, $x3 \rightarrow x3'$)
3,4		Correct	Correct	Correct
2,4		Correct	Correct	Correct
4,5	FP_1	Correct	Correct	0011,0011(y3=x2)
4,12	FP_3	Correct	0011,0011(y2=x2)	Correct
5,13	FP_4	Correct	0000,1111(y2=x1)	Correct
4,13	FP_5	Correct	0111,0001($x1 \rightarrow x1'$, $x3' \rightarrow x3$)	Correct
3,13		Correct	Correct	Correct
2,13		Correct	Correct	Correct
4,14	FP_3	Correct	0011,0011(y2=x2)	Correct
4,21	FP_6	0011,0011(y1=x2)	Correct	Correct
5,22		Correct	Correct	Correct
4,22	FP_7	1011,0010($x1 \rightarrow x1'$, $x3 \rightarrow x3'$)	Correct	Correct
3,22		Correct	Correct	Correct
2,22		Correct	Correct	Correct
4,23	FP_6	0011,0011(y1=x2)	Correct	Correct

Table 11. Single missing cell defect results for QCA2

Fault free outputs: y1=0001,0111 y2=0010,1011 y3=1011,0010				
Missing cell	Faulty pattern	v	y1	y2
4,3	FP_1	Correct	Correct	0011,0011(y3=x2)
5,4		Correct	Correct	Correct
4,4	FP_2	Correct	Correct	0001,0111($x1' \rightarrow x1$, $x3' \rightarrow x3$)
3,4		Correct	Correct	Correct
2,4		Correct	Correct	Correct
4,5	FP_1	Correct	Correct	0011,0011(y3=x2)
4,12	FP_3	Correct	0011,0011(y2=x2)	Correct
5,13	FP_4	Correct	0000,1111(y2=x1)	Correct
4,13	FP_5	Correct	0111,0001($x1 \rightarrow x1'$, $x3' \rightarrow x3$)	Correct
3,13		Correct	Correct	Correct
2,13		Correct	Correct	Correct
4,14	FP_3	correct	0011,0011(y2=x2)	Correct
4,21	FP_6	0011,0011(y1=x2)	Correct	Correct
5,22		Correct	Correct	Correct
4,22	FP_7	1011,0010($x1 \rightarrow x1'$, $x3' \rightarrow x3$)	correct	correct
3,22		Correct	Correct	Correct
2,22		Correct	Correct	Correct
4,23	FP_6	0011,0011(y1=x2)	Correct	Correct

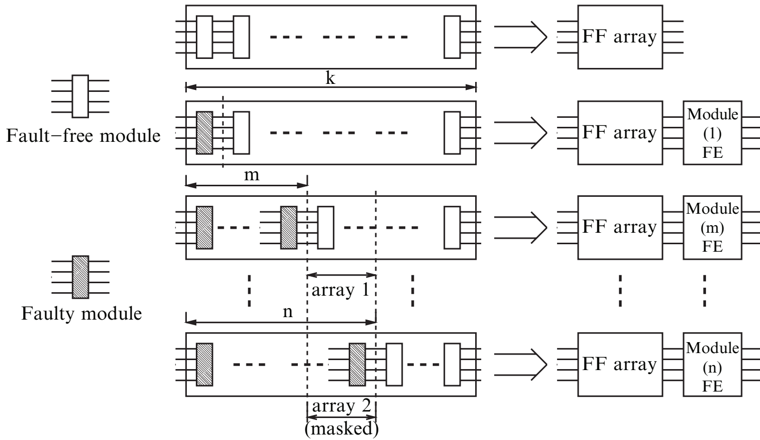


Fig. 14. One-to-one onto mapping and fault masking

Initially, the one-to-one onto mapping of the reversible gate will be analyzed in detail. It shows that if a faulty reversible gate has a one-to-one onto mapping (albeit different from the fault free one), then masking can occur after n iterations in a 1D array. Consider an array divided into two subarrays: the first subarray consists of only faulty modules (all modules are faulty with a pattern given by a one-to-one onto mapping different from the fault free one), while the second subarray is fault free.

This array can be modelled as equivalent to a fault-free k-module array and an additional fault-equivalent (FE) module, as shown in Fig. 14. Consider the function of the whole array, while increasing the length of the faulty subarray from 1 to k. The function of FE module corresponding to the array with n faulty modules is denoted by $FE^{(n)}$. As the possible function of the FE module is finite, for k sufficiently large, there must be a number n ($n < k$) such that $FE^{(n)}$ equals to some $FE^{(m)}$ with $m < n$. So, the n-module faulty array has the same function as the m-module faulty array ($m < n < k$). Observe the array 1 and array 2 in the figure. They receive same input and produce same output when the whole k-module array is tested. Because the input to k-module array is exhaustive and modules in the array have one-to-one onto function, array 1 and array 2 are tested exhaustively. So, array 1 (which is faulty-free) cannot be differentiated from array 2 (which has $m - n$ faulty modules). Fault mask exists where there are $m-n$ concatenating faulty modules.

Consider a 3-input reversible gate as module; the number of possible FE functions in a module is given by $(2^3)! = 40,320$, i.e. a very long array is required to exhaust all output combinations and guarantee the repetition of the FE function for undetection due to masking. If a reversible gate has many inputs, then the probability of masking is further reduced. However, for the considered gates this can be significant. Consider for example QCA1, FP_7

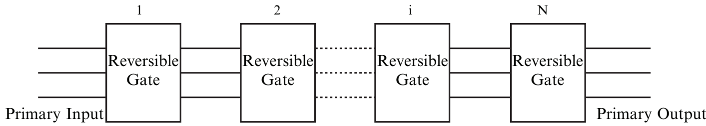


Fig. 15. 1D array of modules made of reversible logic gates

(missing cell 4,22) results in a faulty one-to-one onto function. Let this faulty function be denoted as F and the fault free function by FF. An array of 12 F modules behaves the same as an array of 12 FF modules under any input vector. So, fault masking occurs. A similar problem occurs for QCA2 because FP_5 (missing cell 4,13) results in a one-on-one onto function. These results show that if the fault patterns are one-to-one onto mappings, masking will occur. An analysis of this type of faults will be pursued in a later section.

C-testability in the presence of single and multiple faulty modules can be then analyzed for the 1D array.

- *Fredkin gate*, Consider a 1D array of N concatenated Fredkin gates as shown in Fig. 15. As established previously, the input test set consists of the three vectors: a_1, a_2 and a_7. If these vectors are applied to a fault free module, the output values are a_2, a_1 and a_7. Therefore, in the 1D array the test vectors at the primary inputs can be regenerated internally by the fault free modules. Assume there is at most one faulty module in the 1D array. The vector set a_1, a_2, a_7 is sufficient for detection, this can be proved as follows. Every fault free module located prior to the faulty module will regenerate each input test vector; so a_1, a_2, a_7 will also be applied to the faulty module. As this test set has 100% coverage, the faulty module will produce erroneous outputs. Moreover, as every fault free module located after the faulty module has a function given by a one-to-one onto mapping of the inputs to the outputs, then propagation from the faulty module to the primary outputs is guaranteed. Thus, the fault is detected.
  However, when multiple faulty modules are present in the array, masking can occur. As shown in Table 8, the two patterns FP_7 and FP_{13} result in one-to-one onto mappings, i.e. different permutations between the inputs and outputs. If a module with FP_7 is followed by a module with FP_{13}, then also the application of a fully exhaustive test set will result in an output pattern that is the same as for a concatenation of two fault free modules, thus this fault is not detected.
- *Toffoli gate*. Consider a 1D array of N concatenated modules made of Toffoli gates, as shown in Fig. 15. It will be proved next that the minimal test set a_3,a_4,a_7 is sufficient for detection even in the presence of multiple faulty modules. The mappings of a fault free module and a faulty module are provided in Table 12. When applying the vectors a_3, a_4, a_7, a fault free module regenerates these vectors. If the 1D array has only fault free modules, then at the primary outputs the pattern is a_7, a_4, a_7 (for N even)

Table 12. Outputs of fault free and faulty Toffoli gates

Input vector	Fault free output	FP_1	FP_2	FP_3	FP_4	FP_5	FP_6
a_0	a_0	a_0	a_4	a_0	a_4	a_0	a_0
a_1	a_1	a_1	a_5	a_1	a_1	a_1	a_1
a_2	a_2	a_2	a_6	a_2	a_2	a_2	a_2
a_3	a_7	a_7	a_7	a_3	a_3	a_7	a_7
a_4	a_4	a_4	a_4	a_4	a_4	a_0	a_0
a_5	a_5	a_5	a_5	a_5	a_5	a_5	a_1
a_6	a_6	a_6	a_6	a_6	a_6	a_6	a_6
a_7	a_3	a_7	a_7	a_3	a_3	a_7	a_3
FP_7	FP_8	FP_9	FP_{10}	FP_{11}	FP_{12}	FP_{13}	
a_0	a_4	a_0	a_4	a_0	a_4	a_0	
a_1	a_5	a_1	a_5	a_5	a_1	a_1	
a_2	a_6	a_2	a_6	a_2	a_6	a_2	
a_3	a_7	a_3	a_7	a_3	a_3	a_7	
a_0	a_0	a_0	a_0	a_4	a_4	a_0	
a_1	a_5	a_1	a_1	a_5	a_1	a_5	
a_6	a_2	a_6	a_6	a_2	a_6	a_2	
a_3	a_7	a_7	a_7	a_3	a_3	a_7	

and a_3, a_4, a_7 (for N odd). The fault patterns can be categorized into two types: *type I* includes FP_6 and FP_9, *type II* includes all other fault patterns. As shown in Table 12, when a_3, a_7 are applied at the inputs, faulty modules of type II will generate the same values at the outputs (for example, a module with FP_4 will generate a_3 and a_3). When a_3 and a_7 are applied, a fault free module (faulty module of type I) will generate at the outputs a_3 and a_7 (a_7 and a_3). Consider initially the scenario when there is at least one faulty module of type II in the array. Let the type II faulty module located closest to the primary inputs be denoted by A. Then, from the primary inputs to A, there can only be fault free, or type I faulty modules. Therefore, a_3 and a_7 will be applied to A, thus generating two identical patterns at the outputs of A. For either a faulty, or fault free module the number of distinct output patterns can not be greater than the number of distinct input patterns; so the fault will be propagated to the primary outputs of the array, i.e. detection is accomplished.

A different situation occurs when all faulty modules are of type I. Let the type I faulty module closest to the primary inputs be denoted by B. Then, from the primary inputs to B, there can be only fault free modules. After applying the vector set, a_4 is applied to B; this is mapped to a_0 by B. Since the array consists of only type I and fault free modules, then in the modules following B, a_0 will always be mapped to a_0. So, at the primary outputs of the array, a_0 will be observed (instead of the expected a_4) and thus, detection is achieved.

Table 13. Outputs of fault free and faulty QCA1

Input vector	Fault free output	FP_1	FP_2	FP_3	FP_4	FP_5	FP_6	FP_7
a_0	a_0	a_0	a_0	a_0	a_0	a_0	a_0	a_4
a_1	a_1	a_0	a_0	a_1	a_1	a_3	a_1	a_1
a_2	a_3	a_3	a_3	a_3	a_1	a_3	a_7	a_7
a_3	a_5	a_5	a_4	a_7	a_5	a_7	a_5	a_5
a_4	a_2	a_2	a_3	a_0	a_2	a_0	a_2	a_2
a_5	a_4	a_4	a_4	a_4	a_6	a_4	a_0	a_0
a_6	a_6	a_7	a_7	a_6	a_6	a_4	a_6	a_6
a_7	a_7	a_7	a_7	a_7	a_7	a_7	a_7	a_3

The C-testability of a 1D array made of Toffoli gates is established, because none of the faulty modules will generate a fault pattern that is a permutation in itself. Hence, detection in the presence of multiple faulty modules is always possible.

– *QCA1.* Consider detection in an array made of QCA1 modules under the single faulty module assumption; the test set a_1, a_2, a_3 can detect any single faulty module in the array. As shown in Table 13, a_1, a_2/a_5 and a_3/a_4 combined are sufficient to test all possible faults of QCA1. a_1 can be regenerated by the fault free module. When applying a_2 to the primary inputs of the array, a_2 or a_5 will be applied to odd numbered modules and a_3 or a_4 to the even numbered modules. When applying a_3 to the array, a_2 or a_5 will be applied to even numbered modules and a_3 or a_4 to the odd numbered modules. Thus, 100% coverage of single faulty module can be guaranteed.
  Under a multiple faulty module assumption, FP_1, FP_2, FP_3, FP_4, FP_5 and FP_6 are irreversible faults, so the number of possible combinations at the outputs is less than eight, therefore an exhaustive test can always accomplish detection irrespective of the number of faulty modules. However, for FP_7, the faulty function is a one-to-one onto mapping, and this fault can be masked by multiple occurrence of itself in the array.
– *QCA2.* Under a single faulty module assumption, complete full coverage can be achieved by the test set a_0, a_1, a_2 and a_3. As shown in Table 17, a_0, a_2/a_5 and a_3/a_4 can fully test QCA2. When applying a_0 to the primary inputs of the array, a_0 is regenerated at the inputs of all odd numbered modules. When a_1 is provided as input to the array, a_0 will be regenerated at the inputs of all even numbered modules. Similarly for a_2 (a_3) at the primary inputs of the array, a_2 or a_5 (a_4) will be regenerated for the odd (even) numbered modules and a_3 or a_4 (a_5) for the even (odd) numbered modules. Therefore, every module in the array is fully tested.
  Detection in the presence of multiple faulty modules in a 1D array made of QCA2 can not be guaranteed. For most defects (such as defects 1, 3, 6, 7, 8, 12, 13, 15 and 18), the number of possible combinations of the output

signals is less than 8, so they are always testable. However, for defect 9 (missing cell 4,13), the faulty function is a one-to-one onto mapping, so it can be masked by multiple appearances of itself in the array.

8 Array Testability

In this section, the testability of a 1D ILA made of reversible modules is analyzed. In such analysis. It is assumed that an exhaustive test set will be applied to the primary inputs; controllability and observability must be established to guarantee that C-testability can be accomplished. In a 1D array (as shown in Fig. 16), controllability is not difficult, i.e. in the absence of a fault, if the leftmost module is provided with all possible input combinations during the testing process, then all modules in the array will also receive all possible input combinations. If any module in the array has an *irreversible fault* (i.e. a fault that change the reversible function of a module into an irreversible function), then the erroneous output due to this fault can always be propagated to the primary outputs of the array (provided all possible combinations are applied at the primary inputs). So, *any number* of irreversible faults can be observed and multiple fault detection is accomplished.

Consider in more detail only the faults in the reversable modules of the array (connection between modules are assumed to be fault free). Different fault models can be considered for the modules:

- The traditional Stuck-At Fault (*SAF*), or line Bridging Fault (*BF*) are *irreversible faults*, i.e. faults that will result in an irreversible function. Therefore, they can be detected with a constant number of tests. The array is C-testable and no additional control, or observable lines are required to the modules. However, these fault models are not sufficient for modeling the faults in QCA [21].
- Consider an arbitrary functional fault (*AFF*) model in which it is assumed that a fault may cause an arbitrary change in the truth table of the circuit. If there is any number of faulty modules whose function is not a one-to-one onto mapping (i.e yielding irreversible faults), then the number of possible output combinations of the array will be less than an exhaustive combination (for three inputs this number is less than 8). This scenario (irrespective of the number of faulty modules) can be detected at the primary outputs. So, only those *reversible faults* (faults that will result

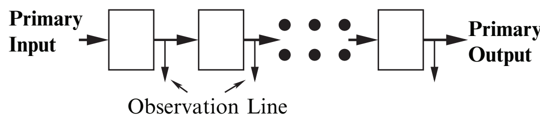


Fig. 16. 1D array of reversible module

in a reversible gate function different from the fault-free one) must be considered.

– The most comprehensive fault model is he so-called arbitrary reversible fault model (*ARF*). In ARF, a fault is assumed to change the truth table of the gate as long as the resulting function is still reversible. The question is, is it possible to achieve C-testability if observability is added such that outputs of intermediate modules are made directly observable? Unfortunately, it will be shown next through an example that even if a subset of the ARF model is considered, then the array is not C-testable.
  Consider the so-called Single Pin Inversion (*SPI*) model as a subset of the ARF model. In the SPI model, every module has at most one fault in one of its input/output pins, such that an inversion of the signal occurs at that pin. The inversion fault is very common in QCA circuits, because it results from misalignment of QCA cells [21].
  As shown in Fig. 17, if a faults is present between the nth output pin of the kth module and the nth input pin of the $(k+1)$th module, then detection can not be guaranteed unless it is possible to directly observe the internal pins (in this case pin n) between these two modules. The faulty pin can be any one of the module pins, so we have to observe every internal connection to detect all SPI faults, which is a pathological case of an array as a fully observable system.

So, any generalized fault model cannot be used to fully assess the C-testability of reversible arrays under multiple faults. A case-by-case study of different logic gates (as making up a module) together with their specific input/output functions and fault patterns, is required. All reversible faults (and corresponding functions) in a module must be considered to make an array C-testable by selecting those lines in a module for observability. Three conditions are proposed for selecting lines for observability in each module of an array:

1. *Rule 1*: If all reversible faults change the entries of an output in the truth table of a module, then this output signal should be a primary observable line.

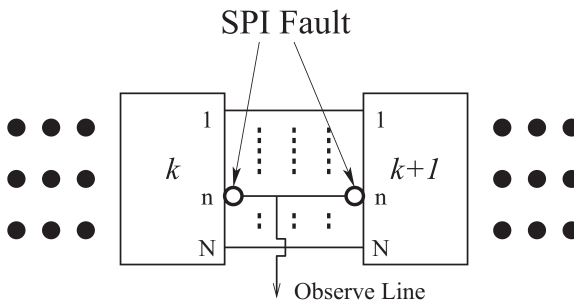


Fig. 17. SPI fault

Table 14. Example for rule 1

Input	Fault free	FP_1	FP_2
$x_1x_2x_3$	$y_1y_2y_3$	$y_1y_2y_3$	$y_1y_2y_3$
0 0 0	0 0 0	0 0 **1**	0 0 0
0 0 1	0 0 1	0 0 **0**	0 0 1
0 1 0	0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1	0 1 1
1 0 0	1 0 0	1 0 0	1 **1** **1**
1 0 1	1 0 1	1 0 1	1 0 **0**
1 1 0	1 1 1	1 1 1	1 **0** 1
1 1 1	1 1 0	1 1 0	1 1 0

Table 15. Example for rule 2

Input	Fault free	FP_1	FP_2	FP_3
$x_1x_2x_3$	$y_1y_2y_3$	$y_1y_2y_3$	$y_1y_2y_3$	$y_1y_2y_3$
0 0 0	0 0 0	0 0 **1**	0 0 0	0 0 0
0 0 1	0 0 1	0 0 **0**	0 0 1	0 0 1
0 1 0	0 1 0	0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1	0 1 1	0 1 1
1 0 0	1 0 0	1 0 0	1 **1** **1**	1 **1** 0
1 0 1	1 0 1	1 0 1	1 0 **0**	1 0 1
1 1 0	1 1 1	1 1 1	1 **0** 1	1 1 1
1 1 1	1 1 0	1 1 0	1 1 0	1 **0** 0

For example, consider a reversible module with fault free and fault patterns given in Table 14. It has two reversible fault patterns: FP_1 affects the output y_3, FP_2 affects the outputs y_2 and y_3. By Rule 1, the truth table of y_3 is changed by all reversible faults, so if y_3 is made a primary observable line in the modules, then the array is C-testable.

The application of Rule 1 to QCA1 and QCA2 results in a C-testable array by making one observable line as primary in each module. As shown in Fig. 21, the arrays can be fully tested by applying the exhaustive test set (of cardinality 8) at the primary inputs. For the QCA1 array, the only reversible fault pattern is FP_7 and it modifies the output $y3$ of a faulty module. So by observing this output, detection will occur at the faulty module. Similarly, the QCA2 array is C-testable by making the output $y2$ as a primary observable line in each module.

2. *Rule 2*: If all reversible faults can be propagated to a module output prior to masking, then such output should be a primary observable line.

   In Table 15, a reversible module and a fault pattern are shown as an example. The module has three reversible fault patterns: FP_1 changes the output y_3, FP_2 changes the outputs y_2 and y_3, FP_3 changes y_2. FP_1 and FP_2 can be propagated to y_3 of the faulty module. FP_3 can be propagated to y_3 of the module after the faulty one, independently of the status (faulty

or fault free) of this module. By Rule 2, the selection of y_3 as an observable line for each module makes the array C-testable.

3. *Rule 3*: If the above conditions cannot be satisfied, two or more observable lines are needed in each module; each of these lines will be required to observe part of the possible fault patterns and the union of the covered fault patterns must be the entire possible fault pattern set.

   By applying Rules 1 and 2, it is possible to observe different module outputs to detect different fault patterns. In the general case, the problem of selecting multiple output lines for observability is a set covering problem. Let the outputs lines of a reversible module be $y_1, y_2, \ldots, y_n$. The possible fault patterns of each module are denoted by $FP_1, FP_2, \ldots, FP_m$. By observing an output line y_i, a group of fault patterns can be detected, i.e. y_i "covers" a group of fault patterns. Thus, the problem is equivalent to selecting the minimum cardinality set of output lines such that all fault patterns are covered, which is the definition of a set covering problem [34]. The set covering problem is NP hard. However, heuristic algorithms based on greedy criteria [34] can be used to solve it.

   Consider the Fredkin gate for example; there are two reversible fault patterns, FP_7 and FP_{13}. FP_7 changes the output $y3$ and FP_{13} changes the output $y2$; so, Rule 1 cannot establish a primary observable line for both fault patterns. The output of two adjacent modules with FP_7 and FP_{13} will result in masking; hence, Rule 2 cannot be used. By applying Rule 3, $y1$ and $y2$ are selected as multiple observable lines, because all possible reversible fault patterns are detected.

Each of the above three rules gives a sufficient condition for constructing a C-testable array. Rule 1 requires only a primary observable line and is relatively easy to assess, so in general it is the preferable condition. Rule 2 requires also one observable line, but propagation under different multiple fault patterns must be established. Therefore, it can be applied if Rule 1 fails. Rule 3 requires more than one observable lines and may account for a large overhead. Rule 3 however, can be applied following an analysis of Rules 1 and 2 for fault detection. Hence, realistically Rule 3 should be applied after Rules 1 and 2 have been unsuccessful. A combination of Rules 1 and 2 provides the necessary condition for constructing a C-testable array with only one primary observable line per module. Rule 3 can ensure C-testability in an array as long as a large overhead is acceptable.

9 Array Controllability and Observability

In this section, a different 1D array configuration is analyzed; controllability and observability are introduced in the intermediate modules to enhance coverage of the fault detection process. In this configuration, intermediate modules are provided with one primary input and one primary output, both

in the vertical direction. As proved in a previous section, a 1D array with Toffoli gates is already C-testable in the presence of multiple faulty modules, hence this case will not be considered. Moreover differently from the previous section, a non fully exhaustive test set (albeit with 100% coverage of the modelled faults) is utilized.

- *Fredkin gate.* It has been shown that fault masking exists in an 1D array such as the one shown in Fig. 15. However, the 1D array of Fig. 18a is C-testable under the assumption that multiple modules may be faulty (each faulty module can only have a fault as described in Sect. 6).
  In the array of N modules, additional controllability is provided by setting the u input of each module as a primary (vertical) input; additional observability is obtained by setting the $y2$ output of each module as a primary (vertical) output. Let the primary (horizontal) inputs $x1, x2$ of the first module in the array be denoted as PI_1, PI_2. Let the ith module in the array be G_i, the input u of G_i be denoted as U_i.
  The mapping between inputs and outputs of a fault free module as well as a faulty module (by using FP_i) is shown in Table 16. The fault patterns are categorized into two types: (1) FP_1, FP_8, FP_9, FP_{10}, FP_{11}, FP_{12}, FP_{13} and FP_14 are type I; (2) FP_2, FP_3, FP_4, FP_5, FP_6 and FP_7 are type II.
  The test vector set that detects multiple faulty modules in an array is, as follows:
  1. *First Test Vector*: $PI_1 = 0, PI_2 = 1$, $U_i = 0$ for all i. If the array is fault free, then all modules in the array will receive input vector $ux1x2 = 001 = a_1$. The expected (fault free) output at $y2$ of any module in the array is "0".
  2. *Second Test Vector*: $PI_1 = 1, PI_2 = 1$, $U_i = 1$ for all i. If the array is fault free, then all modules in the array will receive as input $ux1x2 = 111 = a_7$. The expected (fault free) output at $y2$ of any module is "1".
  3. *Third Test Vector*: $PI_1 = 1, PI_2 = 0$, $U_i = 0$ for $i = 1, 3, 5, 7, \dots$ and $U_i = 1$ for $i = 2, 4, 6, \dots$, as shown in Fig. 18b. If the array is fault free, the input vector $ux1x2 = 010 = a_2$ is applied to all G_i (for i an odd integer). The expected output at $y2$ of G_i is "1" for odd i and "0" for even i.
  4. *Fourth Test Vector*: $PI_1 = 0, PI_2 = 0$, $U_i = 1$ for $i = 1, 3, 5, 7, \dots$ and $U_i = 0$ for $i = 2, 4, 6, \dots$, as shown in Fig. 18b. If the array is fault free, $ux1x2 = 010 = a_2$ is applied as input vector to all G_i (for i as an even integer). The expected output at $y2$ of G_i is "0" for odd i and "1" for even i.

  Initially, it will be shown that if the array contains any (single or multiple) type II fault pattern(s), detection is accomplished by observing $y2$ as primary output of the modules using the first and second test vectors. When applying vector 1, all $U_i = 0$ and $v = u$ for all modules, every module (either faulty or fault free) will have as input $x1 = 0$. Therefore, any module

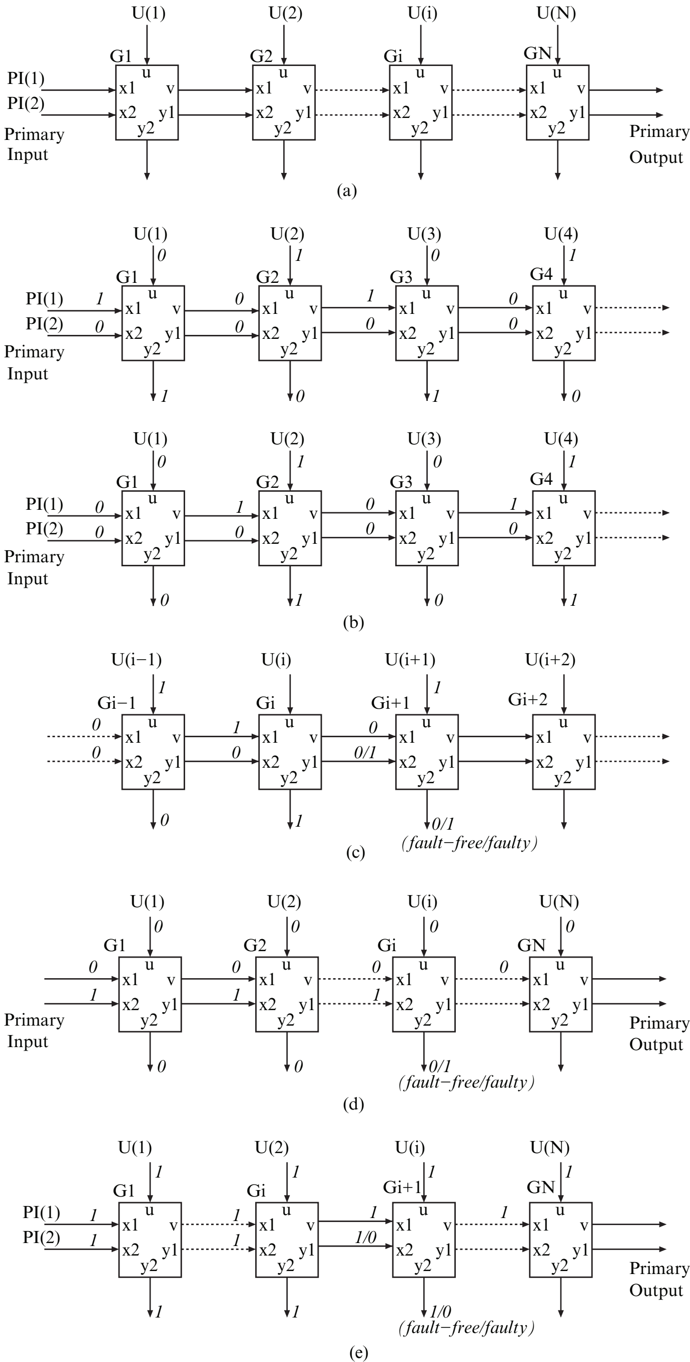


Fig. 18. C-testability of one-dimensional array (made of Fredkin gates) with increased observability and controllability

Table 16. Outputs of fault free and faulty Fredkin gates

Input vector	Fault free output	FP_1	FP_2	FP_3	FP_4	FP_5	FP_6
a_0	a_0	a_0	a_0	a_1	a_0	a_1	a_0
a_1	a_2	a_3	a_3	a_3	a_2	a_3	a_3
a_2	a_1	a_1	a_1	a_1	a_1	a_1	a_1
a_3	a_3	a_3	a_3	a_3	a_3	a_3	a_3
a_4	a_4	a_4	a_5	a_4	a_4	a_5	a_4
a_5	a_5	a_5	a_5	a_5	a_4	a_4	a_4
a_6	a_6	a_6	a_7	a_6	a_6	a_7	a_6
a_7	a_7	a_7	a_7	a_7	a_6	a_6	a_6
FP_7	FP_8	FP_9	FP_{10}	FP_{11}	FP_{12}	FP_{13}	FP_{14}
a_1	a_2	a_2	a_0	a_0	a_2	a_2	a_0
a_3	a_2	a_2	a_2	a_2	a_2	a_0	a_2
a_0	a_3	a_3	a_3	a_1	a_3	a_3	a_3
a_2	a_3	a_3	a_3	a_3	a_3	a_1	a_3
a_4	a_4	a_6	a_4	a_4	a_6	a_4	a_4
a_5	a_5	a_7	a_5	a_5	a_7	a_5	a_5
a_6	a_6	a_6	a_6	a_4	a_4	a_6	a_4
a_7	a_7	a_7	a_7	a_5	a_5	a_7	a_5

in the array has $ux1 = 00$; so, if a module with fault pattern FP_2,FP_3,FP_5 or FP_7 is present, then the $y2$ output of that module will be "1" instead of the expected "0". Thus, these fault patterns will be detected by vector 1. Similarly, when applying vector 2, all $U_i = 1$ and $PI_1 = 1, PI_2 = 1$; therefore, each module in the array will have input $ux1 = 11$. If a module with fault pattern FP_4 or FP_6 is present, the $y2$ output of that module is "0" (instead of the expected "1"). Thus by applying vector 1 and vector 2, any number of type II fault patterns can be detected.
Now consider the case in which the array contains faulty modules with only type I fault patterns. Let the faulty module that is closest to the primary inputs be G_i, i.e. only fault free modules exist from the primary inputs to G_i. G_i must have one of the type I fault patterns, i.e. FP_1, FP_8, FP_9, FP_{10}, FP_{11}, FP_{12}, FP_{13} or FP_{14}. If i is odd, the third vector $ux1x2 = 010$ to G_i is applied; if i is even, then the fourth vector $ux1x2 = 010$ to G_i is applied. In both cases, G_i will have as inputs $ux1x2 = 010$ and the expected output is $vy1y2 = 001$; therefore, G_{i+1} is expected to have as inputs $ux1x2 = 100$ and generate a "0" at the output $y2$, as shown in Fig. 18c. If G_i has fault patterns FP_8, FP_9, FP_{10}, FP_{12}, FP_{13} or FP_{14}, G_i will produce a "1" at the output $y1$, thus G_{i+1} will have as inputs $ux1x2 = 101$. Since G_{i+1} is either fault free, or it generates one of the type I fault patterns, then G_{i+1} will produce a "1" (instead of the expected "0") at the output $y2$ (as shown in Table 16), thus, the fault can be detected. Assume that G_i generates FP_1. When the first vector is applied, G_i will have as inputs $ux1x3 = 001$, at the $y2$ output a "1" will be produced

(instead of the expected "0"). So, this fault can also be detected. The only other case is that G_i has as fault pattern FP_{11}, it will be shown next that the second vector can detect it. When the second vector is applied, G_i will have as inputs $ux1x2 = 111$, the expected value of the $y2$ output of all modules is "1". Since G_1 has as fault pattern FP_{11}, then it will a "0" (instead of a "1") at the $y1$ output. So G_{i+1} has as inputs $ux1x2 = 110$. G_{i+1} is either fault free or contains one of the type I fault patterns so G_{i+1} will produce a "0" (instead of the expected "1") at the primary output $y2$ (as shown in Table 16), thus detection is accomplished.

– *QCA1.* With additional controllability and observability, a 1D array of QCA1 gates as shown in Fig. 19a is C-testable under a multiple faulty module assumption. In the array of N modules, additional controllability is provided by setting $x3$ of each module as a primary input; additional observability is obtained by setting $y3$ of each module as a primary output. Let the primary inputs for $x1$ and $x2$ of the first module in the array be denoted as PI_1 and PI_2; the primary outputs for the $y1$ and $y2$ outputs of the last module be denoted as PO_1 and PO_2 and the i_{th} module in the array be denoted by G_i. If $x3$ of a G_i is denoted by U_i, then the fault patterns of the mapping between inputs and outputs for the fault free and faulty modules with FP_i are shown in Table 13.

The test vector set that detect any multiple faulty modules, is given as follows:

1. *First Test Vector*: $PI_1 = 0, PI_2 = 0$, $U_i = 1$ for all i, as shown in Fig. 19a. If the array is fault free, $x1x2x3 = 001 = a_1$ is applied to all modules G_i. The expected output at $y3$ is "1" for every G_i. The expected value at the primary outputs is $PO_1PO_2 = 00$.
2. *Second Test Vector*: $PI_1 = 1, PI_2 = 0$, $U_i = 1$ for all i, as shown in Fig. 19b. If the array is fault free, this applies $x1x2x3 = 101 = a_5$ to all G_i. The expected value at $y3$ is "0" for each G_i. The expected value at the primary outputs is $PO_1PO_2 = 10$.
3. *Third Test Vector*: $PI_1 = 0, PI_2 = 1$, $U_i = 1$ for $i = 1, 3, 5, 7, \ldots$ and $U_i = 0$ for $i = 2, 4, 6, \ldots$, as shown in Fig. 19c. If the array is fault free, this vector applies the input vector $x1x2x3 = 011 = a_3$ to all G_i (for odd i) and $x1x2x3 = 100 = a_4$ to all G_i (for even i). The expected value at $y3$ of G_i is "1" (for odd i) and "0" (for even i). The expected value at the primary outputs is $PO_1PO_2 = 01$ (for even N) and $PO_1PO_2 = 10$ (for odd N).

A test set with 100% coverage for a module made of a QCA1 gate is $x1x2x3 = a_1 = 001$, $a_3 = 011$ (or $a_4 = 100$) and $a_5 = 101$. The first test applies a_1 to all modules, while the second test applies a_5 to all modules. The third test vector applies a_3 or a_4 to all modules. Next it will be shown that these vectors can detect multiple faulty modules (assuming each module can only have one of the faults described in Sect. 6) for arbitrary N.

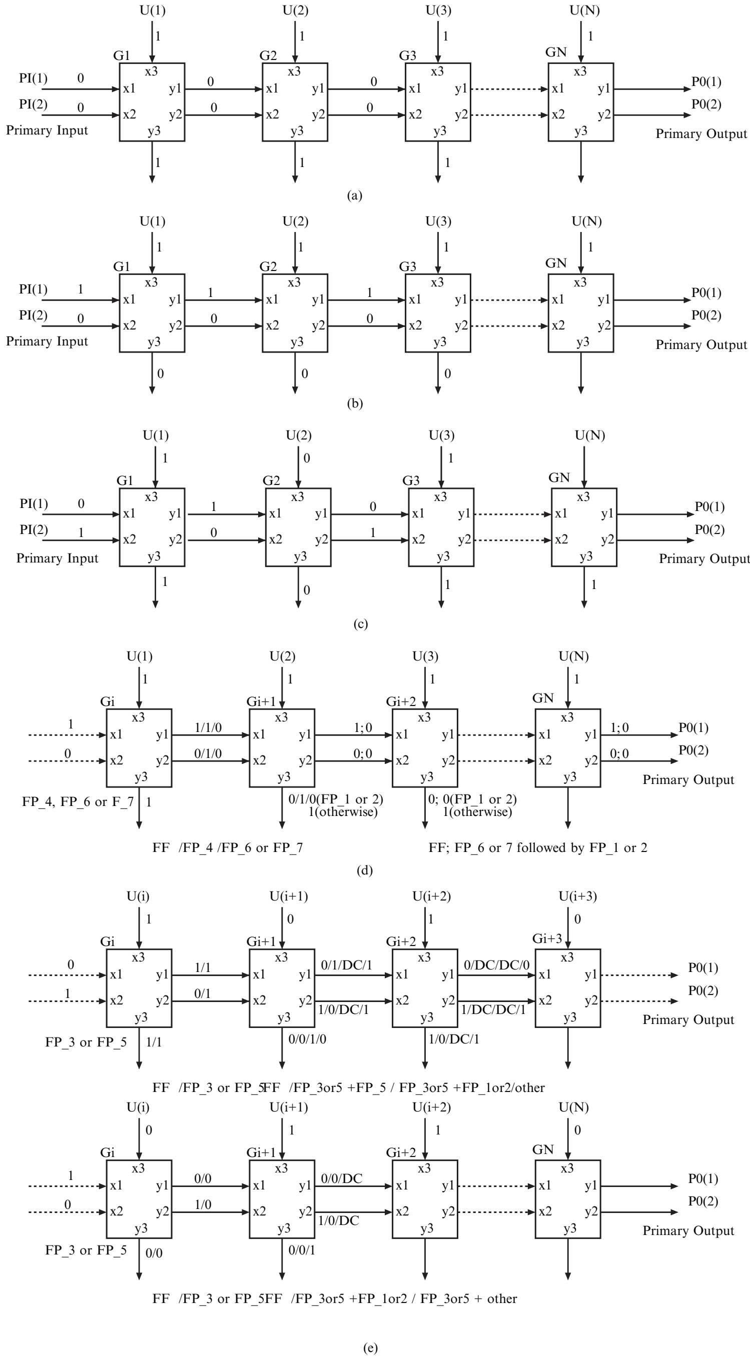


Fig. 19. C-testability of one-dimensional array (made of QCA1 gates) with increased observability and controllability

Assume that the faulty module closest to the primary inputs is given by G_i; then G_i must have one of the fault patterns given in Table 13. Consider initially the case when G_i has FP_1 or FP_2. When applying $x1x2x3 = a_1 =$ 001 to G_i, G_i will produce a "0" at the $y3$ output instead of the expected "1", thus it will be detected. Next consider the case when G_i has FP_4, FP_6 or FP_7. When applying $x1x2x3 = a_5 = 101$ to G_i (as shown in Fig. 19d) and the fault pattern is FP_4, then $x1x2x3 = a_7 = 111$ is applied to G_{i+1}. So, $y3$ of G_{i+1} will be "1" (instead of the expected "0") and detection will occur. If the fault pattern is FP_6 or FP_7, then $x1x2x3 = a_1 = 001$ is applied to G_{i+1}. If G_{i+1} does not have FP_1 or FP_2, then $y3$ of G_{i+1} will become "1" (instead of the expected "0") and detection is accomplished. If there is FP_1 or FP_2 at G_{i+1}, G_{i+2} will get as input $x1x2x3 = a_1 = 001$, and so on, till the last module of the array. So, there will be either an unexpected "1" at $y3$ of some module, or an unexpected "00" at PO_1, PO_2. Detection is always accomplished for a faulty array. Another case is when G_i has FP_3 or FP_5, as shown in Fig. 19e. For the third vector, if i is an odd number, $x1x2x3 = a_3 = 011$ is applied to G_i and G_{i+1} has $x1x2x3 = a_6 = 110$ as input. If G_{i+1} has FP_1 or FP_2, an unexpected "1" will be detected at $y3$ of G_{i+1}. If G_{i+1} has FP_5, then $G_i + 2$ will receive $x1x2x3 = a_4 = 100$ as input and an unexpected "0" is observed at $y3$ of G_{i+2}. Else, $x1x2x3 = a_7 = 111$ will be applied to G_{i+2}. If G_{i+2} has FP_7, it will be detected by the second test vector; if it does not have FP_7, then $x1x2x3 = a_6 = 110$ will be the input for G_{i+3}, and the previously described case will apply as $x1x2x3 = a_6 = 110$ on G_{i+1}. So the fault will be detected either at $y3$ of some module, or at the primary output.
If i is an even number, the fault will result in $x1x2x3 = a_1 = 001$ at G_{i+1}. If G_{i+1} does not have FP_1 or FP_2, this will result in an unexpected "1" at $y3$ of G_{i+1}. If G_{i+1} has FP_1 or FP_2, FP_1 or FP_2 is detected by the first test vector.
In conclusion, the array shown in Fig. 18a is C-testable and four test vectors are required to detect any number of faulty modules, given the fault model in Sect. 6.

– *QCA2.* With additional controllability and observability, a 1D array of QCA2 gates as shown in Fig. 20a is C-testable under the assumption that multiple modules may be faulty (only have faults described in Sect. 6).
In the array of N modules, additional controllability is provided by setting the $x2$ input of each module be a primary input; additional observability is obtained by setting the $y2$ output of each module be a primary output. Let the primary input feeding $x1, x3$ of the first module in the array be PI_1, PI_2; the primacy outputs connecting the $y3, y1$ outputs of the last module be PO_1, PO_2. Let the i_{th} module in the array be G_i, the input feeding $x2$ of the G_i be U_i. The mapping between input and output of the fault free module as well as a faulty module with FP_i is shown in Table 17. The test vector set that can detect any multiple module fault is the following:

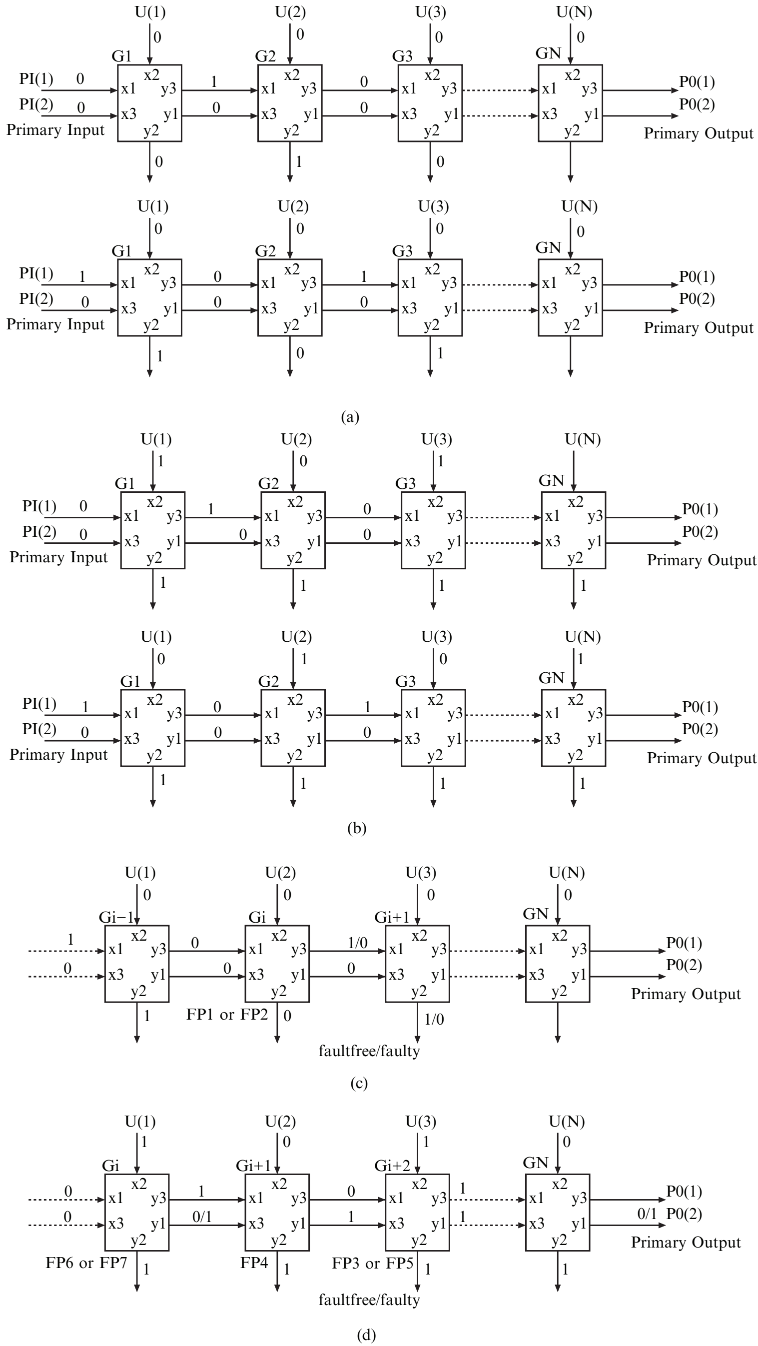


Fig. 20. C-testability of one-dimensional array (made of QCA2 gates) with increased controllability and observability

Table 17. Outputs of fault free and faulty QCA2

Input vector	Fault free output	FP_1	FP_2	FP_3	FP_4	FP_5	FP_6	FP_7
a_0	a_1	a_0	a_0	a_1	a_1	a_1	a_1	a_5
a_1	a_0	a_0	a_0	a_0	a_0	a_2	a_0	a_0
a_2	a_3	a_3	a_2	a_3	a_1	a_3	a_7	a_7
a_3	a_5	a_5	a_5	a_7	a_5	a_7	a_5	a_5
a_4	a_2	a_2	a_2	a_0	a_2	a_0	a_2	a_2
a_5	a_4	a_4	a_5	a_4	a_6	a_4	a_0	a_0
a_6	a_7	a_7	a_7	a_7	a_7	a_5	a_7	a_7
a_7	a_6	a_7	a_7	a_6	a_6	a_6	a_6	a_2

1. *First Test Vector*: $PI_1 = 0, PI_2 = 0$, $U_i = 0$ for all i, as shown in Fig. 20a. If the array is fault free, this vector applies input vector $x1x2x3 = 000 = a_0$ to all G_i such that i is odd and input vector $x1x2x3 = 100 = a_4$ to all G_i such that i is even. The expected output at $y2$ of G_i is "0" for odd i and "1" for even i. The expected output at primary outputs is $PO_1PO_2 = 00$ if N is even and $PO_1PO_2 = 10$ if N is odd.
2. *Second Test Vector*: $PI_1 = 1, PI_2 = 0$, $U_i = 0$ for all i, as shown in Fig. 20a. If the array is fault free, this vector applies input vector $x1x2x3 = 000 = a_0$ to all G_i such that i is even and input vector $x1x2x3 = 100 = a_4$ to all G_i such that i is odd. The expected output at $y2$ of G_i is "0" for even i and "1" for odd i. The expected output at primary outputs is $PO_1PO_2 = 00$ if N is odd and $PO_1PO_2 = 10$ if N is even.
3. *Third Test Vector*: $PI_1 = 0, PI_2 = 0$, $U_i = 1$ for $i = 1, 3, 5, 7, \ldots$ and $U_i = 0$ for $i = 2, 4, 6, \ldots$, as shown in Fig. 20b. If the array is fault free, this vector applies input vector $x1x2x3 = 010 = a_2$ to all G_i such that i is odd and input vector $x1x2x3 = 100 = a_4$ to all G_i such that i is even. The expected output at $y2$ of G_i is "1" for i. The expected output at primary outputs is $PO_1PO_2 = 00$ if N is even and $PO_1PO_2 = 10$ if N is odd.
4. *Fourth Test Vector*: $PI_1 = 0, PI_2 = 0$, $U_i = 0$ for $i = 1, 3, 5, 7, \ldots$ and $U_i = 1$ for $i = 2, 4, 6, \ldots$, as shown in Fig. 20b. If the array is fault free, this vector applies input vector $x1x2x3 = 010 = a_2$ to all G_i such that i is even and input vector $x1x2x3 = 100 = a_4$ to all G_i such that i is odd. The expected output at $y2$ of G_i is "1" for i. The expected output at primary outputs is $PO_1PO_2 = 00$ if N is odd and $PO_1PO_2 = 10$ if N is even.

A 100% coverage test set for a single QCA2 gate is $x1x2x3 = a_0 = 000$, $a_2 = 010$, $a_4 = 100$. The combination of vector 1 and vector 2 applies input vector a_0 and a_4 to all modules, while the combination of vector 3 and vector 4 applies input vector a_2 and a_4 to all modules. Next, we will

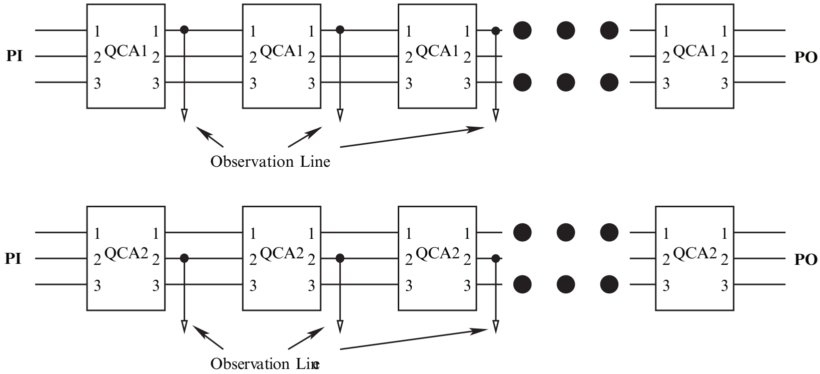


Fig. 21. C-testability of 1D array

show that these vectors can detect multiple module faults (given the fault model in Sect. 6) for arbitrary N.

Assume the faulty module closest to primary input be G_i, the G_i must have one of the fault patterns from Table 17. First consider the case when G_i has FP_3 or FP_5. When applying vector 1 and vector 2, $x1x2x3 = a_4 = 100$ will be applied to G_i once. With this input, G_i will produce an "0" at the $y2$ output instead of the expected "1", thus it will be detected.

Next consider the case when G_i has FP_4. When applying vector 3 and vector 4, $x1x2x3 = a_2 = 010$ will be applied to G_i once. G_i will produce an "0" at the $y2$ output instead of the expected "1", thus it will be detected.

Another possibility is that G_i has FP_1 or FP_2, as shown in Fig. 20c. When applying vector 1 and vector 2, $x1x2x3 = a_0 = 000$ will be applied to G_i once. G_i will produce $y1y2y3 = 000$ at the output, so G_{i+1} will have inputs $x1x2x3 = a_0 = 000$. Since G_{i+1} is either fault free or contains one of the faulty patterns in Table 17, G_{i+1} will produce an "0" at the $y2$ output instead of the expected "1", thus it will be detected.

The only remaining case is when G_i has FP_6 or FP_7. When applying vector 3 and vector 4, $x1x2x3 = a_2 = 010$ will be applied to G_i once. G_i will produce $y1y2y3 = 111$ at the output so G_{i+1} will have inputs $x1x2x3 = a_5 = 101$, as shown in Fig. 20d. If G_{i+1} is fault free or contains any faulty pattern other than FP_4, it will produce an "0" at the $y2$ output instead of the expected "1", thus it will be detected. If G_{i+1} has FP_4, then it will have outputs $y1y2y3 = 110$ and thus apply $x1x2x3 = a_3 = 011$ to G_{i+2}. If G_{i+2} is fault free or contains any faulty pattern other than FP_3 or FP_5, it will produce an "0" at the $y2$ output instead of the expected "1", thus it will be detected. If G_{i+2} has FP_3 or FP_5, it will have outputs $y1y2y3 = 111$ and thus apply $x1x2x3 = a_5 = 101$ to G_{i+3}. At this point it can be seen that unless the faulty patterns of the 1D array is the following: G_i has FP_6 or FP_7, G_{i+j} has FP_4 for $j = 1, 3, 5, 7, \ldots$ and G_{i+j} has FP_3 or FP_5 for $j = 2, 4, 6, 8, \ldots$, the fault can be detected at the $y2$

outputs of some module after G_i. If the faulty array has the above faulty configuration, it will produce a "1" instead of the expected "0" at the primary output PO_2 (the output $y1$ of the last module), thus the fault can also be detected.
In conclusion, the array shown in Fig. 18a is C-testable and four test vectors are required to detect any number of faulty modules (given the fault model in Sect. 6).

10 Conclusion

This chapter has presented a comprehensive analysis of reversible and testable circuits implemented in Quantum-dot Cellular Automata (QCA). Initially, two new reversible gates (denoted as QCA1 and QCA2) have been proposed; these gates take into account the logic primitives of QCA (such as the MV) to retain the one-to-one onto nature of the mapping between inputs and outputs. Albeit the majority function is not reversible, it has been shown in the literature that in QCA different clocking arrangements can be used for reversible computing [32]. In this chapter, testing of one-dimensional arrays under single and multiple faulty modules has been considered; this analysis has encompassed different features as related to the assumed fault model, the cardinality of the test set and controllability/observability in the intermediate modules of the one-dimensional array.

To summarize, Table 18 shows the C-testability of one-dimensional arrays made of Toffoli, Fredkin, QCA1 or QCA2. In all cases, it is assumed the at most one cell fault can occur in each module (as according to the fault model discussed in Sect. 6). The first column of the Table is the type of gate used in the module of the array. The second column shows the fault assumption: "S" stands for single faulty module and "M" stands for multiple faulty modules. The third column shows the cardinality of the test set for detection, i.e. the

Table 18. Benchmark result

Module	Faults S/M	Test set cardinality	Type of array
Toffoli	S	3	1D array
	M	3	1D array
Fredkin	S	3	1D array
	M	4	1D array with 1OB, 1CO
	M	8	1D array with 2OB
QCA1	S	3	1D array
	M	3	1D array with 1OB, 1CO
	M	8	1D array with 1OB
QCA2	S	4	1D array
	M	4	1D array with 1OB, 1CO
	M	8	1D array with 1OB

number of vectors in the test set. The last column is the type of array. The 1D array is shown in Fig. 15, where only the inputs of the first module can be controlled and only the outputs of the last module can be observed. The 1D array with 1OB, 1CO is shown in Fig. 18, where each module has one controllable input and one observable output. the 1D array with 1OB is shown in Fig. 21, where each module has one observable output. The 1D array with 2OB is the array in which each module has two observable outputs.

Appendix A: Benchmark Specifications[2]

MOD5: Four inputs (A, B, C, D, from MSB to LSB) one output.

Output = (A XNOR C) AND (B XNOR D). XNOR implemented using the CNOT gate, AND implemented by QCA1/QCA2 gate.

rd32: Three inputs (A, B, Cin) and two outputs (Sum, $Cout$).

$Sum = A$ XOR B XOR Cin; $Cout = \mathrm{MV}(A, B, Cin)$. The 3-input XOR implemented with two QCA1/QCA2 gates, and $Cout$ is available at the output of one of these QCA1/QCA2 gates, too.

3_17: Three inputs (A, B, C, from MSB to LSB), three outputs (3,2,1, from MSB to LSB).

2 = A XOR B XOR C'

1 = (A' AND C') OR (A AND B) = B CNOT (A' AND 2)

3 = (A' AND B') OR (B AND C) = C CNOT (B' AND 2)

The XOR gate implemented with the CNOT gate, the AND gate is implemented with the QCA1/QCA2 gate.

References

1. S. Muroga, "Threshold Logic and its Applications", *Wiley Interscience*, New York, 1971.
2. T. Toffoli, "Reversible Computing", *Technical Report MITLCSTM151, MIT Laboratory for Computer Science*, 1980.
3. E. Fredkin and T. Toffoli, "Conservative Logic", *International Journal of Theoretical Physics*, vol. 21, pp. 219–253, 1982.
4. C.H. Bennett, "Logic Reversibilty of Computation", *IBM Journal of Research and Development*, vol. 17, pp. 525–532, 1973.
5. D. Maslov, G.W. Dueck and D.M. Miller, "Synthesis of Fredkin-Toffoli Reversible Networks", *IEEE Transaction on VLSI*, 2004.
6. R. Landauer, "Irreversibility and Heat Generation in the Computing Process", *IBM Journal of Research and Development*, vol. 5, pp. 183–191, 1961.
7. M. Nielsen and I. Chuang, "Quantum Computation and Quantum Information", *Cambridge University Press*, Cambridge, 2000.

[2] For QCA, inversion is considered as part of the interconnect, so inverters are not counted in the number of gates.

8. W. Wang, R. Zhang, K. Walus and G.A. Jullien, "A Method of Majority Logic Reduction for Quantum Cellular Automata", *IEEE Transaction on Nanotechnology*, vol. 3(4), pp. 443–450, 2004.
9. C.S. Lent, P.D. Tougaw and W. Porod, "Quantum Cellular Automata: The Physics of Computing with Arrays of Quantum Dot Molecules", *Proceedings of the Workshop on Physics and Computing*, pp. 5–13, 1994.
10. M.T. Niemier and P.M. Kogge, "Problems in designing with QCAs: layout=timing", *International Journal of Circuit Theory and Applications*, vol. 29(1), pp. 49–62, 2001.
11. M.T. Niemier and P.M. Kogge, "Logic-in-Wire: Using Quantum Dots to Implement a Microprocessor", *International Conference on Electronics, Circuits, and Systems (ICECS '99)*, vol. 3, pp. 1211–1215, 1999.
12. K. Hennessy and C.S. Lent, "Clocking of Molecular Quantum-Dot Cellular Automata", *Journal of Vaccum Science and Technology*, vol. 19(5), pp. 1752–1755, 2001.
13. I. Amlani, A.O. Orlov, G. Toth, C.S. Lent, G.H. Bernstein and G.L. Snider, "Digital Logic Gate Using Quantum-Dot Cellular Automat", *Science*, vol. 284(5412), pp. 289–291, 1999.
14. S.E. Frost, A.F. Rodrigues, A.W. Janiszewski, R.T. Rausch and P.M. Kogge, "Memory in Motion: A Study of Storage Structures in QCA", *1st Workshop on Non-Silicon Computation*, 2002.
15. M.T. Niemier, A.F. Rodrigues and P.M. Kogge, "A Potentially Implementable FPGA for Quantum Dot Cellular Automata", *1st Workshop on Non-Silicon Computation (NSC-1), held in conjunction with 8th Int. Symp. on High Performance Computer Architecture (HPCA-8)*, 2002.
16. P.D. Tougaw and C.S. Lent, "Logical Devices Implemented Using Quantum Cellular Automata", *Journal of Applied Physics*, vol. 75(3), pp. 1818–1825, 1994.
17. V.S. Dimitrov, G.A. Jullien and K. Walus, "Quantum-Dot Cellular Automata Carry-Look-Ahead Adder and Barrel Shifter", *IEEE Emerging Telecommunications Technologies Conference*, pp. 2/1–2/4, 2002.
18. C.G. Smith, "Computation Without Current", *Science*, vol. 284(2), p. 274, 1999.
19. K. Walus, R.A. Budiman and G.A. Jullien, "Effects of morphological variations of self-assembled nanostructures on quantum-dot cellular automata (QCA) circuits", *Frontiers of Integration, An International Workshop on Integrating Nanotechnologies*, 2002.
20. K. Walus, A. Vetteth, G.A. Jullien and V.S. Dimitrov, "RAM Design Using Quantum-Dot Cellular Automata", *NanoTechnology Conference*, vol. 2, pp. 160–163, 2003.
21. M.B. Tahoori, M. Momenzadeh, J. Huang and F. Lombardi, "Testing of Qunatum Cellular Automata", *IEEE Transaction on Nanotechnology*, vol. 3(4), pp. 432–442, 2004.
22. J. Huang, M. Momenzadeh, M.B. Tahoori and F. Lombardi, "Defect Characterization for Scaling of QCA Devices", *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 30–38, 2004.
23. D.A. Antonelli, D.Z. Chen, T.J. Dysart, X.S. Hu, A.B. Kahng, P.M. Kogge, R.C. Murphy and M.T. Niemier, "Quantum-Dot Cellular Automata (QCA) Circuit Partitioning: Problem Modeling and Solutions", *Design Automation Conference (DAC)*, pp. 363–368, 2004.
24. K. Walus, G.A. Jullien and V.S. Dimitrov, "Computer arithmetic Structures for Quantum Cellular Automata", *Proceedings of Asimolar Conference*, 2003.

25. J. Huang, M. Momenzadeh, M. Ottavi, L. Schiano and F. Lombardi, "A Pre-deposition Methodology for Tile-Based Design of QCA Combinational Circuits", Internal report, 2004.
26. M. Momenzadeh, J. Huang, M. Ottavi, N. Park and F. Lombardi, "Computing with Grids of QCA Cells", Internal report, 2004.
27. R. Compano, L. Molenkamp and D.J. Paul, "Technology Roadmap for Nano-electroincs", *European Commission IST programme, Future and Emerging Technologies*, 2000.
28. Reversible Logic Synthesis Benchmarks Page, available online: http://www.cs.uvic.ca/damslov
29. Personal communication with Professor Marya Lieberman, Department of Chemistry and Biochemistry, University of Notre Dame, IN, USA.
30. V.D. Agrawal, "An Information Theoretic Approach to Digital Fault Testing", *IEEE Transaction on Computers*, vol. 30, pp. 582–587, 1981.
31. K.N. Patel, J.P. Hayes and I.L. Markov, "Fault Testing for Reversible Circuits", *IEEE Transaction on CAD*, vol. 23(8), pp. 1220–1230, 2004.
32. J. Timer and C.S. Lent, "Maxwell's Demon and Quantum-dot Cellular Automata", *Journal of Applied Physics*, vol. 94(2), pp. 1050–1060, 2003.
33. M. Liu, C.S. Lent, "Bennett and Landauer Clocking in Quantum-dot Cellular Automata", *International Workshop on Computational Electronics*, Abstracts pp. 120–121, 2004.
34. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, "Introduction to Algorithms, 2nd ed.", *McGraw-Hill*, New York, 2001.
35. A. Chakraborty, "Synthesis of Reversible Circuits for Testing with Universal Test Set and C-Testability of Reversible Iterative Logic Arrays", *Proceedings of the 18th International Conference on VLSI Design*, 2005.
36. QCADesigner Homepage, available online: www.qcadesigner.ca

Chapter 7: Cellular Array-Based Delay-Insensitive Asynchronous Circuits Design and Test for Nanocomputing Systems

J. Di and P.K. Lala

1 Introduction

Complementary metal-oxide semiconductor (CMOS) has been the dominant technology for implementing VLSI systems because it provides a good trade-off of high speed and small area. The continuous decrease in transistor feature size has been pushing the CMOS process to its physical limits caused by ultra-thin gate oxides, short channel effects, doping fluctuations, and the unavailability of lithography in nanoscale range. To continue the size/speed improvement trends according to Moore's Law, nanoelectronic and molecular electronic devices are needed. A significant amount of research has been done in nanoscale computing system design [1–5]. Although recent research have resulted in the development of basic logic elements and simple circuits in nanoscale, there are still debates on what logic style and architecture will be the best for nanocomputers. A family of asynchronous logic called delay-insensitive circuits has drawn attention in recent years. The advantages of delay-insensitive circuits include flexible timing requirement, low power, high modularity, etc. These characteristics fit the needs of nanoscale computing. Cellular arrays have an ideal architecture for implementing delay-insensitive circuits in nanoscale; they have highly regular structures, simple cell behavior, and flexible scalability [5, 6]. The regular structure together with delay-insensitive circuit style makes cellular arrays a viable option for implementing nanocomputing systems.

In this chapter the design and layout of delay-insensitive circuits on cellular arrays are presented. One appealing approach to design nanoscale circuits is to express the logic functions in the Reed–Muller form using AND and XOR gates only [7]; the Reed–Muller representation of Boolean functions have been discussed in detail in Sect. 5. The main motivation for using the Reed–Muller form is that the testability of circuits implemented from Reed–Muller expansions is considerably improved compared to that of their original forms. This chapter presents the design and layout of a delay-insensitive Reed–Muller cell which is composed of the three primitives proposed in [6]. In addition,

a potential physical implementation for asynchronous circuits on cellular arrays introduced by IBM researchers [8], and the effects of possible stuck-at faults on this molecular implementation have been analyzed. Certain faults in these circuits can be detected during normal operation without applying any external test patterns, i.e. the faults can be detected on-line [7]. However, faults that escape on-line detection can be easily detected off-line by applying a few external test patterns; this is possible because of the Reed–Muller expression-based structure of the delay-insensitive circuits.

2 Delay-Insensitive Circuits

Currently synchronous logic is predominantly used in commercial integrated circuit chip design. Synchronous logic uses a global clock to control logic behavior. On the other hand asynchronous logic delay-insensitive circuits do not require a clock. Delay-insensitivity is achieved by checking the completion of an operation in a subblock and sending a signal to previous subblock stages to acknowledge the completion. The clock-less operation of such circuits lead to the following benefits:

- *No clock skew.* Since delay-insensitive asynchronous circuits have no globally distributed clock, clock skew need not be considered.
- *High energy efficiency.* Delay-insensitive circuits generally have transitions only where and when involved in the current computation. Some implementations inherently eliminate glitches, therefore decreasing energy consumption.
- *Robust external input handling.* Since signals do not need to be synchronized with a clock, delay-insensitive circuits accommodate external inputs thus avoiding the metastability problem in synchronous circuits [9].
- *Low noise and low emission.* In mixed-signal circuits, a digital subcircuit usually generates noise and/or emits electromagnetic (EM) radiation which affects the whole circuit performance. Due to the absence of a complex clocking network, delay-insensitive circuits may have better noise and EM properties [10].

Asynchronous circuits are very different from Boolean logic based synchronous circuits and are usually more difficult to design without appropriate CAD tools. This remains a major obstacle to the use of asynchronous logic systems. However, in recent years the clock skew problem in synchronous circuits is becoming increasingly critical. Because of above listed advantages delay-insensitive circuits utilizing dual-rail encoding are becoming popular. Dual-rail encoding uses a two-rail code to represent a data bit as shown in Table 1 [11].

As can be seen in Table 1, besides Data 0 and Data 1 there is a spacer state denoted by both rails being logic low. For each circuit element, after a data bit has been processed, it goes to a spacer state before it is allowed to process

Table 1. Dual-rail encoding truth table

State	Rail 1	Rail 0
Spacer	0	0
Data 0	0	1
Data 1	1	0
Invalid	1	1

the next data. This is known as the *return-to-spacer* protocol. This return-to-spacer procedure is actually a self-resetting step of the circuit operation. Delay-insensitivity is achieved by checking the completion of an operation in a subblock and sending a signal to previous subblock stages to acknowledge the completion. Dual-rail encoding is used to design delay-insensitive circuits in this chapter.

3 Asynchronous Cellular Arrays

Cellular arrays have been widely studied as a computational model. A cellular array is a d-dimensional array of identical cells, which are placed next to each other. Each cell has a finite number of states. Based on its current state and the states of its neighbor cells, the cell can change its state by an operation known as a *transition* [6]. Following certain transition rules, the cellular arrays are able to perform logic computation as well as data storage.

In this chapter, two-dimensional cellular arrays are used. Each cell has four bits, represented by rectangles located at north, south, east, and west, respectively, as shown in Fig. 1. If one or more rectangles in a cell are filled a *signal* is said to have arrived at this cell. An empty rectangle represents a logic “0” and a filled rectangle (also known as a *block*) represents a logic “1,” as shown in Fig. 2. Depending on the physical implementation, signals can be generated by different phenomena. In the molecular cascade structure which will be explained in the next section, signals are generated by CO molecular hopping.

Based on the *von Neumann* neighborhood definition, the cell’s next state will be determined by its current state and states of the four nearest bits of its nearest orthogonal cells as shown in Fig. 3. The letters in the rectangles of each cell represent the bit-states (0 or 1) of the cell.

A cell changes its state based on previously mentioned transition rules as shown in Table 2 below [6]. There is no timing restraint on such transitions, except that two neighboring cells may not be updated simultaneously. Instead, the cells are updated in order following the signal transmission direction, and the primitives should be placed in such a way that they will not share cells in each other. Thus cellular arrays are ideal for implementing delay-insensitive circuits.

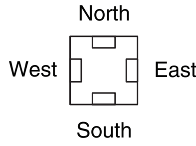


Fig. 1. Cell structure

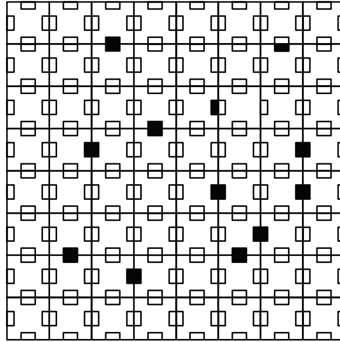


Fig. 2. A 2D cellular array

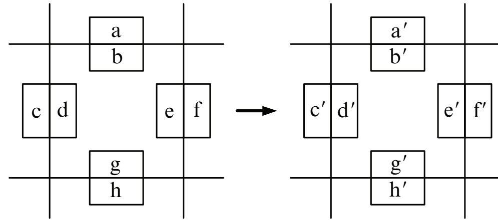
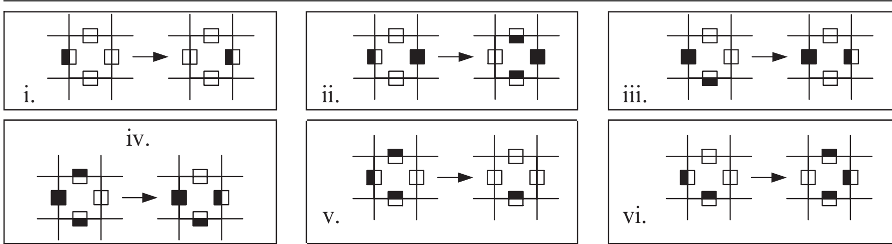


Fig. 3. Transition of cell

Table 2. Transition rules. (i) Signal propagation; (ii) Rule for Fork; (iii) and (iv) Rules for Merge; (v) and (vi) Rules for R-Counter



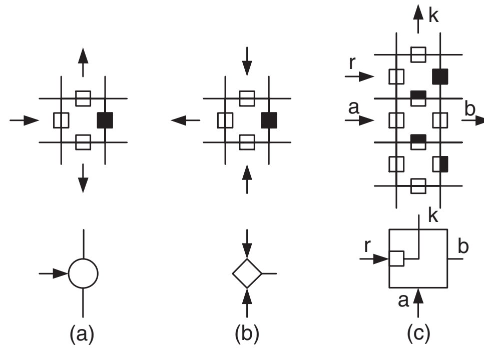


Fig. 4. Primitives (**a**) Fork (**b**) Merge (**c**) R-Counter

Three primitives – *Fork*, *Merge*, and *Resettable Modulo 2 Counter (R-Counter)* – in conjunction with the transition rules shown in Table 2 form a complete set for designing delay-insensitive circuits [6]. They are shown in Fig. 4 with their symbols:

- *Fork*. It produces one signal on each of its two output paths upon assimilating a signal from its input path.
- *Merge*. Signals arriving from the input paths are redirected to the output path. Simultaneous input signals are allowed, giving rise to two consecutive output signals.
- *Resettable Modulo 2 Counter (R-Counter)*. Two successive input signals on a give rise to one output signal on b. One input signal on a and one on r give rise to one output signal on k.

Note that in a cellular array after signals along a path have been processed, the relevant cells along the path including both primitives and signal paths go back to their initial states, which are logic 0's. Thus the requirement of "return-to-spacer" protocol is automatically satisfied, thereby providing several advantages, e.g. less area and power, higher reliability, lower circuit complexity, and less design effort.

A delay-insensitive module named TRIA (see Fig. 5) was introduced in [12]. It has three inputs and three outputs, each output corresponding to a unique pair of inputs. If a TRIA receives one input signal on one input line, it stays pending until a second input signal arrives on another input line. It then responds by outputting a signal to the corresponding output line. For example, if the two inputs are on input line I_i and I_j, where i, $j \in [1,3]$, the output will be on output line O_{6-i-j} [6]. For example if the inputs are on I_1 and I_3 the output will be on O_2 as $6 - 1 - 3 = 2$. A TRIA, which can be expressed in terms of the previously mentioned primitives, and its symbol are shown in Fig. 5. TRIAs are convenient building blocks for constructing delay-insensitive circuits.

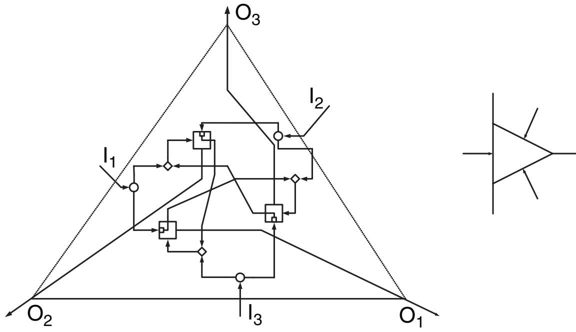


Fig. 5. Design of TRIA constructed from basic modules and its symbol

4 Potential Physical Implementation and Area Discussions of Cellular Arrays

4.1 Molecule Cascades

Recently researchers at IBM have demonstrated the layout of cascades of hopping CO molecules on a Cu(111) surface at cryogenic temperatures [8]. If CO molecules are arranged in configurations such that the motion of one molecule causes the subsequent motion of another, the hops of molecules behave like a row of toppling dominoes [8]. The cascade of CO molecules in staggered chains of "dimers" can be initiated by moving a "trigger" CO molecule to form an initial "chevron." This newly formed chevron then spontaneously decays and forms yet another chevron, and so on for a cascade of any length [8]. Note that although the time for each chevron to decay differs, all chevrons will decay in order to reach the final configuration. This is a typical delay-insensitive mode of operation.

The interactions of the CO molecules can be utilized to realize the transmission along a path, as well as the operation of delay-insensitive primitives and logic gates. A logic AND gate, a two-input sorter, and a three-input sorter were introduced in [8].

4.2 Possible Faults in Molecule Cascades

During the fabrication process of cascades of CO molecules certain faults may occur. The possible faults include missing CO molecule, misplaced CO molecule, wrong configuration, adding "trigger" molecule by mistake, etc. If such a fault occurs inside a logic gate formed by molecule cascades, its effect maybe the same as the stuck-at faults in CMOS circuits, e.g. the output of a logic gate is held at a certain logic value regardless of the input pattern. Thus such faults need to be analyzed and modeled for logic circuits implemented on molecule cascades based cellular arrays.

4.3 Cost

Delay-insensitive circuits usually require more area than their synchronous counterparts due to the multi-rail encoding scheme. For dual-rail encoding delay-insensitive circuits, the total area of logic gates and interconnections is about four times larger. This fact has been restricting the commercialization of delay-insensitive asynchronous circuits because the extra die area is very expensive in modern digital integrated circuits. However, this may not be a problem if implemented using molecule cascades. The most salient feature of molecule cascades is their size: A three-input sorter implemented in CMOS9s technology requires an area of $53\,\mu\text{m}^2$, whereas the cascade implementation uses only $200\,\text{nm}^2$, an improvement factor of 260,000 [8]. Therefore even though delay-insensitive asynchronous circuits have more wires and more complex circuit structure, their implementation on molecule cascades based cellular arrays will still be requiring much less area than their synchronous counterparts implemented using CMOS technology.

5 Testable Circuit Design Based on Reed–Muller Expansion

As indicated earlier a Boolean function can be expressed in the Reed–Muller form, i.e. as a linear sum of product terms with no complementation. The typical Reed–Muller canonical form is shown in (1), where z is the output variable, x_i, $i \in [1,\ n]$, are the input variables, $a_j \in \{0,\ 1\}, j \in [0,\ 2^n - 1]$, are the coefficients.

$$z = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_1x_2 \oplus \cdots \oplus a_{2^n-1}x_1x_2 \ldots x_n \tag{1}$$

An n-variable Boolean function designed in Reed–Muller form has certain properties that make the resulting circuit easily testable. These properties are [13]:

1. If the primary input leads are fault free, then at most $n+4$ tests are required to detect all single stuck-at faults in the circuit.
2. If there are faults on the primary input leads as well, then the number of tests required is $(n + 4) + 2n_e$, where n_e is the number of input variables that appear an even number of times in the product terms of the Reed–Muller expansion. However, by adding an extra AND gate with its output being made observable, the additional $2n_e$ tests can be removed. The input to the AND gate are those inputs appearing an even number of times in the Reed–Muller product terms.

6 Implementation of Delay-Insensitive Circuits Using Reed–Muller Cell

As mentioned earlier, the main motivation for using the Reed–Muller form is to improve the testability of circuits implemented from Reed–Muller expansions. There are only two components in a Reed–Muller circuit, AND gate and XOR gate. A dual-rail delay-insensitive NAND gate is given in [6]. Since a dual-rail signal can be inverted simply by crossing the two rails, the AND gate has the same circuit structure as NAND gate, as shown in Fig. 6. Its layout is shown in Fig. 7.

A delay-insensitive XOR gate, which is more complicated than an AND gate, has been designed by the authors as shown in Fig. 8 and its layout of

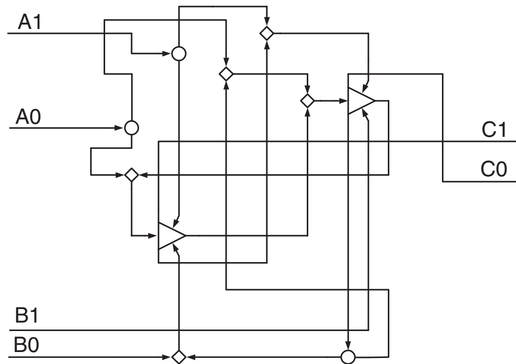


Fig. 6. Delay-insensitive AND gate

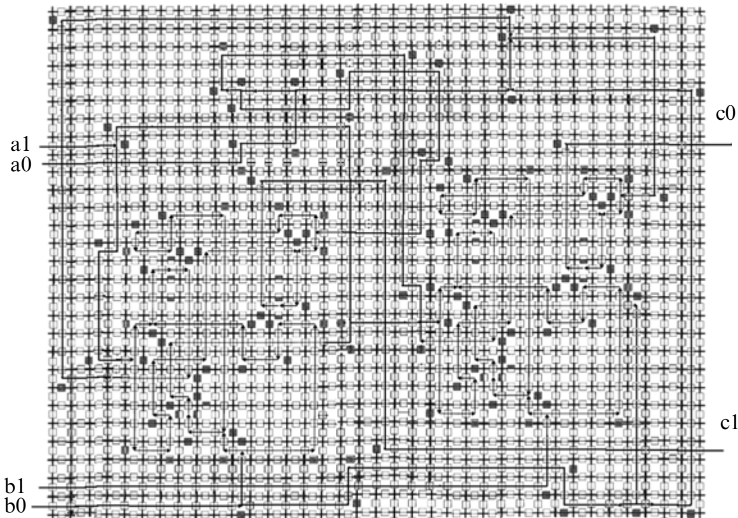


Fig. 7. Layout of AND gate on cellular arrays

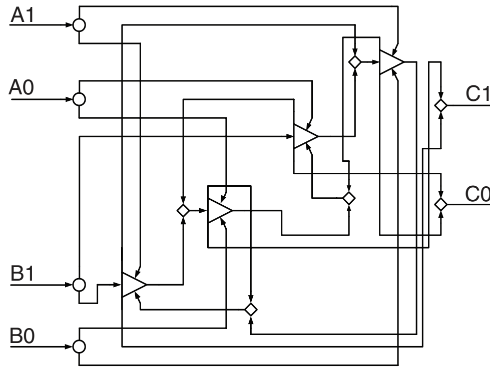


Fig. 8. Delay-insensitive XOR gate

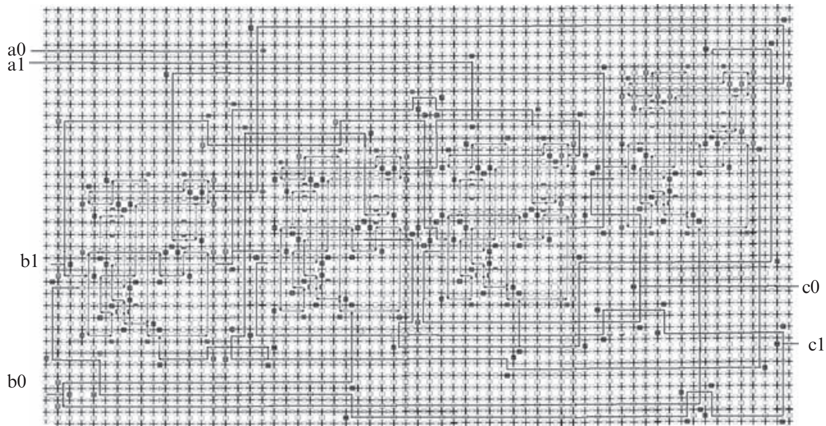


Fig. 9. Layout of XOR gate on cellular arrays

shown in Fig. 9. Note that for each of these two delay-insensitive gates the output is not produced unless both dual-rail inputs arrive. Thus if only one input arrives, it will wait for the other one, and no output will be generated in the meantime.

For implementing sequential logic, a dual-rail delay-insensitive 1-bit register has been designed as shown in Fig. 10. The layout is shown in Fig. 11. Based on the state of REQ signal from the next stage, the register is able to either pass or hold the current input. Note if the input arrives before the REQ signal, the register will hold this input and wait for the REQ signal. Only after the arrival of REQ signal will the input be passed to the output. Once the input has been passed to the next stage, an ACK signal is generated and transferred to the previous stage.

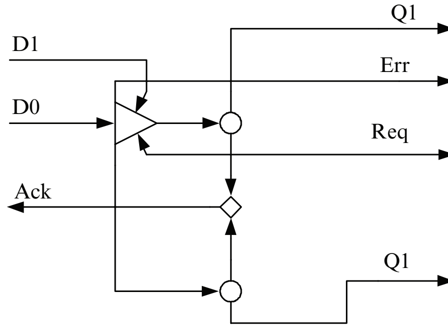


Fig. 10. Delay-insensitive register

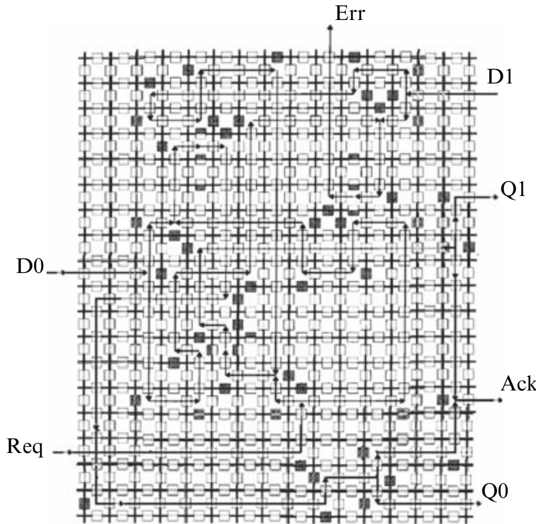


Fig. 11. Layout of delay-insensitive register on cellular arrays

In many cases a Reed–Muller expression has AND terms with more than two variables. Thus multiple-input AND gates are needed to implement certain Reed–Muller expressions. A dual-rail 2-to-1 multiplexer is employed to construct multiple-input AND gates; this will be illustrated later in this section. Figure 12 shows the design of such a multiplexer.

We propose a cell identified as a Reed–Muller cell shown in Fig. 13 to implement Reed–Muller expressions. It contains an AND gate, an XOR gate, a 2-to-1 multiplexer, and a 1-bit register. Every input and output is encoded in dual-rail except the REQ and ACK signals. The select signals, Sel_0 and Sel_1, are logic 0 after reset. Based on the Reed–Muller expression, the synthesis tool connects the REQ signal from the next stage to either Sel_0 or Sel_1 signal of the multiplexer. If the REQ signal is connected to Sel_1, the cell will realize the function shown in (2); if the REQ signal is connected to Sel_0, the cell

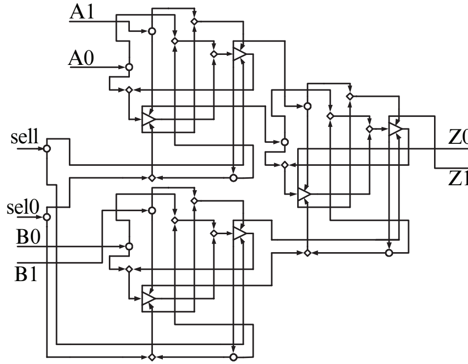


Fig. 12. Dual-rail 2-to-1 multiplexer

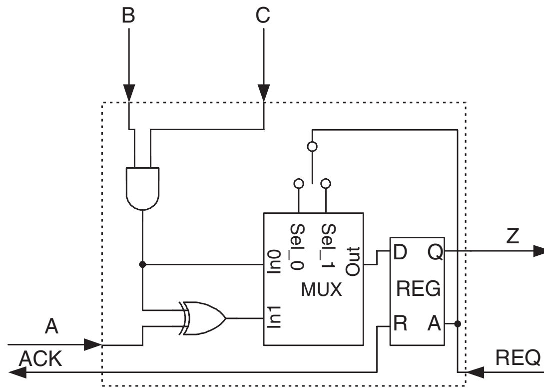


Fig. 13. Basic circuit block for Reed–Muller circuits

will realize the function shown in (3). As stated before, this is to handle expressions in which some AND terms contain more than two variables. The register ensures the delay-insensitivity by getting REQ signal from the next stage and driving ACK signal to the previous stage.

$$Z = A \oplus BC \tag{2}$$

$$Z = BC \tag{3}$$

Figures 14a,b illustrate the use of the Reed–Muller cell in implementing the Boolean expressions of (4) and (5), respectively. Note that (5) requires a three-input AND gate.

$$Z = A \oplus BC \oplus DE \tag{4}$$

$$Z = A \oplus BCD \tag{5}$$

A Boolean expression needs to be converted into Reed–Muller form before it can be implemented using Reed–Muller cells. To illustrate let us realize the

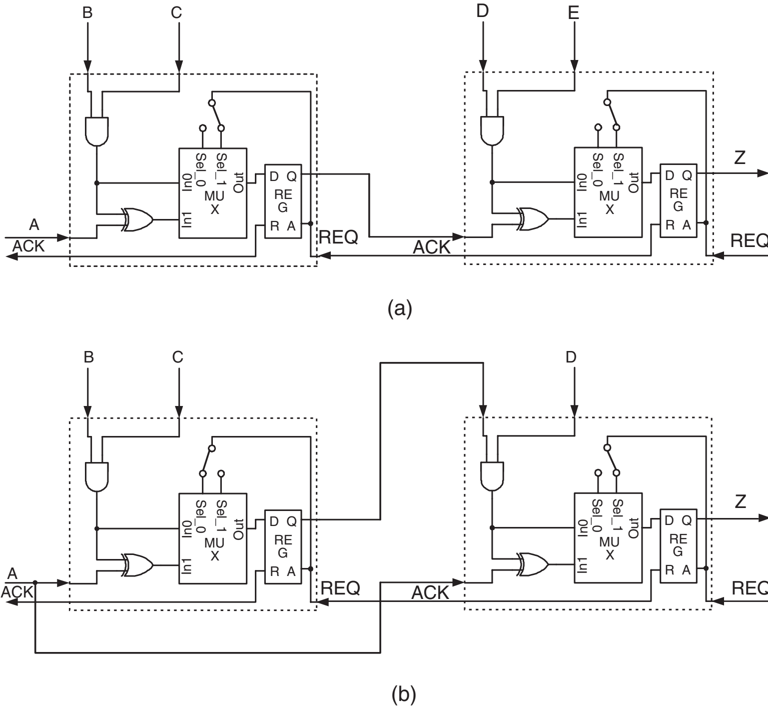


Fig. 14. Implementation of multiple-input AND gates

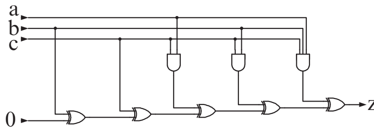


Fig. 15. Normal circuit implementation for (7)

logic function shown in (6). A corresponding complement-free Reed–Muller expression is shown in (7) and its regular implementation is shown in Fig. 15. The implementation of (7) using the Reed–Muller cells is shown in Fig. 16.

$$Z = a \cdot b + \overline{a} \cdot c + b \cdot \overline{c} \tag{6}$$

$$Z = b \oplus c \oplus a \cdot c \oplus b \cdot c \oplus a \cdot b \cdot c \tag{7}$$

Equation (6) can be written in an alternate form as shown in (8). The implementation of the equation is shown in Fig. 17. Note this implementation does not require any inverters; the inversion of an input is achieved by swapping the two rails of each input. Compared to the circuit in Fig. 16, the implementation of Reed–Muller form with complementary inputs uses less

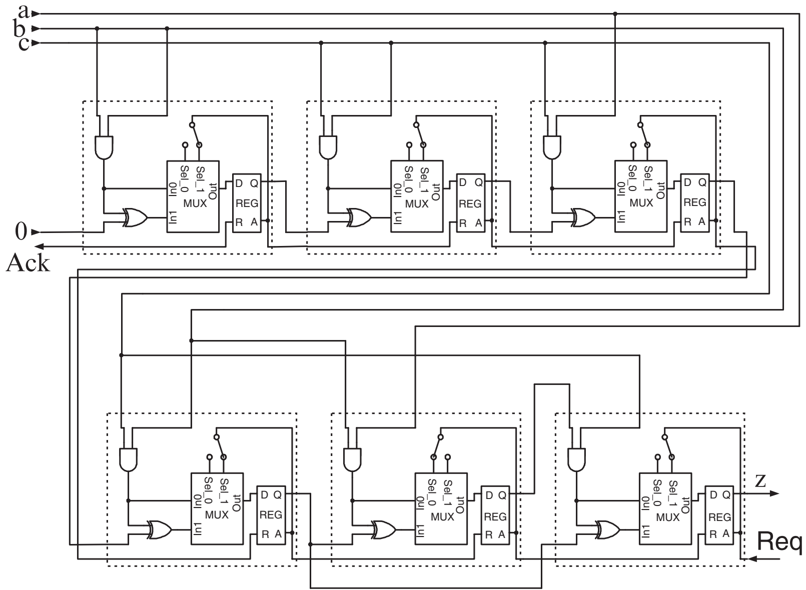


Fig. 16. Circuit realizing equation (7) using basic circuit blocks

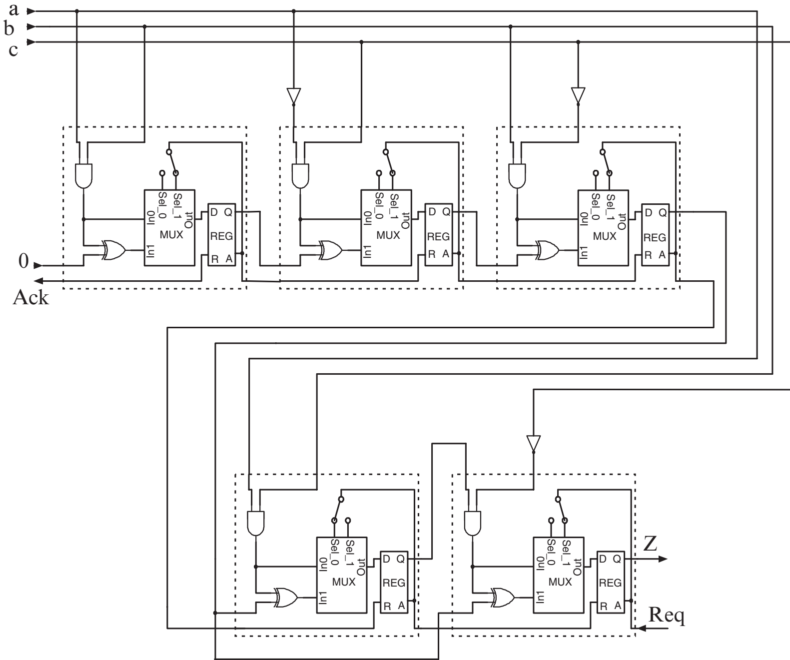


Fig. 17. Circuit realizing equation (8) using basic circuit blocks

area and reduces the number of transitions, thereby reducing power consumption.

$$Z = a \cdot b \oplus \overline{a} \cdot c \oplus b \cdot \overline{c} \oplus a \cdot b \cdot \overline{c} \tag{8}$$

It should be clear from the above example that the delay-insensitive Reed–Muller cell of Fig. 13 can be used to implement arbitrary logic functions. The high modularity feature of the proposed approach should ease the automatic synthesis of nanocomputing circuits/systems.

7 Fault Analysis of Logic Functions Implemented by Reed–Muller Cells

This section provides a comprehensive analysis of the effect of stuck-at faults in the proposed Reed–Muller cell in both primitive- and gate-level.

7.1 Primitive-Level Fault Analysis

During the analysis, two assumptions have been made:

Assumption 1. *The changes of configuration due to a fault happen before any input arrives.*

This assumption indicates that after power on, all configuration changes caused by stuck-at faults happen first. Thus no input signal arrives during the configuration change of a primitive.

Assumption 2. *Stuck-at faults occur one at a time.*

This assumption simplifies the analysis by ignoring the possibility of many faults occurring simultaneously.

The results of all stuck-at faults in primitives can be classified into four categories:

1. *Faulty signal.* One or more faulty signals are transmitted.
2. *Configuration transformation.* The primitive cannot perform its intended function when the correct inputs arrive. Instead, these inputs are processed by a faulty configuration formed due to the transformation.
3. *Null configuration.* The primitive loses its original function but no new configuration is formed to process the input data.
4. *Benign fault.* The fault does not affect the primitive's function.

For each stuck-at fault, the result could be one or a combination of these four categories. Stuck-at fault analysis for all three primitives has been performed. But for the sake of brevity only those for Fork are discussed in detail here. Merge and R-Counter can be analyzed in the same manner. The detailed analysis can be found in [14].

To identify the location of a fault, the rectangles in Fork are numbered as shown in Fig. 18.

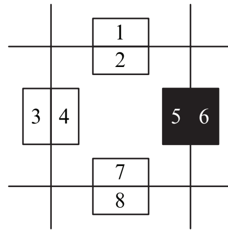
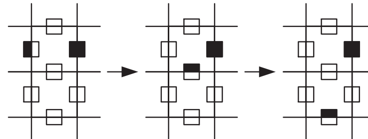


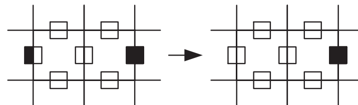
Fig. 18. Rectangle numbers in Fork

The effects of a stuck-at-0 fault in each rectangle/block of Fork are discussed as follows:

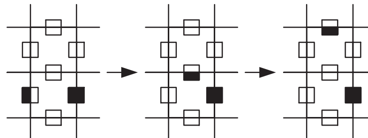
No. 2 – signals to the top are blocked. If the rectangle No. 2 is stuck at 0, it cannot be filled by the incoming signal. Therefore the upper branch of the Fork is blocked.



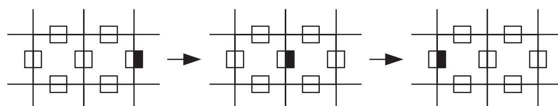
No. 3 – null configuration; circuit fails to function. If the rectangle No. 3 is stuck at 0, it cannot be filled by the incoming signal. Therefore no signal can reach the Fork.



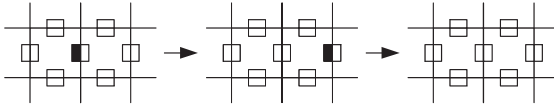
No. 5 – signals to the bottom are blocked. If the rectangle No. 5 is stuck at 0, it cannot be filled by the incoming signal. Therefore the bottom branch of the Fork is blocked.



No. 7 – a faulty signal is transmitted to the left. If the rectangle No. 7 is stuck at 0, it cannot be filled by the incoming signal. The Fork will be transformed to a signal path from right to left.

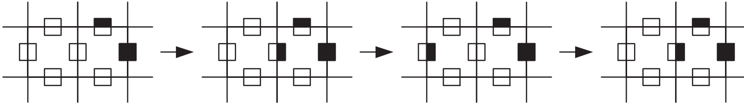


No. 8 – a faulty signal is transmitted to the right. If the rectangle No. 8 is stuck at 0, it cannot be filled by the incoming signal. The Fork will be transformed to a signal path from left to right.

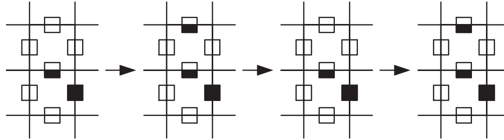


The stuck-at-0 faults in slots 1, 4 and 6 have no effect on the normal operation of a Fork, i.e. these faults are benign.

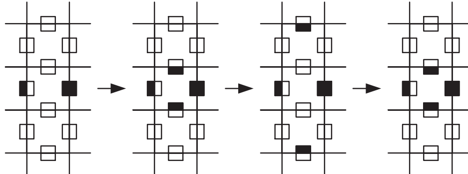
The effects of a stuck-at-1 fault in each slot of Fork are discussed as follows:
No. 1 – faulty signals are transmitted to the left; it works as a Merge



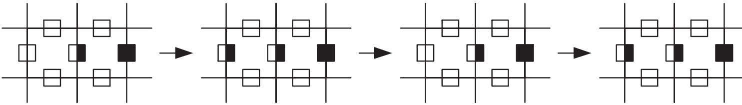
No. 2 – faulty signals are transmitted to the top



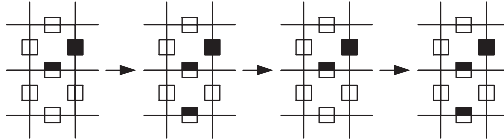
No. 3 – two sets of faulty signals are transmitted to the top and bottom



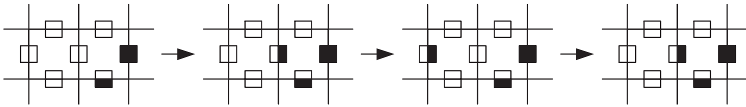
No. 4 – faulty signals are transmitted to the left



No. 5 – faulty signals are transmitted to the bottom



No. 6 – faulty signal are transmitted to the left; it works as a Merge



The stuck-at-1 faults in slots 7 and 8 have no effect on the normal operation of a Fork.

7.2 Gate-Level Fault Analysis

Stuck-at-0 Faults

A stuck-at-0 fault on any input rail will cause a false spacer state if this rail is supposed to be logic 1 during the current data state. For example, if a dual-rail signal A is supposed to be Data 1, i.e. A^1 is 1 and A^0 is 0, and A^1 happens to be stuck-at-0, then A^1A^0 will be 00 and signal A will be in a false spacer state.

Stuck-at-0 faults can be detected by monitoring the output. Due to the delay-insensitive nature of logic gates in the Reed–Muller cells, the output of each gate will remain in a spacer state until all inputs arrive with data. So no data will be detected at the output of the circuit.

Stuck-at-1 Faults

To analyze the effect of stuck-at-1 faults on the Reed–Muller cell, it is necessary to understand how stuck-at-1 faults affect the behavior of TRIA, which is the essential element of all logic gates in the cell.

Single Stuck-at-1 Fault in a TRIA

A single stuck-at-1 fault means there is only one input of TRIA that is stuck-at-1. Figure 19 shows the effect of this type of fault. Note that in Figs. 19–25, a solid dot (•) besides a rail indicates this rail is always at logic 1 because of the stuck-at-1 fault, a solid triangle (▲) besides a rail on the other hand indicates this rail is at a normal logic 1.

As shown in Fig. 19, the stuck-at-1 fault on I_1 causes consecutive signals to a input of the R-Counter at the top, which in turn causes output O_2 to always be logic 1. In general, if the single stuck-at-1 fault occurs on input I_k, the output O_m will always be logic 1, where $m = k + 1$ if $k \neq 3$ otherwise $m = 1$.

Now consider a normal signal comes to input I_2 when I_1 is stuck-at-1. Figure 20 shows that if I_k is stuck-at-1 and a normal input comes to I_j, a normal output will be generated at O_{6-j-k} whereas the faulty output O_m will always be 1, where $m = k + 1$ if $k \neq 3$ otherwise $m = 1$.

If after I_1 is stuck-at-1, normal signals come to both input I_2 and I_3, the effect is shown in Fig. 21.

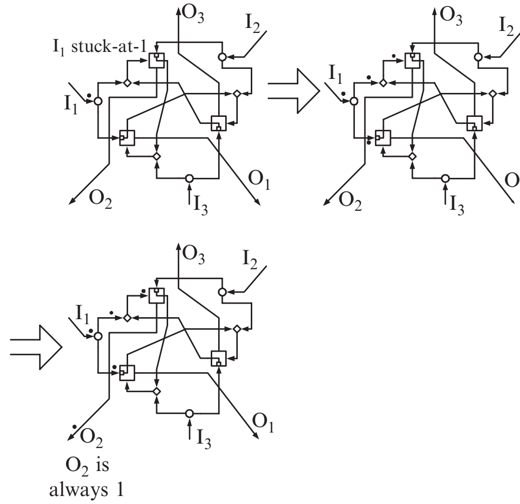


Fig. 19. Single stuck-at-1 fault in TRIA

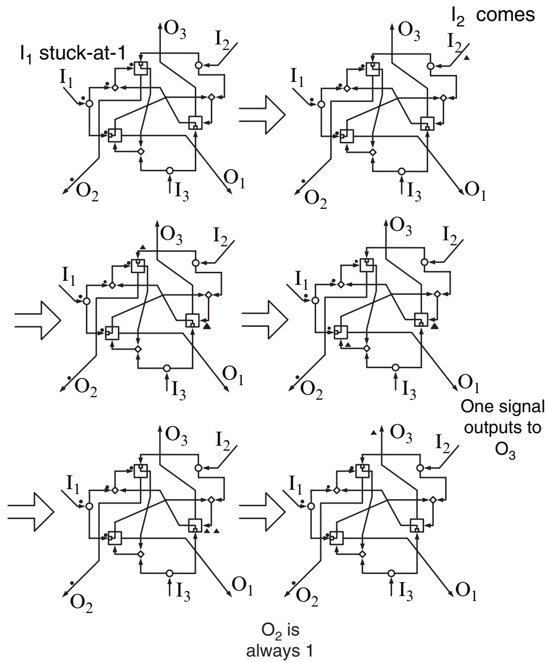


Fig. 20. Single stuck-at-1 fault in TRIA with one normal input

It can be seen from Fig. 21, if I_k is stuck-at-1 and two signals come at the other two inputs, a normal output will be generated at O_t with the faulty output O_m always being 1, where $m = k + 1$ if $k \neq 3$ otherwise $m = 1$, and $t = k - 1$ if $k \neq 1$ otherwise $t = 3$. No output will be generated at O_k.

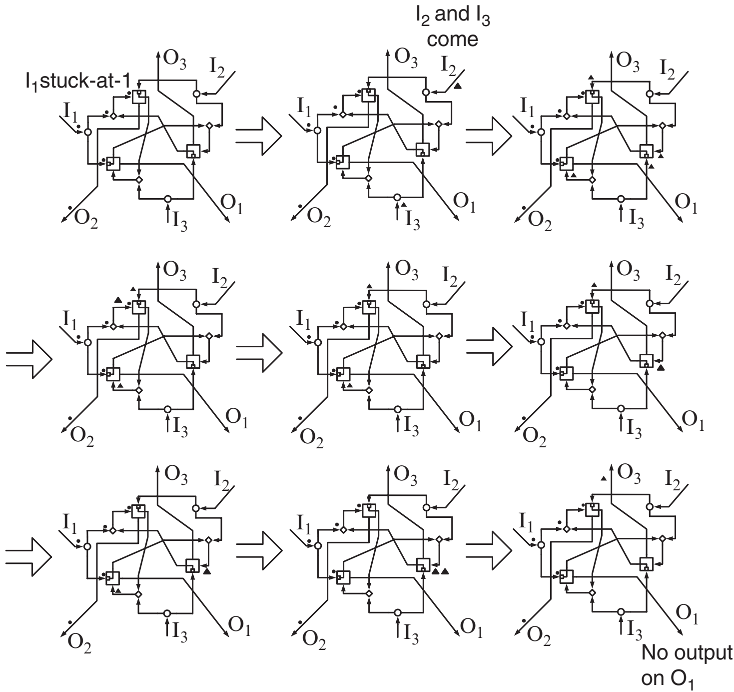


Fig. 21. Single stuck-at-1 fault in TRIA with two normal inputs

Double Stuck-at-1 Faults in TRIA

The effect of two inputs are stuck-at-1 is shown in Fig. 22. If I_j and I_k are stuck-at-1, O_{6-j-k} will always be logic 1. If then a normal signal comes at the third input, it will have no effect on the outputs as explained in Fig. 23.

Triple Stuck-at-1 Faults in TRIA

Figure 24 shows if all three inputs are stuck-at-1, no output will be generated. The signals (shown as dots) circulate inside the circuit, i.e. a loop is formed.

Stuck-at-1 Faults in AND Gate

The Reed–Muller cell has three AND gates – two in the multiplexer and one separate. Each AND gate has two TRIAs. If A and B are the two dual-rail inputs to the delay-insensitive AND gate and A is at Data 1 and B is at Data 0, thus A^1 is 1, A^0 is 0, B^1 is 0, and B^0 is 1. Therefore the output C should be at Data 0, i.e. C^1 is 0 and C^0 is 1. However, if a stuck-at-1 fault occurs on B^1, the effect is shown in Fig. 25.

Most faults can be detected on-line, i.e. during the normal operation of a delay-insensitive circuit. However, there are certain faults can only be detected

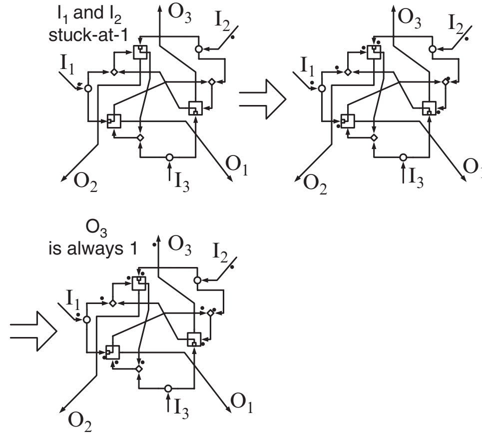


Fig. 22. Double stuck-at-1 faults in TRIA

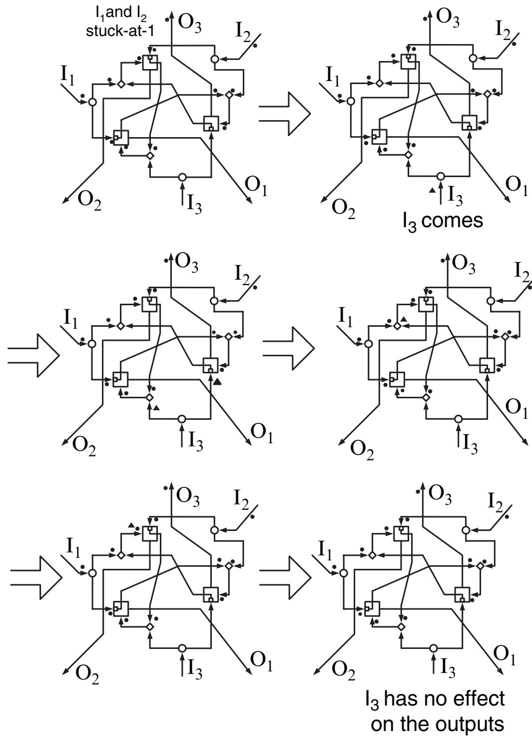


Fig. 23. Double stuck-at-1 faults in TRIA with a normal input

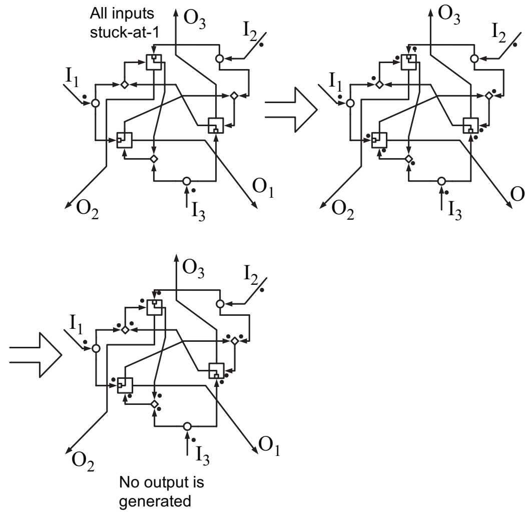


Fig. 24. Triple stuck-at-1 faults in TRIA

during off-line testing of a circuit. As explained previously (Fig. 25), an AND gate outputs a wrong (but valid) data value due to the presence of the stuck-at-1 fault. Since this value is still valid it cannot be detected on-line by monitoring the output. For off-line testing of a large delay-insensitive circuit, a set of test patterns that provides both reasonable number of tests and high fault coverage is extremely difficult to generate.

In our approach, circuits are designed using the proposed Reed–Muller cell. Based on the Reed–Muller expansion theory discussed in Sect. 4, the number of test patterns for off-line testing is proportional to the total number of inputs and all single stuck-at faults in the circuit are covered. Therefore our approach makes it possible to detect those faults like the one shown in Fig. 25 by off-line testing, thus improving the overall testability of nanoscale delay-insensitive circuits on cellular arrays.

The effect of stuck-at faults in an OR gate and an XOR gate can be analyzed in the same manner. As in an AND gate, certain stuck-at faults in OR gate and XOR gate cannot be detected on-line. However, those can be detected off-line because of the Reed–Muller based circuit structure.

8 Comparison with Previous Work

So far only limited amount of research has been done on designing and laying out delay-insensitive circuits on cellular arrays, testing of delay-insensitive circuits has not been addressed at all. In [1] a five-state asynchronous cellular automaton was presented and its behavior was discussed. Five primitives were proposed with twelve transitions rules describing their behavior. The

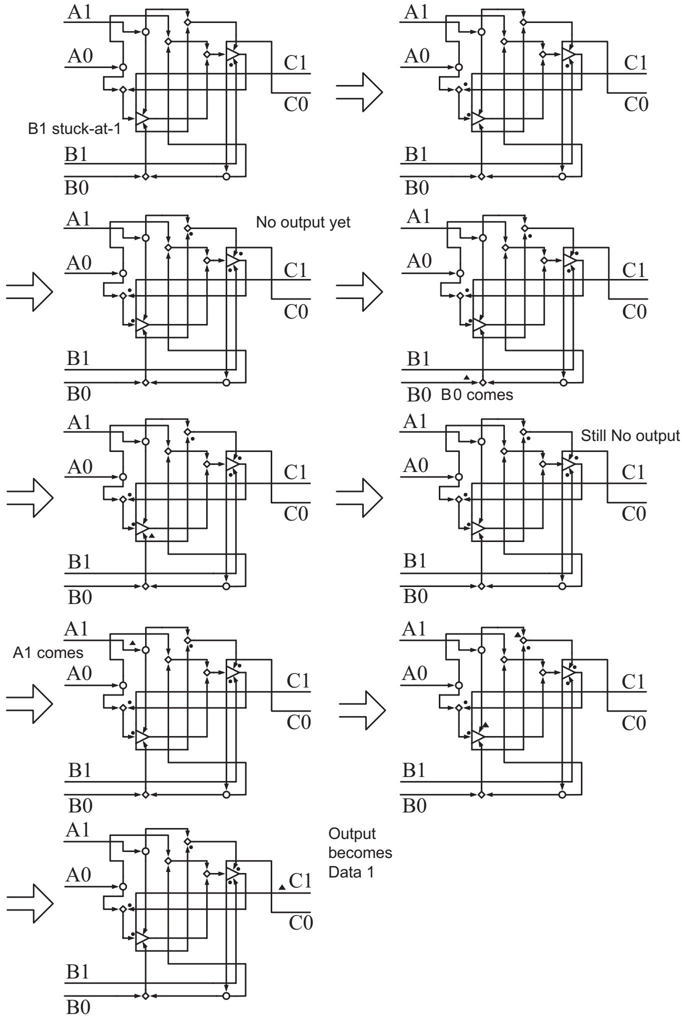


Fig. 25. Single stuck-at-1 fault in AND gate example

large numbers of primitives and transition rules make it difficult to use them for designing delay-insensitive circuits, e.g. any transition including signal propagation needs five stages to complete. Two even more complicated structures were introduced in [2] and [3]. In [2] a Hexagonal Cellular Automaton (HCA) was presented. Each cell in HCA has six bits and six neighbors. The Reversible HCA has 64 states. In [3] each cellular automaton has six states, and thirteen transition rules were needed to propagate a signal. Lee et al. [4] used the same cell structure as in [3] with different transition rules. Layout of delay-insensitive circuits on cellular arrays was discussed in [5]. However, it

used bi-directional signal paths resulting in the increase of design complexity. In [6], only the design of a NAND gate was shown and no design methodology or layout strategy was given.

As for Field-Effect Transistor (FET)-based nanoscale electronics design, an approach to build defect-tolerant, nanoscale compute fabrics of assemblies of defective crossbars of configurable FETs and switches was presented in [15]. Both n-channel FETs and p-channel FETs are arranged in crossbar architecture to implement CMOS-like logic capable of universal computation. The functional redundancy makes them tolerate both "Stuck Open" and "Stuck Close" faults although the latter one is more problematic. Research in the same category includes [16], in which a FET-based nanoarray architecture was proposed. Carbon nanotubes (CNTs), silicon nanowires (SiNWs), and molecular-scale devices were organized to form the array. Defect tolerance was provided through hierarchically sparing, e.g. fabricate more wires than it is actually needed. These approaches are very interesting and might be considered to implement cellular arrays.

9 Conclusions

The fundamental concepts of delay-insensitive circuits and layout of such circuits on cellular arrays for nanocomputing system implementation have been introduced in this chapter. A delay-insensitive Reed–Muller cell, which consists of an AND gate, an XOR gate, a 2-to-1 multiplexer, and a 1-bit register, is proposed. The successful layouts of these circuit components are also included. Arbitrary logic functions will first be transformed into Reed–Muller form; then be synthesized using a series of the proposed Reed–Muller cells.

The effect of stuck-at faults in delay-insensitive circuits has been analyzed and the improvement in testability by using the proposed Reed–Muller cells is also discussed. The next step will be investigating the fault-tolerant design methodology for delay-insensitive circuits on cellular arrays. Due to the high regularity of the cellular arrays and the flexible timing requirements of delay-insensitive circuits, this approach is an important step towards feasible nanocomputing system design.

References

1. J. Lee, S. Adachi, F. Peper, and K. Morita, "Embedding Universal Delay-Insensitive Circuits in Asynchronous Cellular Spaces," *Fundamenta Informaticae*, XX (2003) I-24, IOS Press
2. K. Morita, M. Margenstern, and K. Imai, "Universality of Reversible Hexagonal Cellular Automata," *MFCS'98 Satellite Workshop on Frontiers Between Decidability and Undecidability*, Aug. 24–25, 1998
3. S. Adachi, F. Peper, and J. Lee, "Computation by Asynchronously Updating Cellular Automata," *Journal of Statistical Physics*, Vol. 114, Nos. 1–2, Jan. 2004

4. J. Lee, F. Peper, S. Adachi, K. Morita, and S. Mashiko, "Reversible Computation in Asynchronous Cellular Automata," *Unconventional Models of Computation* (2002), LNCS 2509, pp. 220–229
5. F. Peper, J. Lee, S. Adachi, and S. Mashiko, "Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers?" *Nanotechnology*, Vol. 14, 469–485, 2003
6. F. Peper, J. Lee, F. Abo, T. Isokawa, S. Adachi, N. Matsui, and S. Mashiko, "Fault-tolerance in nanocomputers: a cellular array approach," *IEEE Transactions on Nanotechnology*, Vol. 3, No. 1, 187–201, Mar. 2004
7. P. K. Lala, *Self-checking and Fault-tolerant Digital Design*, Morgan Kaufmann, San Francisco, 2001
8. A. J. Heinrich, C. P. Lutz, J. A. Gupta, and D. M. Eigler, "Molecule cascades," *Science*, Vol. 298, 1381–1387, Nov. 2002
9. M. Renaudin and B. El Hassan, "The design of fast asynchronous adder structures and their implementation using DCVS logic," *1994 IEEE International Symposium on circuits and systems*, Vol. 4, 291–294, 1994
10. W. Kuang, "*Iterative Ring and Power-Aware Design Techniques for Self-Timed Digital Circuits*," Ph.D. dissertation, University of Central Florida, July 2003
11. J. Di, J. S. Yuan, and M. Hagedorn, "Energy-aware Multiplier Design in Multi-rail Encoding Logic," *IEEE 45th Midwest Symposium on Circuits and Systems*, Aug. 2002
12. P. Patra and D. S. Fussell, "Building-blocks for designing DI circuits," *Technical report TR93-23*, Dept. of Computer Sciences, The University of Texas at Austin, Nov. 1993
13. P. K. Lala, *Digital Circuit Testing and Testability*, Academic Press, New York, 1997
14. J. Di, P. K. Lala, and D. Vasudevan, "On the Effect of Stuck-at Faults on Delay-Insensitive Nanoscale Circuits," *Defect and Fault Tolerance in VLSI Systems Symposium 2005 (DFT 2005)*, Oct. 2005
15. G. Snider, P. Kuekes, and R. S. Williams, "CMOS-like logic in defective, nanoscale crossbars," *Nanotechnology*, Vol. 15, 881–891, 2004
16. A. Dehon, "Array-based architecture for FET-based nanoscale electronics," *IEEE Transactions on Nanotechnology*, Vol. 2, No. 1, 23–32, Mar. 2003

Chapter 8: QCA Circuits for Robust Coplanar Crossing

S. Bhanja, M. Ottavi, S. Pontarelli, and F. Lombardi

1 Introduction

Quantum-dot cellular automata (QCA) [16] may overcome some of the limitations of current technologies, while meeting the density foreseen by Moore's Law and the International Technology Roadmap for Semiconductors (ITRS). For manufacturing, molecular QCA implementations have been proposed to allow for room temperature operation; the feature of wire crossing on the same plane (coplanar crossing) provides a significant advantage over CMOS. Coplanar crossing is very important for designing QCA circuits; multi-layer QCA has been proposed [4] as an alternative technique to route signals, however it still lacks a physical implementation. At design level, algorithms have been proposed to reduce the number of coplanar wire crossings [9]. In QCA circuits, a reliable operation of coplanar crossing is dependent on the temperature of operation. Resilience to temperature variations due to thermal effects is also an important feature to consider for practical applications. A reduction in the probability of generating an erroneous signal is also of concern, hence, robustness must be addressed.

Robustness to thermal effects must consider the repeated estimates of ground (and preferably near-ground) states, along with cell polarization for different designs. This evaluation is presently possible only through a full quantum-mechanical simulation (over time) that is known to be computationally intensive. Tools such as AQUINAS [16] and the coherence vector simulation engine of QCADesigner [17] perform an iterative quantum mechanical simulation (as a self-consistent approximation, or SCA) by factorizing the joint wave function over all QCA cells into a product of individual cell wave functions (using the Hartree–Fock approximation). This results in accurate estimates of ground states, cell polarization (or probability of cell state), temporal progress and thermal effects, but also at the expense of a large computational complexity. Other techniques such as QBert [12], Fountain-Excel

simulation, nonlinear simulation [14,17], and digital simulation [17] are faster, but they only estimate the state of the cells; in some cases unfortunately, they may fail to estimate the correct ground state. Also these techniques do not fully estimate the cell polarization or take into account thermal effects. In this chapter, we use a Bayesian modeling method that allows to estimate the cell polarization for the ground state and to study the effects of thermal variations and layout defects. As introduced in [1], a Bayesian model makes possible to perform a thermal characterization of coplanar crossing; in the next sections, the Bayesian model is also amenable for simulating the combined effects of layout defects and temperature.

The objective of this chapter is to propose and analyze different circuits for QCA coplanar crossing. The coplanar crossing designs that are analyzed in this chapter are for two signals orthogonally routed on the same plane using the following circuits: (1) the coplanar crossing of [8], (2) a novel TMR-based coplanar crossing , (3) the so-called thick coplanar crossing of [3]. This chapter deals with the robust operation of these three coplanar crossing circuits to thermal variation and in the presence of cell defects; the proposed circuits utilize different features of the majority voting function of QCA circuits to route signals on a Cartesian plane. Also, they utilize different types of QCA cells (rotated and not rotated) and their immediate adjacency. The objective of this analysis is to select the coplanar crossing circuit that offers the highest performance. Finally a simulation on a full adder circuit proves that the use of the proposed crossing designs increases the thermal and defect robustness when applied to a generic circuit.

This chapter is organized as follows: Sect. 2 provides a brief overview of QCA technology, Sect. 3 introduces the Bayesian model used for temperature characterization and Sect. 4 describes the coplanar wire crossing circuits (inclusive of layouts). Section 5 provides an analysis of the designs with respect to normalized temperature, while Sect. 6 shows the simulation results for defective circuits. Section 7 shows the results of the thermal characterization of defective layouts under temperature variations while Sect. 8 analyzes the thermal and defect robustness of a full adder circuit. Finally, Sect. 9 draws the conclusion of this analysis.

2 Review of QCA

A QCA cell can be viewed as a set of four charge containers or “dots”, positioned at the corners of a square. The cell contains two extra mobile electrons which can quantum mechanically tunnel between dots, but not cells. The electrons are forced to the corner positions by Coulomb repulsion. Therefore, electrons have a preferential alignment along one of the two perpendicular cell axes, as shown in Fig. 1. The polarization δ (δ refers to polarization as P is used for defining probabilities) measures the extent of this alignment.

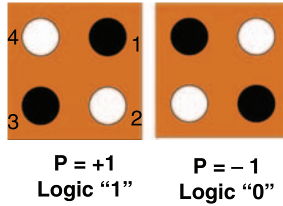


Fig. 1. QCA cell and polarization states

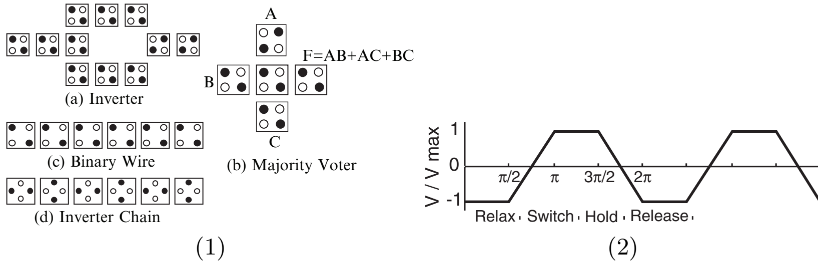


Fig. 2. (1) Basic QCA devices (2) Cmos-clock-signal

If the two extra electrons are completely localized on dots 1 and 3, the polarization is +1 (binary 1); if they are localized on dots 2 and 4, the polarization is −1 (binary 0). Tunneling between dots implies that charges may not be not completely localized and consequently, the polarization value can be not integer.

Unlike conventional logic circuits in which information is transferred by electrical current, QCA operates by the Coulombic interaction that connects the state of one cell to the state of its neighbors. This results in a technology in which information transfer (interconnection) is the same as information transformation (logic manipulation) with low power dissipation [15]. One of the basic logic gates in QCA is the so-called majority voter (MV) with logic function $\mathrm{Maj}(A, B, C) = AB + AC + BC$. MV can be realized by 5 QCA cells, as shown in Fig. 2(1b). Logic AND and OR functions can be implemented from the MV by setting an input (the so-called programming or control input) permanently to a "0" or "1" value. The inverter (INV) is the other basic gate in QCA and is shown in Fig. 2(1a). The binary wire and inverter chain (as interconnect fabric) are shown in Fig. 2(1c)(1d). In VLSI systems, timing is controlled through a reference signal (i.e., the clock), however timing in QCA is accomplished by clocking in four distinct and periodic phases [5] (as shown in Fig. 2(2)). A QCA circuit is partitioned into *serial* (one-dimensional) zones, and each zone is maintained in a phase. Clocking implements quasi adiabatic switching to ensure that the QCA cells reach the lowest energy state (or ground state) during this operation.

3 Bayesian Model

An approximate two-state model of a single QCA cell [16] is utilized. In this model, each cell can be observed to be in one of two possible states, corresponding to logical states 0 and 1. Let the probability of *observing* the ith QCA cell at state 0, be denoted by $P(X_i = 0)$ or $P_{X_i}(0)$, or simply by $P(x_i)$. Hence for *polarization*, $\delta_{X_i} = P_{X_i}(1) - P_{X_i}(0)$. The joint probability of observing a set of steady-state assignments for the cells is denoted by $P(x_1, \cdots, x_n)$. To reduce the combinatorial complexity of the analysis, the joint wave function must be considered in terms of the product of the wave function over one or two variables (i.e., the Slater determinants). This corresponds to a factored representation of the wave function (Hartree–Fock approximation) [7, 15]. As an example, consider the linear wire arrangement of nine QCA cells, shown in Fig. 3a. With no assumption, the joint state probability function can be decomposed into a product of *conditional* probability functions by the repeated use of the property that $P(A, B) = P(A|B)P(B)$ (as shown in Fig. 3d).

$$P(x_1, \cdots, x_9) = P(x_9|x_8 \cdots x_1)P(x_8|x_7 \cdots x_1) \cdots P(x_2|x_1)P(x_1) \quad (1)$$

The *radius of influence* (denoted by r) is defined as the maximum distance (normalized to the cell-to-cell distance) that allows interaction between two cells. If a 2-cell radius ($r = 2$) of influence is considered, then the conditional probability $P(x_i|x_{i-1}, \cdots, x_1)$ can be approximated by $P(x_i|x_{i-1}, x_{i-2})$, and the overall joint probability can be factored as

$$P(x_1, \cdots, x_9) = \begin{cases} P(x_9|x_8, x_7)P(x_8|x_7, x_6) \cdots P(x_2|x_1)P(x_1) & r = 2 \\ P(x_9|x_8)P(x_8|x_7) \cdots P(x_2|x_1)P(x_1) & r = 1 \end{cases} \quad (2)$$

3.1 Inferring Link Structure

The complexity of a Bayesian network representation is dependent on the order of the conditional probabilities, i.e., the maximum number of parents (N_p)

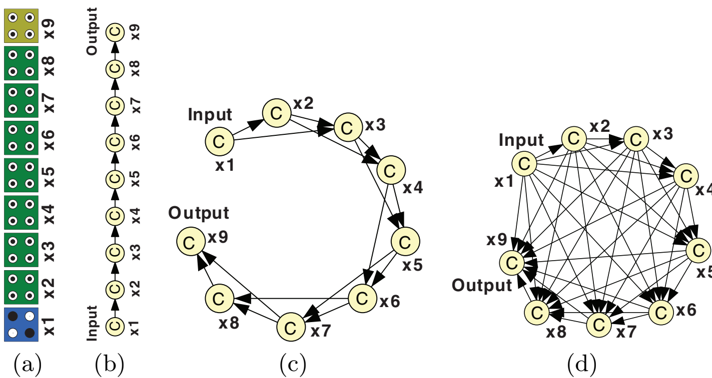


Fig. 3. Bayesian net dependency model (BN) for (**a**) 9-cell QCA wire with (**b**) 1-cell radius of influence (**c**) 2-cell radius of influence, and (**d**) all cells

for a node. The maximum size of the conditional probability table stored is 2^{N_p+1}; thus, it is important to have a representation with a minimal possible number of parents per node, while preserving all dependencies. For this representation conditional *independencies* that might exist must be used. Note that modeling all dependencies is possible by utilizing a complete graph representation; however, it is the independencies that result in a sparse graph representation. It can be shown that all conditional *independencies* among all triple subsets of variables can be captured by a directed acyclic graph (DAG) representation if the links are directed along *causal* directions [13], i.e., a parent should represent the direct causes of its children. Such minimal representations are termed Bayesian networks. A link is directed from node X to node Y, if X is a direct cause of Y. For QCA circuits, there is an inherent causal ordering among cells. Part of the ordering is imposed by the clocking zones. Cells in the previous clock zone are the drivers or the causes of the change in polarization of the current cell. Within each clocking zone, ordering is determined by the direction of propagation of the wave function [16].

Let $Ne(X)$ denote the set of all neighboring cells than can effect a cell, X. It consists of all cell within a pre-specified radius. Let $C(X)$ denote the clocking zone of cell X (as commonly assumed for phased clocking zones in QCA). Let $T(X)$ denote the time for the wave function to propagate from the nodes nearest to the previous clock zone or from the inputs, if X shares the clock with the inputs. Only the relative values of $T(X)$ are important to decide upon the causal ordering of the cells. A breadth first search strategy, outlined in Fig. 4 is employed to decide upon the time ordering, $T(X)$.

The direct causes or parents of a node X are determined based on the inferred causal ordering; this parent set is denoted by $Pa(X)$ and is logically specified as follows.

$$Pa(X) = \{Y | Y \in Ne(X), (C(Y) <_{mod4} C(X)) \vee (T(Y) < T(X))\} \qquad (3)$$

The *causes*, and hence the parents, of X are the cells in the previous clocking zone and the cells are nearer to the previous clocking zone than X. The children set, $Ch(X)$, of a node, X, are the neighbor nodes that are not parents, i.e., $Ch(X) = Ne(X)/Pa(X)$.

An important part of a Bayesian network is the conditional probabilities $P(x|pa(X))$, where $pa(X)$ represents the values taken on by the parent set, $Pa(X)$.

3.2 Quantification of Conditional Probabilities

In a four-phased clocked design [16], all cells must be placed into the ground state by systolically driving subgroups of cells (all in one clock zone) into their local ground states. So, in this respect the conditional probabilities are the probabilities of the ground states, defined locally over the Markov neighborhood of each cell, i.e., to decide upon the conditional probability of a cell

```
Variables:
 Q: queue to process cells in a breadth first order
 T(X): each cell, X, has a time tag, T(X), that is initialized to -1
 Count: counter that is incremented at each iteration.
 Clock: to keep track of the clock zone being processed.

1.  Enqueue input cells onto Q 2.  Set time tags of input cells
to -2 to denote cells in Q 3.  do repeat 4.     while Q is not
empty 5.       X = Dequeue(Q); 6        Clock = Clock(X); 7.
T(X) = Count++; 8.       Ne(X) = Neighbors of X sorted from
nearest to farthest 9.       for neighbor, Y,  in Ne(X) 10.
if Y has not been tagged, i.e., T(Y) == -1, and
                Y is in the same clock zone as X
11.                    Enqueue(Q, Y)
12.                    T(Y) = -2
13.                end if
14.       end for
15.    end while
16.    Enqueue cells onto Q that are adjacent to cells in Clock zone
17.    Set the time tags of these cells to -2;
18.  while (Q is not empty)
```

Fig. 4. Breadth-first search algorithm to establish the causal order of the QCA cells

state given the states of the parent nodes, $P(X = 1|Pa(X) = pa(X))$. Hence, all cells within the Markov neighborhood, $Ne(X)$ must be considered. This includes the cells that are the parents in the Bayesian network representation $Pa(X)$ and also the children, $Ch(X)$. The states of these parents are fixed at the conditioned state assignments $pa(X)$; however, the states of the children are unspecified. As a clocked circuit is modelled (in this circuit the phased clock design keeps the cells at their ground states in each clocking epoch), then the polarization of X and $Ch(X)$ is chosen such that given the parent states, the energy (Hamiltonian) in the local neighborhood is minimized. A quantum mechanical formulation is effectively achieved.

An array of cells can be modeled by considering the cell-level quantum entanglement of the two states and the Coulombic interaction of nearby cells (that is modeled using the Hartee–Fock (HF) approximation [10, 16]). The HF model approximates the joint wave function over all cells by the product of the wave functions over individual cells (actually the sum of permutations of the individual wave functions by their Slater determinant). This allows to characterize the evolution of the individual wave functions. The evolution of the wave function of the cell X in the local neighborhood $Ne(X)$ is of interest.

Let denote the eigenstates of a cell corresponding to the 2-states by $|0\rangle$ and $|1\rangle$. The state at time t, that is referred to as the wave-function and denoted by $|\Psi(t)\rangle$, is a linear combination of these two states, i.e., $|\Psi(t)\rangle = c_0(t)|0\rangle + c_1(t)|1\rangle$. The coefficients are function of time. The expected value of

any observable, $\langle \hat{A}(t) \rangle$, can be expressed in terms of the wave function as $\langle \hat{A} \rangle = \langle \Psi(t)|\hat{A}(t)|\Psi(t)\rangle$ or equivalently as $\mathrm{Tr}[\hat{A}(t)|\Psi(t)\rangle\langle\Psi(t)|]$, where Tr denotes the trace operation, $\mathrm{Tr}[\cdots] = \langle 0|\cdots|0\rangle + \langle 1|\cdots|1\rangle$. The term $|\Psi(t)\rangle\langle\Psi(t)|$ is known as the density operator, $\hat{\rho}(t)$. The expected value of an observable of a quantum system can be computed if $\hat{\rho}(t)$ is known.

The entries of the density matrix, $\rho_{ij}(t)$, can be shown to be defined by $c_i(t)c_j^*(t)$ or $\rho(t) = \mathbf{c(t)c(t)}^*$, where * denotes the conjugate transpose operation. The density matrix is Hermitian, i.e., $\rho(t) = \rho(t)^*$; each diagonal term, $\rho_{ii}(t) = |c_i(t)|^2$, represents the *probabilities* of finding the system in state $|i\rangle$. It can be easily shown that $\rho_{00}(t) + \rho_{11}(t) = 1$. These two entries of the density matrix are pertinent to logic modeling; ideally, these probabilities should be zero or one. For QCA device modeling, the *polarization* index (P) is commonly used, i.e., $\rho_{00}(t) - \rho_{11}(t)$ as the difference of the two probabilities in a range between -1 and 1.

The density operator is a function of time, $\hat{\rho}(t)$, and its dynamics is captured by the Loiuville equation or the von Neumann equation, that can be derived from the basic Schrodinger equations to capture the evolution of the wave function over time, $\Psi(t)$.

$$\begin{aligned} i\hbar\tfrac{\partial}{\partial t}\rho(t) &= i\hbar\tfrac{\partial}{\partial t}\mathbf{c(t)c(t)}^* \\ &= \mathbf{H}\rho(t) - \rho(t)\mathbf{H} \end{aligned} \tag{4}$$

where $\mathbf{H}$ is a 2 x 2 matrix representing the Hamiltonian of the cell. For QCA cells, it is common to assume only Columbic interaction between cells and use the Hartree–Fock approximation to arrive at the matrix representation of the Hamiltonian given by [16]

$$\mathbf{H} = \begin{bmatrix} -\frac{1}{2}\sum_{i\in Ne(X)} E_k\delta_i f_i & -\gamma \\ -\gamma & \frac{1}{2}\sum_{i\in Ne(X)} E_k\delta_i f_i \end{bmatrix} \tag{5}$$

where the sums are over the cells in the local neighborhood, $Ne(X)$. E_k is the energy cost of two neighboring cells having opposite polarizations; this is also referred to as the "kink energy". f_i is the geometric factor capturing the electrostatic fall off with distance between cells. δ_i is the polarization of the ith neighboring cell. The tunneling energy between the two states of a cell, that is controlled by the clocking mechanism, is denoted by γ.

In the presence of inelastic dissipative heat bath coupling (open world), the system moves towards the ground state [10, 16]. At thermal equilibrium, the steady-state density matrix is given by

$$\rho^{ss} = \frac{e^{-\mathbf{H}/kT}}{\mathrm{Tr}[e^{-\mathbf{H}/kT}]} \tag{6}$$

where k is the Boltzman constant and T is the temperature. Of particular interest are the diagonal entries of the density matrix, that express the probabilities of observing the cell in the two states. They are given by

$$\rho_{11}^{ss} = \frac{1}{2}\left(1 - \frac{E}{\Omega}\tanh(\Delta)\right)$$
$$\rho_{22}^{ss} = \frac{1}{2}\left(1 + \frac{E}{\Omega}\tanh(\Delta)\right) \tag{7}$$

where $E = \frac{1}{2}\sum_{i \in Ne(X)} E_k \delta_i f_i$, the total kink energy at the cell, $\Omega = \sqrt{E^2 + \gamma^2}$, the energy term (also known as the Rabi frequency), and $\Delta = \frac{\Omega}{kT}$, is the thermal ratio. These probabilities are used for establishing the minimum energy ground state values. This is determined by the eigenvalues of the Hamiltonian (5) that are $\pm\Omega$, a function of the kink energy with the neighbors. However, the states (or equivalently, polarization) of only the parents are specified in the conditional probability that we seek. The polarization of the children are unspecified. The children states (or polarization) are chosen such that Ω is maximized, i.e., to minimize the ground state energy over all possible ground states of the cell. Thus, the chosen children states are

$$ch^*(X) = \arg\max_{ch(X)} \Omega = \arg\max_{ch(X)} \sum_{i \in (Pa(X) \cup Ch(X))} E_k \delta_i f_i \tag{8}$$

The steady state density matrix diagonal entries (7) with these children state assignments are used to decide upon the conditional probabilities in the Bayesian network (BN):

$$P(X = 0 | pa(X)) = \rho_{11}^{ss}(pa(X), ch^*(X))$$
$$P(X = 1 | pa(X)) = \rho_{22}^{ss}(pa(X), ch^*(X)) \tag{9}$$

It is presently possible to estimate the polarization and ground state probabilities through a full quantum-mechanical simulation of the system evolution over time, that is known to be computationally intensive. Tools such as AQUINAS [16] and the coherence vector engine of QCADesigner [17] perform an iterative quantum mechanical simulation (self-consistent approximation, SCA) by factorizing the joint wave function over all cells into a product of individual cell wave functions exploiting the Hartree–Fock approximation. These approaches obtain accurate results for the computation of ground states, cell polarization (or probability of cell state), temporal progress, and thermal effects, but they are slow. In addition, they cannot estimate the near-ground state configurations, that are important for analyzing the sensitivity of circuits to parametric variations (such as temperature). Other tools such as QBert [12], nonlinear simulation [17], and digital simulation [17] are fast iterative schemes; however, they just estimate the state of the cells and in some cases, some fail to estimate the correct ground state. Moreover, they do not estimate the cell polarization and cannot take into account temperature effects. The BN simulator presented in [2] is used in this work because it is is very efficient in terms of computational complexity and its features are well suited to analyze parametric variations in the operation of a circuit. In particular this simulator provides the following features as outcome:

1. The ground state configuration;
2. The polarization of each cell;
3. The probability of the near-ground state (next to the lowest one) to study sensitivity to the most probable erroneous behavior (such as due to variations in operating parameters or defects);
4. The feature of each type of cell (rotated and non-rotated) to ensure robustness in operation;
5. The study of the thermal characteristics of a QCA circuit.

4 Coplanar Crossing Circuits

Coplanar wire crossing is one of the most interesting features of QCA; it allows for the physical intersection of horizontal and vertical QCA wires on the same plane, while retaining logic independence in their values; the vertical wire is implemented by rotating the QCA-cells at 45°, i.e., by means of an inverter chain. The feature of this structure is that the information along the vertical wire does not interact with the horizontal, wire. Crossing is obtained by *interrupting* either the horizontal, or the vertical wire; these interruptions are hereafter also referred to as *cuts*. Switching of the signals is accomplished by the four phased clock through the release phase.

As in previous papers in the technical literature, layouts are considered to be in a single clocking zone. The outputs are evaluated when the ground state is attained by quasi adiabatic switching. A different approach [6] proposes the vertical and horizontal waves alternatively passing through an intersection. While this approach has the interesting feature of exploiting the intrinsically pipelined behavior of QCA, crossings in a single clocking zone require less area and a simple clocking circuitry.

A set of three layouts for the coplanar crossing analogous to that introduced in [1] is hereafter analyzed.

1. *Normal crossing: this is based on the orientation of the cells.*
2. *TMR crossing: this is based on the voting nature in the QCA layout.*
3. *Thick crossing: this is based on the interaction among cells in an enlarged wire.*

For normal crossing, the cell orientation is interrupted on the central cell of either the horizontal (A line), or vertical line (B line). For the other two circuits, the cell orientation is interrupted on the horizontal (A line), or vertical line (B line).

4.1 Normal

The normal coplanar crossing circuit can have two arrangements (shown in Figs. 5 and 6) as corresponding to the employed cut. This circuit has been

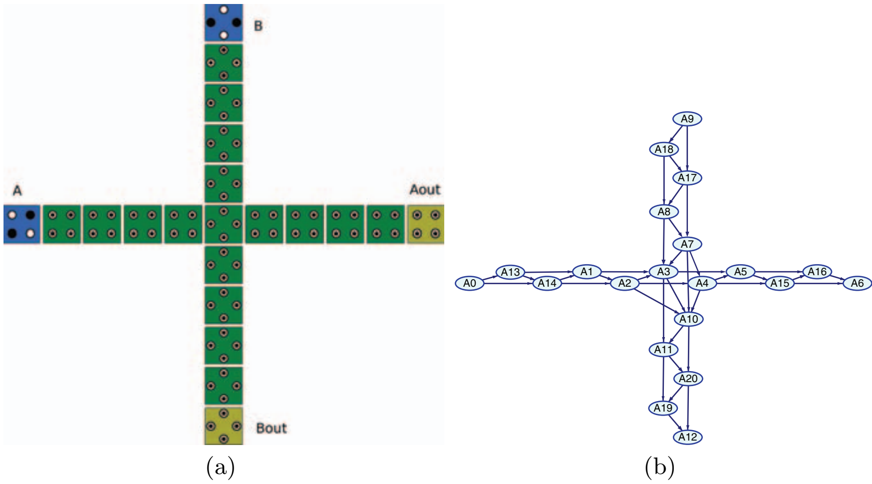


Fig. 5. Normal crossing with rotated central cell (Xa) (**a**) Layout (**b**) BN with 2-cell radius of influence

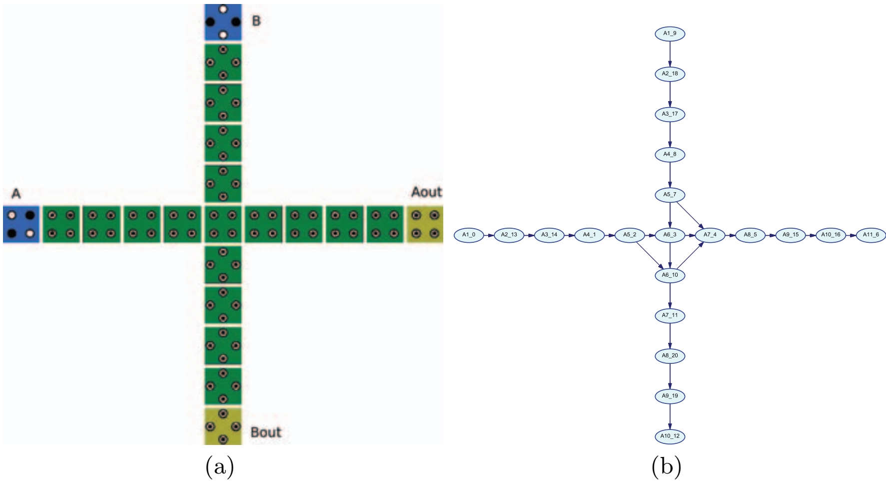


Fig. 6. Normal crossing with rotated central cell (Xb). (**a**) Layout (**b**) BN with 2-cell radius of influence

proposed in [8]; it has been shown that an horizontal wire (with input A and output $Aout$) can be crossed with a vertical inverter chain (with input B and output $Bout$) with no interference among wires.

These arrangements differ by the orientation of the cell at the crossing point: Xa in Fig. 5(a) has the central cell rotated by 45°, Xb in Figs. 6(a) has a non-rotated cell. Figures 5(b) and 6(b) show the BN for analyzing these two arrangements. Note that only the BN shown in Fig. 5(b) reports the actual

number of connections which account for a radius of influence of two; hereafter in this chapter, all BNs are simplified for improved readability of the figures.

4.2 TMR

A simple approach for implementing robust crossing in QCA is to take advantage of the inherent voting characteristic of this technology. The QCA wire is split through fanout, crossed and then re-converged and voted by a MV which performs a TMR voting function of the signals.

Two types of arrangement for the TMR based coplanar crossing circuit are proposed:

1. 3-to-1 TMR;
2. 3-to-3 TMR.

In the 3-to-1 TMR shown in Fig. 7 and associated BN, voting occurs along the direction on which the cell rotation is interrupted, thus producing two different arrangements TMR_Xa for voting the *A* line and TMR_Xb for voting the *B* line (shown in Fig. 7).

If both wires are split and reconverged, the more complex 3-to-3 (triple) TMR (as shown in Fig. 8 with corresponding BN) is applicable. The triple TMR has also two arrangements: double_TMR Xa (Fig. 8) for the interrupted *A* line direction, and double_TMR Xb for the interrupted *B* line direction. The 3-to-3 TMR utilizes a larger number of cells (92 vs. 41) than the 3-to-1 TMR.

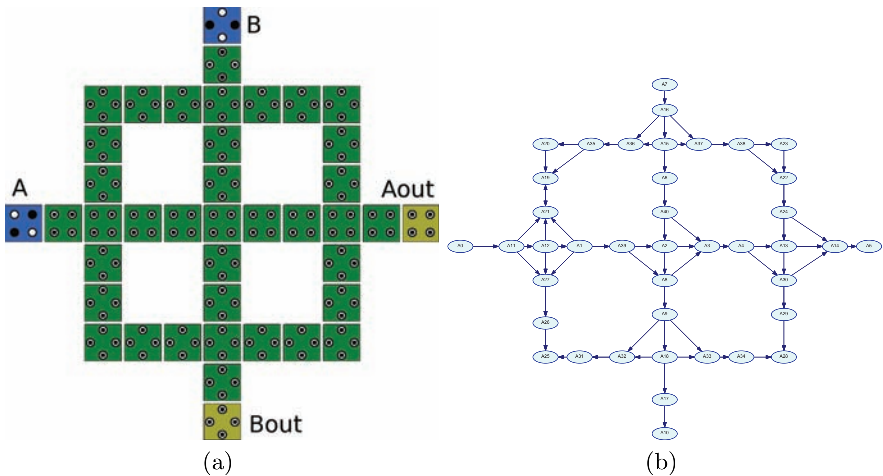


Fig. 7. 3×1 TMR crossing with non-rotated central cell (TMRXb voting on the B line). (**a**) Layout (**b**) BN with 2-cell radius of influence

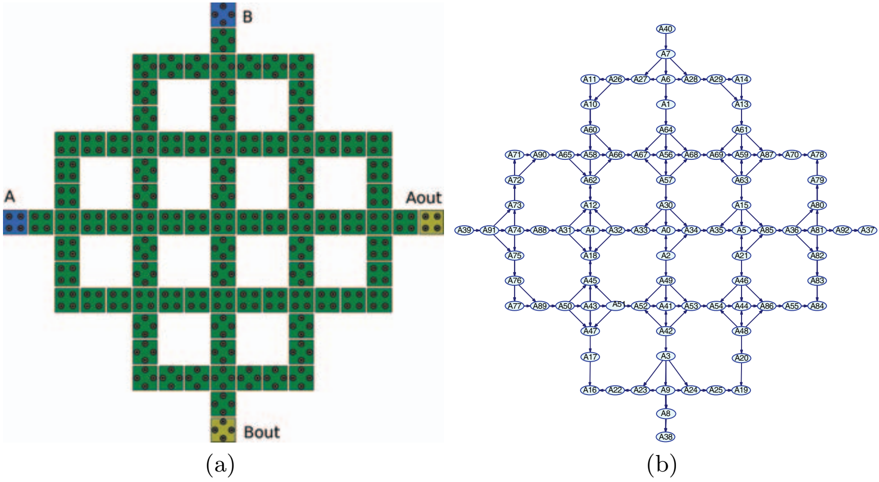


Fig. 8. 3×3 TMR crossing with rotated central cell (**a**) Layout (**b**) BN with 2-cell radius of influence

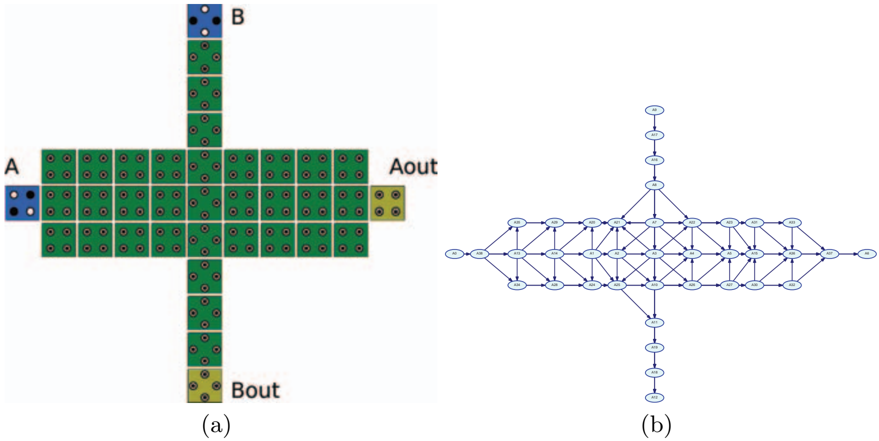


Fig. 9. Thick horizontal crossing. (**a**) Layout (**b**) BN with 2-cell radius of influence

4.3 Thick

A coplanar crossing circuit that is still based on TMR voting, has been proposed in [3] and is hereafter referred to as thick crossing. Differently from TMR, in thick crossing the fanout of the three wires generates a “thick” wire that has a width of three cells; crossing between wires is performed by interrupting the thick wire with a single wire whose cells are rotated with respect to the thick wire. Figures 9 and 10 show these arrangements together with

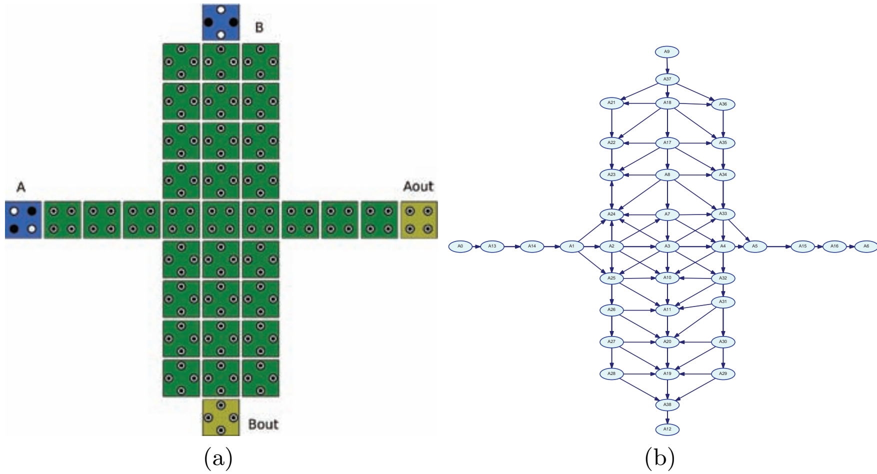


Fig. 10. Thick vertical crossing. (**a**) Layout (**b**) BN with 2-cell radius of influence

the corresponding BN for horizontal and vertical crossings. A thick circuit requires 37 QCA cells.

5 Temperature Characterization

This section presents the simulation results using the Bayesian network of the proposed coplanar crossing circuits with respect to temperature. All plots start from the correct (expected) value of the output; this output value tends to 0 when the normalized temperature tends to one, i.e., when the temperature is such that $kT \simeq E_k$ (the thermal energy is equal to the kink energy) and the two extra electrons are delocalized. The increase in temperature has different effects on the layouts, therefore allowing to define a metric. Figures 12–15 show the output value vs. temperature for the previously introduced circuit arrangements when considering the exhaustive combinations of the A, B inputs, i.e., $(0,0)$ $(1,0)$ $(0,1)$ and $(1,1)$, respectively. The plots show the robustness of the proposed designs with respect to a temperature increase: a steep slope at the output to reach the zero polarization accounts for an inefficient temperature solution, while a smooth slope shows a good temperature performance. A quantitative metric for evaluating the performance of the different arrangements is also introduced by taking into account the increase of normalized temperature for a drop in output polarization from 90 to 10% of the nominal value. This metric is referred to as Thermal robustness (Th) and is defined as

$$Th = \Delta T_{\Delta P_{90-10}}$$

Table 1. Thermal robustness of Aout for the different circuits

A	B	Xa	Xb	TMRXa	TMRXb	dblTMRXa	dblTMRXb	ThickXa	ThickXb	Average
0	0	0.355	0.383	0.429	0.383	0.263	0.35	0.457	0.383	0.375
0	1	0.355	0.383	0.429	0.383	0.263	0.35	0.457	0.383	0.375
1	0	0.355	0.383	0.429	0.383	0.263	0.35	0.457	0.383	0.375
1	1	0.355	0.383	0.429	0.383	0.263	0.35	0.457	0.383	0.375

Table 2. Thermal robustness of Bout for the different circuits

A	B	Xa	Xb	TMRXa	TMRXb	dblTMRXa	dblTMRXb	ThickXa	ThickXb	Average
0	0	0.543	0.474	0.543	0.54	0.46	0.33	0.543	0.679	0.514
0	1	0.543	0.474	0.543	0.54	0.46	0.33	0.543	0.679	0.514
1	0	0.543	0.474	0.543	0.54	0.46	0.33	0.543	0.679	0.514
1	1	0.543	0.474	0.543	0.54	0.46	0.33	0.543	0.679	0.514

Tables 1 and 2 report the Th computed for $Aout$ and $Bout$, respectively, for the considered coplanar crossing circuits; a higher value accounts for better performance.

The following observation can be drawn from analyzing the plots and tables:

(1) In all circuit arrangements, thermal robustness is not affected by the input values, i.e., there is no relation between polarization levels for boolean states and temperature;
(2) In all circuit arrangements, the outputs along the uninterrupted direction behave in a similar fashion: for example, in the A direction ThickXb, Xb and TMRXb result in the same Th, because there is no interrupted wire in such direction.
(3) The double TMR layout has always the lowest performance along the interrupted direction, i.e., dblTMRXa has the lowest Th value in Table 1, while dblTMRXb has the lowest Th value in Table 2.
(4) Thick crossings have always the highest performance along the interrupted direction.
(5) Cuts reduce performance, for example double TMRXa has a lower performance than TMRXa.

In general, the Th of $Bout$ is higher than $Aout$ for the same circuit design. This is also applicable if "uncut" circuit arrangements are compared. For example, in Table 1, $Aout$ for Xb is 0.383, while in Table 2, $Bout$ for Xa is 0.543. The last observation can be explained as follows. The kink energy between two cells is determined by the difference in energy between the higher and the lower energy configurations. Assume two possible states for each cell; then the two possible energy configurations for two cells are shown in Fig. 11.

The energy of each configuration is computed by summing the Coulomb energies between the dots in the cells:

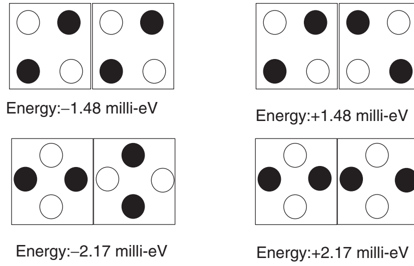


Fig. 11. Configuration energies for normal (*top row*) and rotated (*bottom row*) cells (the lowest energy configurations on the left, the highest energy configurations on the right)

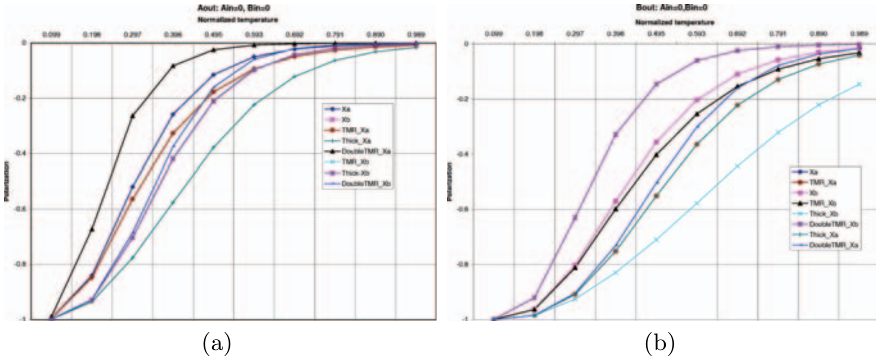


Fig. 12. Output polarization vs. normalized temperature for A=0 B=0. (a) *Aout* (b) *Bout*

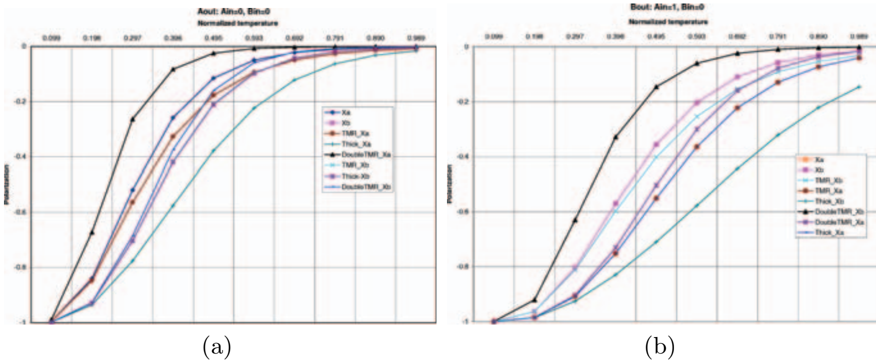


Fig. 13. Output polarization vs. normalized temperature for A=1 B=0 (a) *Aout* (b) *Bout*

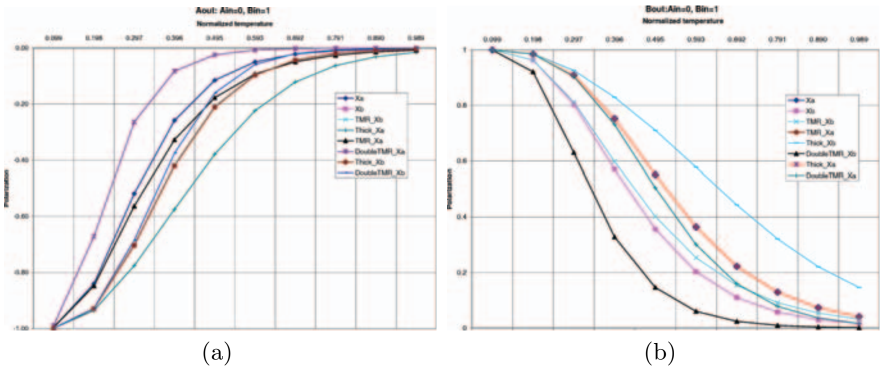


Fig. 14. Output polarization vs. normalized temperature for A=0 B=1. (**a**) *Aout* (**b**) *Bout*

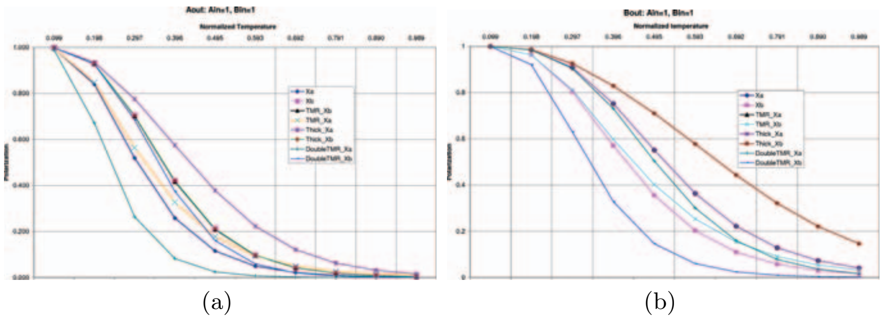


Fig. 15. Output polarization vs. normalized temperature for A=1 B=1. (**a**) *Aout* (**b**) *Bout*

$$E_{12} = \sum_{i=1}^{4} \sum_{j=1}^{4} \frac{q_{1i} q_{2i}}{4\pi\epsilon\epsilon_r} \frac{1}{d_{ij}} \tag{10}$$

The charge at the ith dot of the first cell is denoted by q_{1i}, and the distance between the ith dot in the first cell and the jth dot in the second cell is denoted by d_{ij}. On the assumption that there exists a $-1/2q$ charge at each black dot and $+1/2q$ at the white dots, the overall charge of a cell is zero. The kink energy for the normal cell is 2.96 meV, while the energy of the rotated cell is higher at 4.34 meV. The difference in kink energy is due to the distance between the dots for the two cell types. The distance between two dots in a normal cell is greater than for a rotated cell. Therefore, this suggests that a rotated cell is thermally more stable than a non-rotated one.

6 Single Defect Characterization

In this section, the coplanar crossing circuits are analyzed with respect to the occurrence of a single missing cell defect. It has been shown in [11] that missing cell placement (as defects) contribute to the almost totality of the logic faults occurring in molecular QCA circuits. Results have been obtained by modifying the Bayesian networks of the coplanar crossing circuits to simulate the absence of cells and record the logic faults due to these defects. Each circuit has been simulated for all possible single missing cell defects under the exhaustive combinations of inputs and at T=10 K (with a normalized temperature ratio (kT/Ek)= 0.198), i.e., the highest value (as found previously) prior to the steep drop in performance.

An example of the different effects of QCA cells is shown in Fig. 16 in which the case of the polarization of the outputs A and B for inputs $1, 1$ is provided for TMRXahor. The data in Fig. 16 shows that the effects of a fault are (a) a strong and mild lack of polarization and (b) a strong and mild inversion at the outputs.

Table 3 reports the results of simulation for all circuit arrangements; the incorrect outputs are either inverted or undetermined (when the polarization is under the threshold of uncertainty given by 0.1). In Table 3, the results are specified by the number of defective cells resulting in faults on the outputs for each proposed crossing layout.

From the analysis of the simulation results of Table 3 it is evident that as expected, inversion always happens in the B direction (as corresponding to an inverter chain). Moreover, the following conclusions can be drawn.

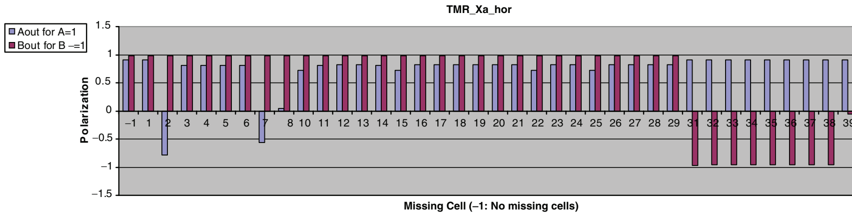


Fig. 16. TMRXahor

Table 3. Defect effects for the different circuits

Circuit	Number of cells	Ain=0 Bin=0 Inv.	Undet.	Ain=1 Bin=0 Inv.	Undet.	Ain=0 Bin=1 Inv.	Undet.	Ain=1 Bin=1 Inv.	Undet.
Xa	17	8	6	8	6	8	6	8	6
Xb	17	3	6	3	6	3	6	3	6
TMRXaHor	37	4	8	4	8	4	8	4	8
TMRXbHor	37	1	4	1	4	1	4	1	4
ThickXa	35	10	2	10	2	10	2	10	2
ThickXb	35	1	2	1	2	1	2	1	2

Table 4. Fault percentages for circuits

Circuit	Fault occurrence (%)		
	Inversion	Undetermined	Total
Xa	47.1	35.3	82.4
Xb	17.6	35.3	52.9
TMRXaHor	10.8	21.6	32.4
TMRXbHor	2.7	10.8	13.5
ThickXa	28.6	5.7	34.3
ThickXb	2.9	5.7	8.6

1. Faults appear at the output independently of the values of the inputs, thus a fault can be detected by any test vector.
2. ThickXb shows the highest performance with respect to a single missing cell defect.
3. Xa shows the lowest performance with respect to a single missing cell defect.

As faults are independent of the values of the inputs, the results are shown in Table 4. The percentages of occurrence for each of the two types of fault are computed as the number of single missing cells over the total number of cells that cause the fault. Also the total percentage of single missing cells causing any type of fault is reported as the sum of the percentage of occurrence of any of the two faults.

The results reported in Table 4 show that the coplanar crossing circuits that present the highest resilience to defects, are ThickXb and TMRXbHor.

7 Thermal Characterization of Defective Circuits

In the previous sections, defect free circuits with respect to temperature and at a given temperature have been evaluated. In this section, the circuits that have shown the highest resilience to defects are considered further to assess whether the presence of a defect increases the loss of correct polarization at the outputs with an increase of temperature.

The analysis has been performed on the circuits that in the previous section have shown the highest performance, i.e., ThickXb and TMRXbHor. Figure 17 shows the simulation results; as observed previously, the values of the inputs have no effect, so the results show no inversion when positive (upper half of the figure) and inversion when negative (lower half of the figure). Therefore, the following conclusions can be drawn:

1. Both circuits present two inversions;
2. ThickXb has in almost all cases a better thermal robustness than TMRXb.

The last result and the assumption of randomly distributed defects imply that ThickXb should be preferred as coplanar crossing circuit because on

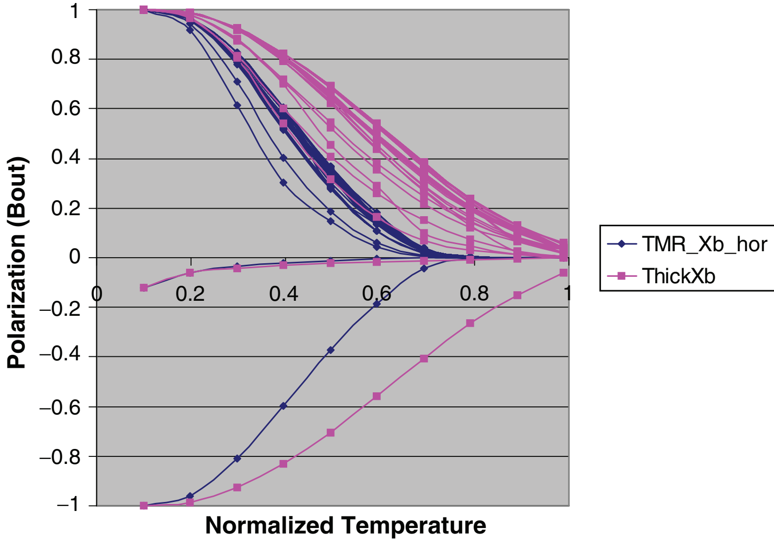


Fig. 17. Polarization vs. temperature for different defects

Table 5. Thermal robustness for *Aout* in presence of defects

A	*B*	TMRXb			ThickXb		
		Minimum	Median	Maximum	Minimum	Median	Maximum
0	0	0.146	0.268	0.347	0.192	0.257	0.280
0	1	0.147	0.267	0.291	0.192	0.259	0.280
1	0	0.147	0.280	0.347	0.192	0.259	0.279
1	1	0.146	0.265	0.280	0.192	0.251	0.280

Table 6. Thermal robustness for *Bout* in presence of defects

A	*B*	TMRXb			ThickXb		
		Minimum	Median	Maximum	Minimum	Median	Maximum
0	0	0.307	0.337	0.350	0.3578	0.563	0.602
0	1	0.297	0.336	0.349	0.358	0.563	0.602
1	0	0.298	0.337	0.349	0.358	0.563	0.602
1	1	0.307	0.337	0.349	0.358	0.563	0.602

average its thermal robustness is better than TMRXb. To better understand the behavior of these circuits in the presence of defects and resulting faults, the Thermal robustness (Th) (as defined in Sect. 5) has been computed for each simulated defect. The minimum, maximum and median values of Th for the defective circuits has been reported in Tables 5 and 6. Even if the selected circuits have a good thermal robustness for almost all simulated defects, those defects that produce as fault an inverted value at the outputs, are serious, because the inversion appears also at low temperature. The erroneous outputs

appear across the whole temperature range and therefore for these defects, thermal robustness is not fully accounted. The values reported in the tables are computed only for the non-inverting defects and the range of Th can be used to provide a quantitative comparison of the robustness of ThickXb and TMRXb.

Tables 5 and 6 show that on the interrupted direction, both circuits behave in a similar manner for the A direction, even though ThickXb shows a higher minimum value; for the B direction ThickXb outperforms the TMR circuit. The smallest value of ThickXb is higher than the highest value of TMRXb, corresponding to a better behavior for all possible missing cell defects.

8 Performance Analysis on a Full Adder

In this section a full adder circuit is analyzed when using the proposed arrangements for the coplanar crossing.

Figure 18 shows as an example three of the layouts using Xb TMRXb and ThickXb, respectively.

The results of the temperature analysis in Figs. 19 and 20 show that ThickXb and ThickXa have the best performance, although the difference between them is less due to the fault masking induced by the inherent signal regeneration of the cell-to-cell non-linear response of QCA. These results are closely dependent on the considered layout and that are not fully applicable in general as when considering the coplanar crossing as a stand-alone device.

The single defect characterization for a full adder using the coplanar crossings Xa TMRXa ThickXa TMRXb ThickXb is reported in Table 7.

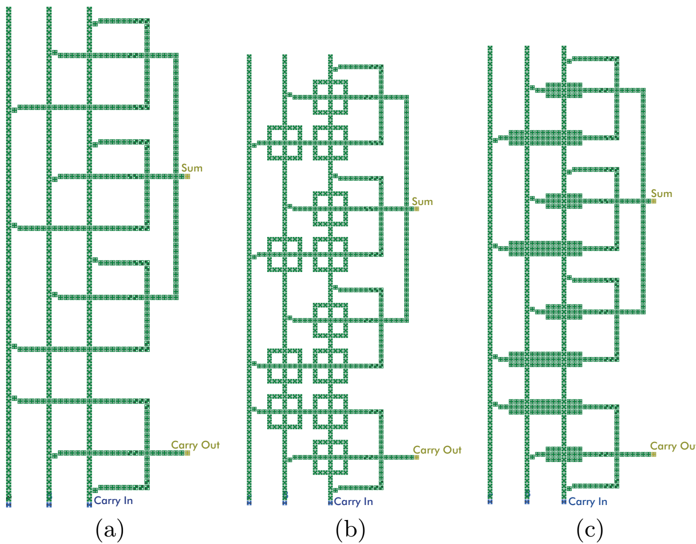


Fig. 18. Full adder. (**a**) Xb crossings (**b**) TMRXb crossings (**c**) ThickXb crossings

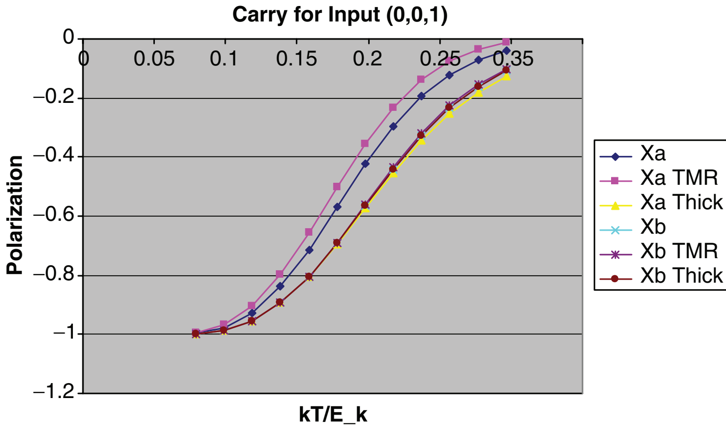


Fig. 19. Full adder: thermal performances on the carry output

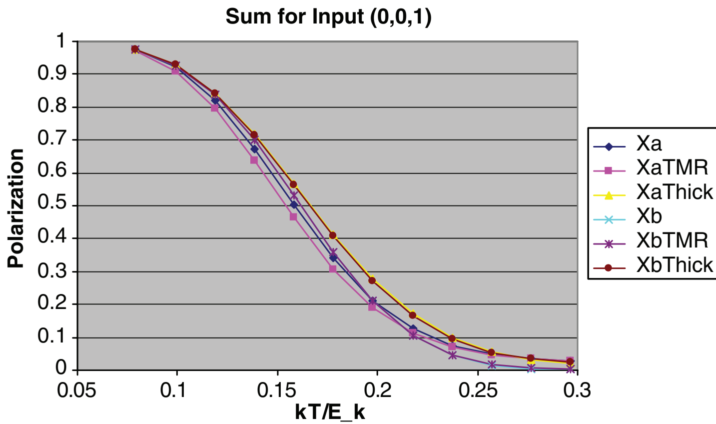


Fig. 20. Full adder: thermal performances on the sum output

Table 7. Full adder: fault injection of single missing cell

Crosswire designs	Number of cells	Sum output (inversions) for inputs				Carry output (inversions) for inputs			
		“0,0,0”	“0,1”	“0,1,0”	“0,1,1”	“0,0,0”	“0,0,1”	“0,1,0”	“0,1,1”
Xa	172	41	61	72	69	2	10	12	1
TMRXa	402	52	80	93	89	0	12	13	3
ThickXa	252	18	42	47	58	0	21	19	5
Xb	180	37	62	82	64	0	0	0	0
TMRXb	396	6	14	20	18	0	0	2	0
ThickXb	252	0	0	0	0	0	0	0	0

As done previously for each of the single crossing layouts the simulations were performed after injecting a single missing cell on the layout of the full adder. The targets of the defects were only the coplanar crossings and the

Table 8. Percentage of single missing cell faults in the crosswire designs that are detected in the sum and carry outtputs

Designs	"Correct"	"Incorrect"	"Total faults"	Fault detected %
Xa	77	95	172	55
TMRXa	279	123	402	31
ThickXa	144	108	252	43
Xb	98	82	180	46
TMRXb	370	26	396	7
ThickXb	252	0	252	0

number of simulations has been such that every single cell defect on all the coplanar crossings has been injected and its effects simulated and evaluated at the sum and carry outputs. We report the number of faults that generated error in sum outputs in columns 2–5 in Table 7 for each cross-wire design for four of the input combinations. For symmetry, results on the other four inputs are not reported. Note that a fault can generate error in sum for more than one input combinations. The same is also reported for the carry outputs in columns 6–10 in Table 7.

As could be expected, Table 7 shows that Thick crossings especially ThickXb provides the best results in terms of resilience to the occurrence of a single defect.

In Table 8 shows the single missing cell faults (in column 2) that did not generate any error in both outputs sum and carry for all eight input combinations and in column 3 we report the faults that generated at least one output error for at least one input. Fault occurrence percentage is then computed.

We provide the system perspective of both thermal and defect studies however, we believe that relative merits of the various implementation of cross-wires is more meaningful considering them as stand-alone but system analysis would help study various masking effects offered by the layouts.

9 Conclusion

This chapter has analyzed the robustness and thermal performance of different circuits for coplanar crossing in QCA. Resilience to temperature and to missing cell defects has been treated in detail. The use of a Bayesian Network (BN) simulator has allowed for fast and reliable computation of the thermal properties of these circuits. The BN simulator is useful for studying the near-ground state (as related to the error probability) and the thermal characterization of QCA circuits. In this chapter, it has been shown that in all circuits and related configurations for the two directions of signal flow, thermal robustness is not affected by input values; moreover, the use of the so-called thick crossing circuit accounts for the highest resilience to temperature. From the simulation results it has been shown and then proved that

rotated cells are thermally more stable than non-rotated ones. A missing cell defect model has been evaluated for the coplanar crossing to select the circuit with the highest performance for thermal robustness. Simulation has shown that that a thick crossing circuit is very robust also in presence of defects and related logic faults. Finally a simulation on a full adder circuit has proved that the use of thick crossing increases the thermal and defect robustness.

References

1. S. Bhanja, M. Ottavi, S. Pontarelli, and F. Lombardi. Novel designs for thermally robust coplanar crossing in qca. *IEEE Design and Testing in Europe*, pp. 786–791, 2006.
2. S. Bhanja and S. Sarkar. Probabilistic modeling of qca circuits using bayesian networks. *IEEE Transactions on Nanotechnology*, 5(6):657–670, November 2006.
3. A. Fijany, N. Toomarian, and K. Modarress. Block qca fault-tolerant logic gates. Technical report, Jet Propulsion Laboratory, California, 2003.
4. A. Gin, P. D. Tougaw, and S. Williams. An alternative geometry for quantum-dot cellular automata. *J. Appl. Phys.*, 85(12):8281–8286, June 1999.
5. K. Hennessy and C. Lent. Clocking of molecular quantum-dot cellular automata. *Journal of Vacuum Science and Technology*, 19(B):1752–1755, 2001.
6. C. Lent. Molecular quantum-dot cellular automata. *Seminar*, May 2004.
7. C. Lent and P. Tougaw. Lines of interacting quantum-dot cells - a binary wire. *Journal of Applied Physics*, 74:6227–6233, 1993.
8. C. Lent and P. Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85(4):541–557, April 1997.
9. S. K. Lim, R. Ravichandran, and M. Niemier. Partitioning and placement for buildable qca circuits. *J. Emerg. Technol. Comput. Syst.*, 1(1):50–72, 2005.
10. G. Mahler and V. A. Weberruss. *Quantum Networks: Dynamics of Open Nanostructures.* Springer Verlag, 1998.
11. M. Momenzadeh, M. Ottavi, and F. Lombardi. Modeling qca defects at molecular-level in combinational circuits. *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems DFT 2005*, pages 208–216, 2005.
12. P. M. Niemier, M. T. Kontz, M. J. Kogge. A design of and design tools for a novel quantum dot based microprocessor. In *Design Automation Conference*, pages 227–232, June 2000.
13. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference.* Morgan Kaufmann Publishers, 1998.
14. G. Toth. *Correlation and Coherence in Quantum-dot Cellular Automata.* PhD thesis, University of Notre Dame, 2000.
15. P. D. Tougaw and C. S. Lent. Logical devices implemented using quantum cellular automata. *Journal of Applied Physics*, 75(3):1818–1825, Oct 1994.
16. P. D. Tougaw and C. S. Lent. Dynamic behavior of quantum cellular automata. *Journal of Applied Physics*, 80(15):4722–4736, Oct 1996.
17. K. Walus, T. Dysart, G. Jullien, and R. Budiman. QCADesigner: A rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. on Nanotechnology*, 3(1):26–29, 2004.

Chapter 9: Reliability and Defect Tolerance in Metallic Quantum-Dot Cellular Automata

M. Liu and C.S. Lent

1 Introduction

Conventional transistor-based CMOS technology faces great challenges with the down-scaling of device sizes in recent years. Issues such as quantum effects, dopant-induced disorder, and power dissipation may hinder further progress in scaling microelectronics. As the scaling approaches a molecular level, a new paradigm beyond using current switches to encode binary information may be needed. Quantum-dot cellular automata (QCA) [1–3, 5, 11, 12, 15, 18] emerges as one such a paradigm. In the QCA approach bit information is encoded in the charge configuration within a cell. Columbic interaction between cells is sufficient to accomplish the computation; no current flows out of the cell. It has been shown that very low power dissipation is possible [14].

A clocked QCA cell constructed with six quantum dots is shown in Fig. 1. Dots are simply places where a charge is localized. Two mobile electrons are present in the cell. The electrons will occupy antipodal sites in the corner dots because of Coulomb repulsion. The two configuration states correspond to binary information of "1" and "0". The electrons can also be pulled to middle dots if the occupancy energy in the middle dots is lower than corner dots. In this case we term the configuration "null", with no binary information present. The clock adjusts the relative occupancy energy between active dots in the corner and null dots in the middle, pushing electrons to either active dots or null dots. The cell therefore switches between null state and active state. When a cell is placed close to another cell (as shown in Fig. 1b), they will have the same polarization due to Coulomb coupling. Based on the cell-to-cell interaction, logical QCA devices like binary wires, inverters, majority gates and full adders can all be implemented [15].

QCA devices exist. QCA devices made of metal-dot cells have been successfully demonstrated at low temperatures. Majority gates, binary wires, memories, clocked shift registers and fan outs have all been fabricated [1–3,11,12,18]. Figure 2 shows a schematic diagram and scanning electron micrograph of a clocked shift register. Aluminum islands form the dots and Al/AlO_x tunnel

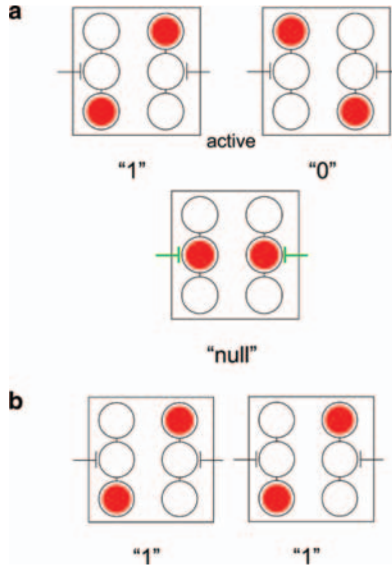


Fig. 1. Schematic of a QCA cell. (**a**) The three states of a single cell. (**b**) Coulomb interactions couple the states of neighboring cells

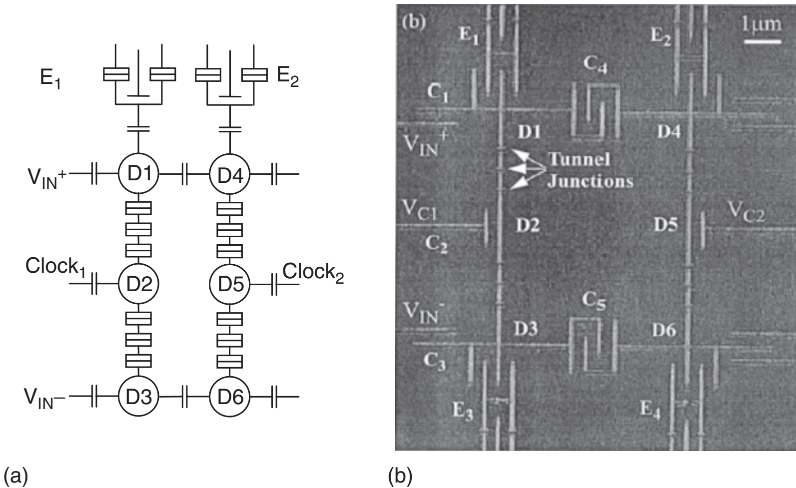


Fig. 2. (**a**) Schematic of a clocked shift register. (**b**) Scanning electron micrograph of a clocked shift register

junctions serve as the tunneling path between dots. Tunnel junctions are fabricated with shadow evaporation technique. Multiple tunnel junctions are used instead of a single junction to suppress co-tunneling. The clock is implemented by simply applying voltage to leads capacitively coupled to the middle dots. Single electron transistors (SET's) are used as readout electrometers.

Though the operation of metal-dot QCA devices is restricted to cryogenic temperatures, they may be viewed as prototypes for molecular QCA cells that will operate at room temperature. It may well be that molecular QCA, with the possibility of enormous functional densities, very low power dissipation, and room temperature operation, is finally the most promising system [4, 7–10, 13]. Metal-dot QCA do have the advantage of having been already created and tested, and we expect that understanding the details of robustness in the metal-dot system will yield benefits for designing molecular systems.

Here we focus on the robustness in metal-dot QCA circuits. In particular, we consider theoretically the effects of temperature, random variations in capacitance, and operating speed, on the performance of a semi-infinite QCA shift register. The chapter is organized as follows: In Sect. 2, we describe the application of single-electron tunneling theory to metal QCA devices. Section 3 describes the characterization of power gain in QCA circuits. In Sect. 4 we analyze the operation of a semi-infinite QCA shift register. Finally, in Sect. 5 we calculate behavior of the QCA shift register in the limits of high speed, high temperature, and high defect levels.

2 Single Electron System Theory

Metal-dot QCA can be described with the orthodox theory of coulomb blockade [16]. The circuit is defined by charge configurations, which are determined by the number of electrons on each of the metal islands. Metal islands are regions of metal surrounded by insulators; at zero temperature they hold an integer number of charges. The islands play the role of QCA dots and are coupled to other islands and leads through tunnel junctions (i.e. quantum-mechanically leaky capacitors) and non-leaky capacitors. Leads by contrast are metal electrodes whose voltages are fixed by external sources. We define dot charge q_i as the charge on island i and q'_k as the charge on lead k. The free energy of charge configuration within the circuit is the electrostatic energy stored in the capacitors and tunnel junctions minus the work done by the voltage sources [17]:

$$F = \frac{1}{2} \begin{bmatrix} q \\ q' \end{bmatrix}^T C^{-1} \begin{bmatrix} q \\ q' \end{bmatrix} - v^T q' \tag{1}$$

Here C is the capacitance matrix including all the junctions and capacitors, v is the column vector of lead voltages, and q and q' are the column vectors of dot charges and lead charges. At zero temperature, the equilibrium charge configuration is the one that has minimum free energy and the number of charges on each islands is exactly an integer. A tunneling event happens at zero temperature only if the free energy is lower for the final state than for the initial state. At finite temperatures, a dot charge need no longer be an integer but is rather a thermal average over all possible configurations. A thermally

excited tunneling event may happen even when the free energy increases. The transition rate of tunneling between two charge configuration states at a certain temperature T is given by

$$\Gamma_{ij} = \frac{1}{e^2 R_T} \frac{\Delta F_{ij}}{1 - e^{-\Delta F_{ij}/(kT)}} \tag{2}$$

where R_T is the tunneling resistance, ΔF_{ij} is the energy difference between the initial state i and final state j.

The tunneling events can be described by a master equation – a conservation law for the temporal change of the probability distribution function of a physical quantity,

$$\frac{dP}{dt} = \Gamma P \tag{3}$$

where P is the vector of state probabilities and Γ is the transition matrix. From the solution P(t) we can obtain the ensemble average of the charge on each dot. We solve (3) directly and find the dot charge as a function of time; from this we can obtain any other voltage or charge in the circuit. In many systems direct solution of the master equation, which requires the enumeration of all the accessible states of the system is impractical due to the large set of accessible states. Because QCA operates so near the instantaneous ground state of the system, complete enumeration of the accessible states is possible and we need not resort to Monte Carlo methods.

3 Power Gain in QCA

A robust circuit must have power gain in order to restore signals weakened due to unavoidable dissipative processes. In conventional CMOS, the power supply provides the energy power gain. In QCA systems the energy needed for power gain is supplied by the clock. A weak input is augmented by work done by the clock to restore logic levels. Power gain has been studied theoretically in molecular QCA circuits [14] and measured experimentally in metal-dot QCA circuits [3]. Power gain is defined by the ratio of the work done *by* the cell on its neighbor to the right (the output of the cell), to the work done *on* the cell by its neighbor to the left (the input to the cell). The work done on a cell by an input lead coupled through an input capacitor C over a time interval T is given by

$$W = \int_0^T V(t) \frac{d}{dt} Q_c(t) dt \tag{4}$$

where V(t) is the lead voltage, $Q_c(t)$ is the charge on the input capacitor. We consider the total work done over a clock period so the cell configuration is the same at t = 0 and t = T. The power gain is thus the ratio of output to input signal power W_{out}/W_{in}, where each sums the work done by (on) all input (output) leads.

4 Operation of Semi-Infinite QCA Shift Register

The schematic of a clocked half QCA cell is shown in Fig. 3a. The capacitances are taken to be $C_j = 1.6aF$, $C_g = 0.32aF$, $C_c = 0.8aF$, and the tunneling resistance $R_T = 100k\,\Omega$. Each island is grounded through a capacitance of 0.32 aF. These are physically reasonable though somewhat better (meaning capacitances are smaller) than the experiments have so-far achieved. Input is applied to the top and bottom dot through coupling capacitors. The potential difference between the top and bottom dots is the output V_{cell}.

The phase diagram of the equilibrium charge state configuration of the cell shown in Fig. 3a is plotted in Fig. 4. The diagram shows the calculated stable regions of charge configuration as a function of input and clock potential. Each hexagonal region is labeled by three integers (n_1, n_2, n_3), the number of elementary charges in the top, middle, and bottom dot respectively. A positive number indicates an extra hole and negative number represents an extra electron. Each hexagon represents the configuration state that has, for those values of input voltage and clock voltage, the lowest free energy.

The clocking cycle can be envisioned as follows. First, a small input bias is applied, when the clock is high (less negative, in fact for this circuit 0). This situation corresponds to point (a) in Fig. 4; no electron switching event happens and the cell remains in the null state, holding no information. When the clock is then lowered (more negative) the system moves along the line

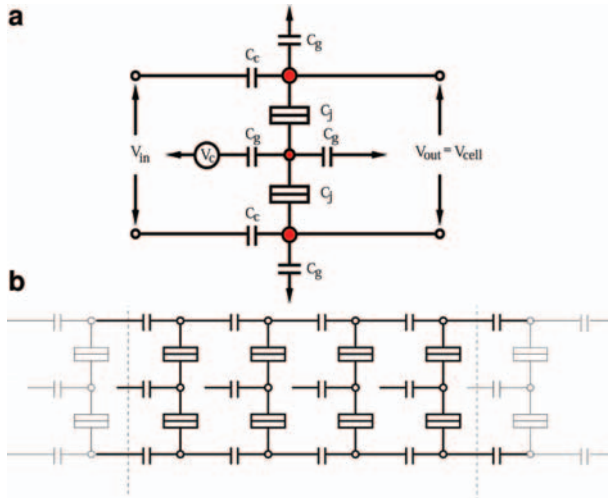


Fig. 3. (**a**) Schematic of a clocked triple dot. The input is applied to the top and bottom dot. The clock is set to the middle dot. The output defined as V_{cell} is the differential potential between the top and the bottom dot. $C_j = 1.6$ aF, $C_g = 0.32$ aF, $C_c = 0.8$ aF. The capacitor to ground is 0.32 aF. $R_T = 100k\Omega$. (**b**) Schematic of a shift register composed of a line of identical triple dots in (**a**). The *thick line* described the actual four cells simulated

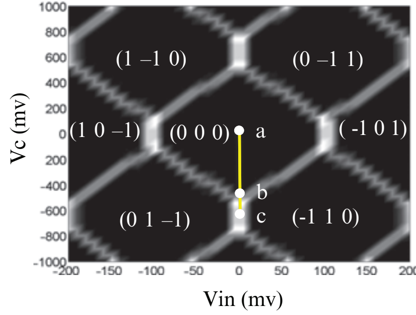


Fig. 4. The equilibrium state configuration of a *triple dot* cell described in Fig. 6. (n1, n2, n3) are the number of charges in the top, middle and bottom dot, respectively. The cell is in the null state in point (a). The cell is in the active state in point (b). The cell is in locked state in point (c)

shown through point (b). An electron is switched to either top dot or bottom dot, decided by the input; the cell is then in the active state. If the clock is held very negative point (c), the electron is locked in the active state, since the energy barrier in the middle dot is too high to overcome. The locked cell is essentially a single bit memory – its present state depends on its state in the recent past, not on the state of neighbors. Varying clock potential gradually between point (a) and (c) will switch the cell between null, active and locked state adiabatically.

A QCA shift register can be constructed with a line of capacitively coupled half QCA cells shown in Fig. 3b, where the output from each cell acts as the input to its right neighbor. The transport of information from cell to cell is controlled by clock signals. Initially, all the cells are in the null state since the clocks are high. Then an input signal is applied to the first cell and the clock for the first cell is lowered. The first cell thus switches to the opposite state of the input and holds to that state even when input is removed. When the clock for the second cell is lowered, the second cell switches to the opposite state to the first cell accordingly and locks the bit. The information is thereafter propagated along the cell line by the clock signals. Each cell in turn copies (an inverts) a bit from its neighbor to the left when the left neighbor is in the locked state and erases the bit, i.e. returns to the null state, while its right neighbor still holds a copy (inverted) of the bit. The copying of the bit can be accomplished gradually so that the switching cell is always close to its instantaneously ground state and thus dissipates very little energy.

It's instructive to model a semi-infinite shift register in order to study the robustness in the QCA circuit. A four phase clocking scheme is adopted to achieve adiabatic switching, shown in Fig. 5. Each clock signal is shifted a quarter-period. As a bit moves down the shift register, we need model only a four QCA half-cells at a time, since by the time the bit is latched in the leading cell, the leftmost cell has returned to null. This is equivalent to viewing the simulation as occurring on a ring of four half-cells. Figure 6 shows the

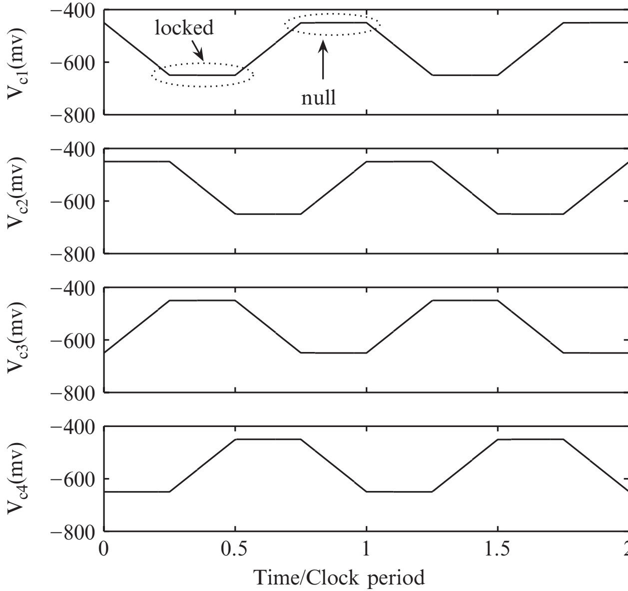


Fig. 5. A four phase clocking scheme in metal-dot QCA

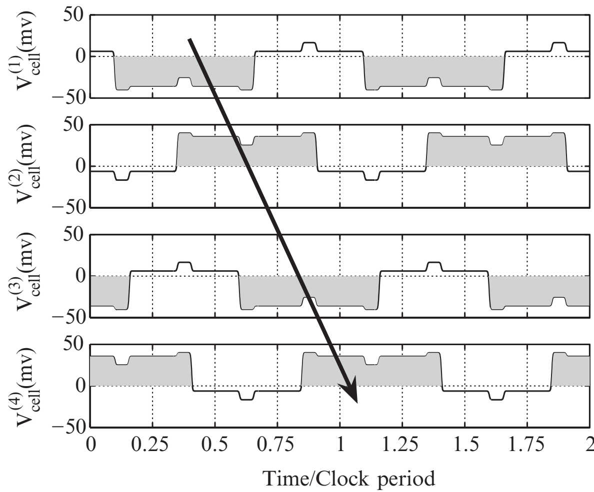


Fig. 6. Time evolution of cell potential in the neighboring cells. $V_{cell}^{(n)}$ is the differential potential between the top and the bottom dot of the nth cell

time evolution of cell potentials for four neighboring cells in a semi-infinite shift register. The shaded areas indicate stored bit information. Each cell has the opposite signal to its neighboring cells with a quarter period shift; the information is both copied and inverted. The arrow indicates the direction of

the information flow. At the end of the first quarter clock period, the first clock is set to low so that the first cell latches the input and locks it while the second cell is in the null state. By the time the second clock is low, the first cell is still kept locked. The second cell thus copies the bit from the first cell. By the end of the third quarter period, the bit in the first cell is erased as its clock is set to high. The third cell copies the bit from the second cell and holds it. The process goes on and the bit information is transported along the chain. Note that there are always at least two copies of the bit at one time. When there are three copies of the bit, the cell potential in the middle cell decreases slightly (in absolute value) while the cell potential in its left and right neighbor increase slightly (thus the small "notch" in the center of the flat parts of the waveform).

5 Robustness in Semi-Infinite QCA Shift Register

5.1 Effect of Temperature and Speed

Because of the difficulty of fabricating small capacitors, metal-dot QCA circuits operate at low temperatures. Thermal excitation is therefore a potential source of random error in metal-dot QCA circuits, and it is clear that at high enough temperatures the circuit will fail. It is tempting to conclude that for a long line of cells, failures are unavoidable at any non-zero temperature. It is well known that there is no long-range order in one-dimensional systems [17]. While the energy for a mistake might be higher than $\mathrm{k_BT}$, the degeneracy (and therefore entropy) of mistake states increases as the system size expands. For a system in thermal equilibrium therefore, the free energy of the mistake states eventually become lower than the mistake-free zero-entropy ground state [6]. A static (unclocked) chain of QCA cells therefore has, for any non-zero temperature, a characteristic length ($\sim e^{E_k/k_BT}$) after which mistakes become very likely. But a clocked line is not in thermal equilibrium – it is actively driven. The clock can supply energy to the system to restore signal states.

To see the effect of temperature on the performance of the clocked semi-infinite shift register, we here solve the time-dependent problem of the clocked shift register using the master-equation (3) approach described in Sect. 2. The calculated cell potential (see Fig. 3) of the $\mathrm{k^{th}}$ cell in the chain at time t is $\mathrm{V_{cell}(k,t)}$. When each cell in the chain in turn latches the bit the cell potential is at its largest magnitude. Figure 7 shows this maximum cell potential $\mathrm{V_{cell}(k) = max(|V_{cell}(k,t)|)}$ as a function of cell number k down the chain. The calculated response is plotted for various values of the temperature. The cell potential is higher at the very beginning of the chain simply because the first cell is driven by an input voltage which is a stronger driver than subsequent cells see; they are driven by other cells. At temperatures above 10 K the cell potential decays with distance as information is transported along the chain. At each stage the signal deteriorates further, and for a long shift register the

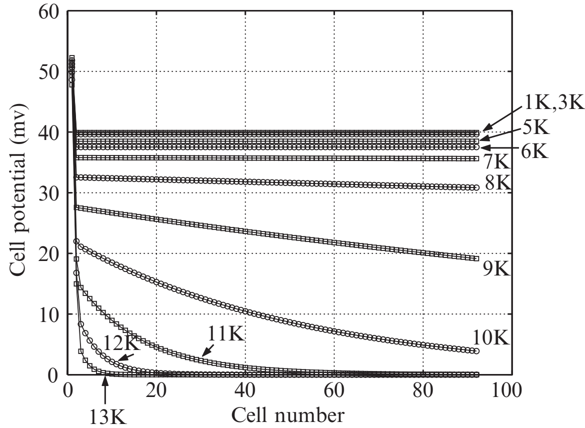


Fig. 7. Cell potential as a function of cell number at different temperatures

information will be lost. For individual cells, this means errors due to thermal fluctuations become increasingly more likely. As the temperature is lowered the signal decay-length increases. At temperatures below 5 K, however, the behavior appears qualitatively different – the cell potential remains constant along the long the chain. To the accuracy of our calculation for a large but finite number of cells, no signal degradation appears at all.

The degradation of performance with increasing temperature can be explained in terms of power gain. We calculate the power gain of each individual cell in the chain by directly calculating the work done on the cell by its neighbor to the left, and the work done by the cell on its neighbor to the right. For each operating temperature the power gain is the same for each cell (apart from those very near the beginning of the line). If the power gain is precisely 1 (or greater), then there is no signal degradation moving down the line. At each cell, power is drawn from the clock sufficient to completely restore the signal as it is copied to the next cell. We refer to the situation in which unity power gain enables transmission of signals over arbitrarily long distances as "robust" operation. If the power gain is less than 1, then the signal will be degraded as it moves down the line. Figure 8 shows the deviation from unity power gain as a function of temperature on a logarithmic scale. For temperatures below 5 K the power gain is 1; above 5 K, the power gain is less than 1. At higher temperatures, the flow of energy from the clock can no longer compensate for the energy loss to the thermal environment, with the result that the signal decays at each stage. As the difference between the power gain and 1 becomes small our analysis is limited by the numerical accuracy of the calculation.

Nevertheless, the exponential character of the approach to unity power gain supports the interpretation that this transition is a qualitative change between robust and non-robust behavior, analogous to a phase transition.

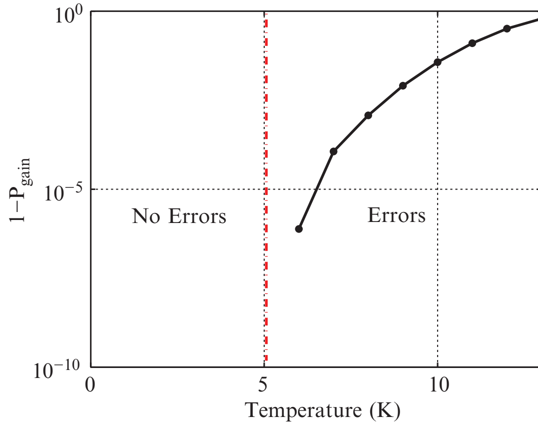


Fig. 8. Deviation from unity power gain for an individual cell as a function of temperature

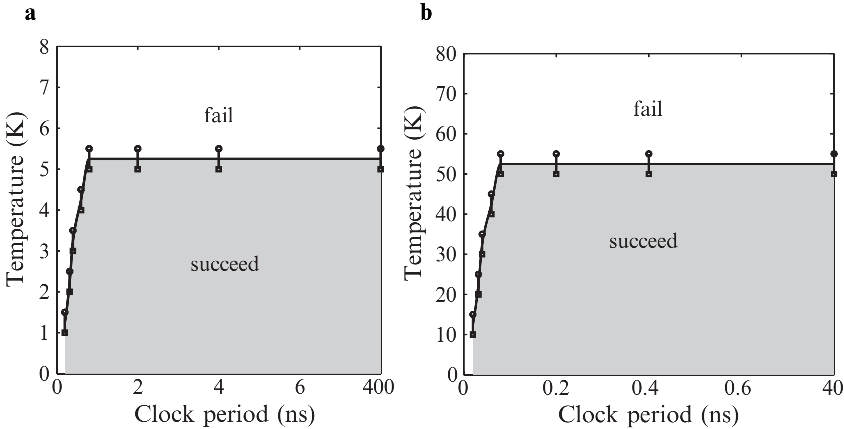


Fig. 9. The phase diagram of the operation space as a function of temperature and clock period when (**a**) Cj = 1.6 aF, and (**b**) Cj = 0.16 aF. The *shaded area* below the curve is where the circuit succeeds and the *white area* is where the circuit fails

The time-dependent calculation above is repeated for various temperatures and clock speeds to generate the phase diagram of the operational space of the circuit shown in Fig. 9. We display the results for the circuit with our standard parameters, with $C_j = 1.6\,aF$ in Fig. 9a and for more aggressively scaled parameters, with $C_j = 0.16\,aF$ in Fig. 9b. All capacitances and voltages in the circuit are scaled appropriately with C_j. The aggressively scaled parameter calculation illustrates scalability of QCA circuits. The performance of the circuit will increase with smaller capacitances. The shaded area below the curve indicates speeds and temperatures for which the circuit is robust. The white area represents non-robust operation for which bit information decays

along the chain. The two figures are identical except for the scale: the aggressively scaled circuit of Fig. 9b operates ten times faster and at a temperature ten times higher than the circuit in Fig. 9a.

The area of robust operation is limited by both speed and temperature. In Fig. 9a, when the clock period is less than about 0.2 ns (corresponding to 5 GHz), the circuit fails (is not robust) even at zero temperature. This occurs as the clock period approaches the electron tunneling rate. When the clock speed is too fast, the electrons do not have enough time to tunnel reliably from one dot to another. The error probability accumulates as the information moves along the chain. Increasing the clock period increases the probability of electrons being in the "right" states. This improvement quickly saturates and further increasing the clock period has no effect since the electrons have had enough time to be in the correct state. The tunneling rate is related to the tunnel resistance, so this description is equivalent to the observation that the speed is limited to the RC time-constant of the circuit.

5.2 Defect Tolerance in the QCA Shift Register

A robust circuit must be tolerant of defects that introduce variations in the values of the designed parameters. We consider the situation of a very long shift register in which the value of each capacitor in the circuit is varied randomly within a fixed percentage range from its nominal value. The circuit is robust if the perturbation of the capacitances does not influence the performance of the circuit. We choose a working point in Fig. 9a where clock period is 5 ns, the temperature is 4K, and vary all the capacitances randomly by $\pm 10\%$ and $\pm 15\%$. Figure 10 shows the cell potential as a function of cell number with random capacitance variation. Different color represents different capacitance variation within the certain percentage range. When the deviation is $\pm 10\%$, the circuit is robust and transmits bit information with no errors. The bit information is carried on correctly even at the 2000th cell. When the deviation

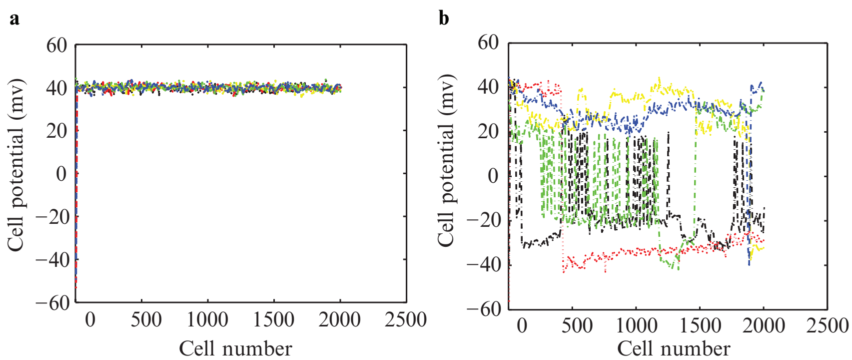


Fig. 10. Cell potential as a function of cell number at 4 K when capacitance variation is **a** $\pm 10\%$ **b** $\pm 15\%$

increases to ±15%, the circuit is fragile since cells are flipped to the wrong states during propagation. This calculation demonstrates, again as a result of the power gain in each cell, that QCA circuits can tolerate considerable variation in parameter values and still function correctly.

6 Conclusion

The QCA approach represents an entirely new way of encoding, moving, and processing binary information. As more experimental realizations of devices appear, attention naturally turns to the broader circuit behavior of these new devices. While molecular QCA may represent the most realistic long-term system for robust room temperature operation, the metal-dot QCA system provides an extremely valuable prototype system in which to explore QCA properties. Metal dot systems also have the advantage of being realizable now.

We have explored here the behavior of metal-dot QCA systems under stress – stressed by high temperature operation, high speed operation, and random variation in parameter values. In each case enough stress destroys the correct operation of the circuit. What we observe however is that these systems are not terribly fragile, they can survive in a broad range of operational space. In each case small errors threaten to accumulate over many cells and result in signal loss. The key feature is power gain from the clocking circuit which provides considerable robustness against these error mechanisms, restoring signal levels at each stage.

References

1. I. Amlani, A. Orlov, G. Toth, G. H. Bernstein, C. S. Lent, G. L. Snider, "Digital logic gate using quantum-dot cellular automata," Science 284: 289–291, 1999.
2. R. K. Kummamuru, J. Timler, G. Toth, C. S. Lent, R. Ramasubramaniam, A. O. Orlov, G. H. Bernstein, G. L. Snider, "Power gain in a quantum-dot cellular automata latch," Applied Physics Letters 81: 1332–1334, 2002.
3. R. K. Kummamuru, A. O. Orlov, C. S. Lent, G. H. Bernstein, G. L. Snider, "Operation of a quantum-dot cellular automata (QCA) shift register and analysis of errors," IEEE Transactions on Electron Devices 50: 1906–1913, 2003.
4. C. S. Lent, B. Isaksen, "Clocked molecular quantum-dot cellular automata," IEEE Transctions on Electron Devices 50: 1890–1896, 2003.
5. C. S. Lent, P. D. Tougaw, W. Porod, G. H. Bernstein, "Quantum cellular automata," Nanotechnology 4: 49–57, 1993.
6. C. S. Lent, P. D. Tougaw, W. Porod, "Quantum cellular automata: the physics of computing with quantum dot molecules," PhysComp'94, Proceedings of the Workshop on Physics and Computing, IEEE Computer Society Press, pp. 5–13, 1994.
7. C. S. Lent, B. Isaksen, M. Lieberman, "Molecular quantum-dot cellular automata," Journal of American Chemical Society 125: 1056–1063, 2003.

8. A. Li, T. P. Fehlner, “Molecular QCA Cells. 2. Characterization of an unsymmetrical dinuclear mixed-valence complex bound to a Au surface by an organic linker,” Inorganic Chemistry 42: 5715–5721, 2003.
9. Z. Li, A. M. Beatty, T. P. Fehlner, “Molecular QCA Cells. 1. Structure and functionalization of an unsymmetrical dinuclear mixed-valence complex for surface binding,” Inorganic Chemistry 42: 5707–5714, 2003.
10. M. Lieberman, S. Chellamma, B. Varughese, Y. L. Wang, C. S. Lent, G. H. Bernstein, G. L. Snide, F. C. Peiris, “Quantum-dot cellular automata at a molecular scale,” Annals of the New York Academy of Sciences 960: 225–239, 2002.
11. A. O. Orlov, I. Amlani, G. H. Bernstein, C. S. Lent, G. L. Snider, “Realization of a functional cell for quantum-dot cellular automata,” Science 277: 928–930, 1997.
12. A. O. Orlov, I. Amlani, R. K. Kummamuru, R. Ramasurbramaniam, G. Toth, C. S. Lent, G. H. Bernstein, G. L. Snider, “Experimental demonstration of clocked single-electron switching in quantum-dot cellular automata,” Applied Physics Letters 77: 295–297, 2000.
13. H. Qi, S. Sharma, Z. Li, G. L. Snider, A. O. Orlov, C. S. Lent, T. P. Fehlner, “Molecular quantum cellular automata cells. Electric field driven switching of a silicon surface bound array of vertically oriented two-dot molecular quantum cellular automata,” Journal of American Chemical Socety 125: 15250–15259, 2003.
14. J. Timler, C. S. Lent, “Power gain and dissipation in quantum-dot cellular automata,” Journal of Applied Physics 91: 823–831, 2002.
15. P. D. Tougaw, C. S. Lent, “Logical devices implemented using quantum cellular automata,” Journal of Applied Physics 75: 1818–1825, 1994.
16. C. Wasshuber, Computational single-electronics, Springer, Berlin Heidelberg New York, 2001.
17. C. Wasshuber, H. Kosina, S. Selberherr, “SIMON: A simulator for single-electron tunnel devices and circuits,” 16: 937–944, 1997.
18. K. K. Yadavalli, A. O. Orlov, R. K. Kummamuru, C. S. Lent, G. H. Bernstein, G. L. Snider, “Fanout in quantum dot cellular automata,” 63rd Device Research Conference 1: 121–122, 2005.

Section 3: Testing Microfluidic Biochips

M. Tehranipoor

Recent years have seen the emergence of droplet-based microfluidic systems for safety-critical biomedical applications. Microfluidics-based biochips are soon expected to revolutionize biosensing, clinical diagnostics and drug discovery. Digital microfluidics is an alternative technology for lab-on-a-chip systems based upon micromanipulation of discrete droplets. Microfluidic processing is performed on unit-sized packets of fluid which are transported, stored, mixed, reacted, or analyzed in a discrete manner using a standard set of basic instructions. This section contains two chapters dealing with issues of test planning and diagnosing realistic defects in Digital Microfluidic Biochips.

Authors in Chap. 10, entitled "Test Planning and Test Resource Optimization for Droplet-Based Microfluidic Systems", investigate test planning and test resource optimization for droplet-based microfluidic arrays. The authors first formulate the test planning problem and prove that it is NP-hard. They then describe an optimization method based on integer linear programming (ILP) that yields optimal solutions. Due to the NP-hard nature of the problem, the authors develop heuristic approaches for optimization. Experimental results indicate that for large array sizes, the heuristic methods yield solutions that are close to provable lower bounds. These heuristics ensure scalability and low computation cost.

The second chapter in this section (Chap. 11) "Testing and Diagnosis of Realistic Defects in Digital Microfluidic Biochips" argues that robust off-line and on-line test techniques are required to ensure system dependability as these biochips are deployed for safety-critical applications. Due to the underlying mixed-technology and mixed-energy domains, biochips exhibit unique failure mechanisms and defects. The authors first relate some realistic defects to fault models and observable errors. They next set up an experiment to evaluate the manifestations of electrode-short faults. Motivated by the experimental results, the authors present a testing and diagnosis methodology to detect catastrophic faults and locate faulty regions. The proposed method is evaluated using a biochip performing real-life multiplexed bioassays.

Chapter 10: Test Planning and Test Resource Optimization for Droplet-Based Microfluidic Systems

F. Su, S. Ozev, and K. Chakrabarty

1 Introduction

Next-generation system-on-chip designs are expected to be composite microsystems with microelectromechanical and microfluidic components [15,23]. These mixed-signal and mixed-technology systems monolithically integrate microelectronics with microsensors and microactuators, thereby leading to chips that cannot only compute and communicate, but also sense and actuate. This high level of integration is enabling a new class of microsystems targeted at health care, environmental monitoring, biomedical analysis, harmful agent detection for countering bio-terrorism, and precision fluid dispensing [13].

In recent years, novel droplet-based microfluidic systems have been developed to analyze nanoliter volumes of agents [18]. These systems reduce the rate of reagent consumption, thereby enabling continuous sampling and analysis for on-line, real-time biological/chemical analysis. By scaling down the concentration of the samples, simple sensing techniques can be utilized to replace conventional, costly, and time-consuming practices involving batch analysis, sample pre-treatment and frequent calibration. Droplet-based microfluidic systems therefore offer a promising platform for massively parallel DNA analysis, and real-time molecular detection and recognition.

As microfluidic systems become widespread in safety-critical biomedical applications, system reliability emerges as an essential performance parameter. In order to ensure reliability, composite microsystems incorporating microfluidic components must be tested adequately. Therefore, there is a pressing need for efficient test methodologies for these microsystems. The ITRS 2003 document recognizes the need for new test methods for disruptive device technologies that underly microelectromechanical systems and sensors, and highlights it as one of the five difficult test challenges beyond 2009 [27].

Recently, a fault classification and a unified test methodology for droplet-based microfluidic systems has been developed [22]. Faults are classified as either catastrophic or parametric, and they are detected by electrostatically controlling and tracking droplet motion. This cost-effective test methodology

facilitates concurrent testing, which allows fault testing and biomedical assays to run simultaneously on a microfluidic system. Test planning and test resource optimization are motivated by the need for concurrent testing.

In this chapter, we investigate test planning and test resource optimization problems for droplet-based microfluidic arrays. We first formulate the test planning problem and prove that it is NP-hard. We then show how optimal solutions can be obtained using integer linear programming (ILP). Due to the NP-hard nature of the problem, the ILP model is not applicable to large microfluidic arrays. We therefore develop heuristics to solve this problem in a computationally efficient manner. Experiments show that for large array sizes, the results obtained from the heuristic method are close to provable lower bounds.

2 Background: Droplet-Based Microfluidic Systems

The operation of droplet-based microfluidic systems is based on the principle of electrowetting actuation. By varying the electrical potential along a linear array of electrodes, electrowetting can be used to move liquid droplets of nanoliter volume along this line of electrodes [18]. Droplets can also be transported, in user-defined patterns under clocked-voltage control, over a two-dimensional array of electrodes without the need for pumps and valves.

The basic component of a droplet-based microfluidic system is shown in Fig. 1. The droplet, usually containing biomedical samples, and the filler medium, such as silicone oil, are sandwiched between two parallel glass plates. The bottom plate contains a patterned array of individually controllable electrodes, while the top plate is coated with a ground electrode. The hydrophobic dielectric insulator is added to the top and bottom plates to decrease the wettability of the surface and to add capacitance between the droplet and the control electrode.

The basic principle underlying droplet transportation is the electrostatic control of the interfacial tension at the droplet–insulator interface. A control

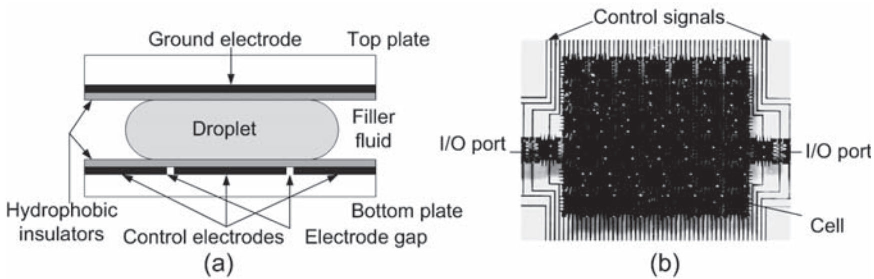


Fig. 1. (**a**) Basic cell used in a droplet-based microfluidic system; (**b**) a two-dimensional array for digital microfluidics

(actuating) voltage is applied to an electrode adjacent to the droplet and, at the same time, the electrode just under the droplet is deactivated. This causes an accumulation of charge in the droplet–insulator interface, resulting in a surface tension gradient across the gap between the adjacent electrodes, which consequently causes the transportation of the droplet. The velocity of the droplet can be controlled by adjusting the actuation voltage (0–90 V), and droplets can be moved at speeds of up to $20\,\mathrm{cm\,s^{-1}}$. Based on this principle, microfluidic droplets can be moved freely to any location of a two-dimensional array; see Fig. 2. This design, which has been fabricated on PCBs at Duke University [18], is ideally suited for a large-scale integrated microfluidic system. Such a system is expected to be common in the near future for various biomedical applications, such as DNA sequencing and bimolecular detection. A droplet can be easily detected using the capacitive sensing circuit shown in Fig. 3.

Using a two-dimensional microfluidic array, many common operations for different biomedical assays can be performed, such as sample introduction (*dispense*), sample movement (*transport*), temporarily sample preservation (*store*), and mixing of different samples (*mix*). Note that these operations can be performed anywhere on the array, whereas in continuous-flow systems they must operate in a specific micromixer or microchamber. The configurations of the microfluidic array, i.e., the routes that sample droplet travel and

Fig. 2. Droplet transport in a two-dimensional array. (detailed video available at http://www.ee.duke.edu/Research/microfluidics)

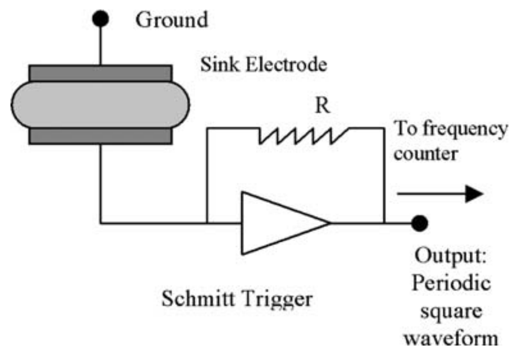


Fig. 3. Simple capacitive sensing circuit

the rendezvous points of droplets, can be obtained using software running on a PC or an ASIC [20, 21]; they are then programmed into a microcontroller that controls the voltages of the electrodes in the array. Test planning for a microfluidic array can also be implemented using a PC or an ASIC.

3 Related Prior Work

Over the past decade, the focus in testing research has broadened from logic and memory test to include the testing of analog and mixed-signal circuits. MEMS is a relatively young field compared to IC design, and MEMS testing is still in its infancy. Recently, fault modeling and fault simulation in surface micromachined MEMS have received attention [4, 11, 14]. Researchers at Carnegie Mellon University are developing a comprehensive testing methodology for a class of MEMS known as surface micromachined sensors.

However, test techniques for MEMS cannot be directly applied to microfluidic systems, since the techniques and tools currently in use for MEMS testing do not handle fluids. Hence they are of limited use for testing microfluidic devices. Most recent work in this area has been limited to the testing of continuous-flow microfluidic systems [**?**]. Researchers at the MESA+ Research Institute of the University of Twente have applied mixed-signal testing techniques to the problem of testing a microanalysis system. Also, a design-for-testability (DFT) technique for Flow-FET-based microfluidic systems has been proposed [9]. Similar to the MOSFET, a Flow-FET has source and drain electrodes over which a relatively large voltage ($\sim$100 V) is applied. Due to the principle of electro-osmotic flow, the electric field moves the charge accumulated between the fluid and the surface of channel, dragging the bulk liquid through the channel.

Optimal strategies for moving droplets in a microfluidic system are proposed in [3]. The A^* algorithm from artificial intelligence is used as the basis of a systematic search, which is performed to generate a sequence of control signals for moving one or multiple droplets from the start to the goal positions in the shortest number of steps. This method is closely related to the optimization problem of motion planning with multiple moving robots [1, 12]. There are two different groups of path planning problems for moving robots. Navigation problem attempts to find a path from a start position to a goal position through the shortest path, whereas coverage problem focuses on finding the path of coverage of an environment by mobile robots.

4 Problem Definition

In the test methodology proposed in [22], test stimuli droplets are dispensed into the microfluidic system from the droplet source and transported through the array (traversing the cells) by following the designed testing scheme.

As described in [22], most catastrophic faults in droplet-based microfluidic systems can lead to a complete cessation of droplet transportation. Thus, for the faulty case, the test stimuli droplet is stuck at an intermediate point during motion. On the other hand, the detection of all test stimuli droplets at the droplet sinks indicates fault-free operation. This methodology allows fault testing and biomedical assays to run concurrently on a microfluidic system. An efficient test plan not only ensures that the testing operation does not conflict with the normal biomedical assay, but it also guides test stimuli droplets to cover all the cells available for testing. This test plan can be optimized to minimize the total testing time cost for a given test hardware overhead, which refers here to the number of droplet sources and droplet sinks. Note that some faults such as electrode shorts affect two adjacent electrodes [19,22]. To detect such faults, defect-oriented test procedures are required, which focus on pairs of cells and the traversal of droplets from one cell to all its neighbors [19]. For simplicity, we do not take into account such types of faults in this chapter; only catastrophic faults related to a single cell are targeted.

We can formulate the test planning problem in terms of graph partitioning and the Hamiltonian path problem from graph theory [5]. The key idea underlying this optimization approach is to model the two-dimensional microfluidic array as a directed graph, and then partition it into non-overlapping subgraphs. Each part of the microfluidic array is represented by a subgraph that is tested concurrently and independent of the other parts. In this way, the total test application time is reduced.

First we model the array of microfluidic cells using a directed graph $G = (V, E)$ where the set of vertices V represents the set of available microfluidic cells, droplet sources and droplet sinks, and $e_{ij} \in E$ is a directed edge from vertex i to vertex j if and only if these two vertices represent two adjacent microfluidic cells and they satisfy the criterion described below.

Note that unlike V, E is not determined a priori; rather the set of edges is a variable, and the edges are determined through the optimization procedure.

Definition 1. *A Hamiltonian path from vertex s to vertex t in a graph G is a path that starts at vertex s, ends at vertex t, and visits every vertex of G exactly once.*

We define e_{ij} as follows:

$$e_{ij} = \begin{cases} 1 & \text{if a Hamiltonian path from a droplet source to a droplet sink} \\ & \text{includes vertex } i \text{ and vertex } j \text{ in consecutive order} \\ 0 & \text{otherwise.} \end{cases}$$

If a Hamiltonian path exists in an array with n cells, then for any cell i in the array, $\sum_{j=1}^{n} e_{ij} = \sum_{j=1}^{n} e_{ji} = 1$.

The problem of finding a Hamiltonian path in graph G from one source to one sink can be expressed as the following problem: find a numerical instance

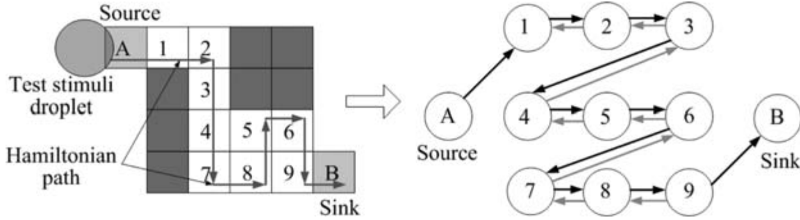


Fig. 4. Graph model for a 4×4 microfluidic array

of the set of binary variables $E = \{e_{ij}\}$, e.g., $\{e_{12} = 1,\ e_{21} = 0, \ldots,\ e_{ij} = 1, \ldots\}$, that represents a Hamiltonian path from one source to one sink.

If a Hamiltonian path exists, the cost C for this path is defined as $C = \sum_{i=1}^{n} \sum_{j=1}^{n} e_{ij} w_{ij}$, where i represents any vertex in this path, j is the vertex adjacent to i in the path, and w_{ij} is the weight of e_{ij}. Without loss of generality, we set w_{ij} to be a constant value, assuming that the transportation velocity between any two adjacent microfluidic cells is the same. For simplicity, let $w_{ij} = 1$. Therefore, $C = \sum_{i=1}^{n} \sum_{j=1}^{n} e_{ij} = \sum_{i=1}^{n} 1 = n$, i.e., the number of vertices on the Hamiltonian path. If G has no Hamiltonian path, the cost C is infinite.

Figure 4 gives an example of a graph model for single source and single sink. In the graph model of this 4×4 array, a black arrow between vertices i and j denotes that $e_{ij} = 1$, while the gray arrow between vertices i and j denotes that $e_{ij} = 0$. The cost C for this example is 11.

Based on the above definitions, we now develop the test planning problem for multiple sources and multiple sinks. We attempt to partition the directed graph representing the microfluidic array into subgraphs, such that in each subgraph there exists a Hamiltonian path from one source to one sink. In this way, the testing of the different partitions can be performed independently and simultaneously in non-overlapping parts of the microfluidic array. The total cost for the array is the maximum of the cost for any of these subgraphs. This leads us to the following optimization problem for minimizing the total cost:

– *Optimal partitioning problem (OPP).* Given N source/sink pairs, determine an optimal partition that divides the available cells in the array into N non-overlapping partitions, such that in each partition there exists a Hamiltonian Path from one source to one sink and the maximum of the cost for these Hamiltonian paths is minimized.

5 Analysis of Computational Complexity

In this section we prove that OPP is NP-hard. We first review the following definition from computational complexity theory:

Definition 2. *[17]: Let L1 and L2 be two decision problems. L1 is polynomial-time reducible to L2* ($L1 \leqslant L2$) *if a polynomial-time reduction f from L1 to L2 exists, subject to*

- *f(x) is a yes-input for L2 if and only if x is a yes-input for L1*
- *f is computable in polynomial-time.*

We next note that if L1 is NP-complete, and $L1 \leqslant L2$, then L2 is NP-hard. This is a common technique to prove that a given optimization problem is NP-hard.

We first consider the decision version D-PP of OPP, which is expressed as follows.

- D-PP: *Given N source/sink pairs and an upper limit D on the cost, is it possible to partition array into N parts such that there exists a Hamiltonian path of cost* C_i *for each partition and* $\max\limits_{1 \leqslant i \leqslant N} \{C_i\} < D$?

Theorem 1. *OPP is NP-hard.*

Proof. We first show that D-PP $\in$ NP. We can non-deterministically generate a N-partition and then verify in polynomial time that $\max\limits_{1 \leqslant i \leqslant N} \{C_i\} < D$. To show that D-PP is NP-hard, we reduce the problem of determining a Hamiltonian cycle in grid graph (HC-GG), which is known to be NP-complete [7]. A grid graph G is a finite, induced subgraph of the infinite two-dimensional grid. It has a finite set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, where v_i represents a grid point (x, y). Note that x and y are positive integers, denoting the x and y coordinates, respectively. An edge exists in G between point (x, y) and (x', y') if and only if $|x - x'| + |y - y'| = 1$.

We next define a polynomial-time reduction f from an arbitrarily-chosen instance of HC-GG to an instance of D-PP with $N = 1$ and $D = \infty$. Given a grid graph G, any vertex v_i in G is mapped to a cell c_i in array A, such that cell $c_i = f(v_i)$ and $c_j = f(v_j)$ are adjacent in A if and only if there exists an edge between v_i and v_j in G. We define the vertices with the maximum (or minimum) value x of the x-coordinate (or the y-coordinate y) in the corresponding grid graph to be boundary vertices in G. Similarly, the cells in the array obtained by mapping from the boundary vertices in G are defined as boundary cells in A. Next we attempt to add a droplet source s_1 and a droplet sink s_2 to this array. There are two possible cases. In Case 1, there exist two adjacent boundary vertices (noted as v_1 and v_n) in G, such that there also exist two adjacent cells (noted as c_1 and c_n) on the boundary of array A. We then add $s_1$next to c_1 and s_2 next to c_n; see Fig. 5a. In Case 2, if there are no adjacent boundary vertices in G and neither are there adjacent boundary cells in A, we select a single boundary cell denoted by c_1, and place s_1 and s_2 together adjacent to c_1; see Fig. 5b. It is obvious that the transformation described above can be carried out in polynomial time.

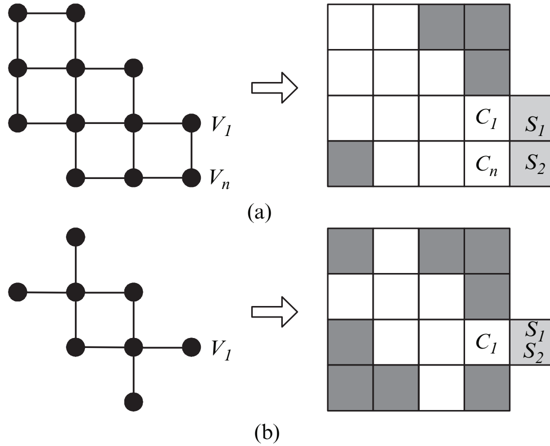


Fig. 5. (**a**) Illustration of Case 1; (**b**) illustration of Case 2

Next we prove that there exists a Hamiltonian path from s_1 to s_2 of cost $C < \infty$ in A if and only if there exists a Hamiltonian cycle in G of cost less than ∞.

(1) *Proof for Case 1.* Assume there exists a Hamiltonian cycle in G, denoted by $v_1 v_2 \dots v_n v_1$, where v_1 and v_n are two adjacent boundary vertices. Due to the mapping $f: G \to A$, $c_1 = f(v_1)$, $c_n = f(v_n)$ and they are two adjacent cells on the boundary of array A. In this way, there exists a path $f(v_1)f(v_2)\dots f(v_n)$ from c_1 to c_n that visits every cell exactly once. In addition, s_1 is adjacent to c_1 and s_2 adjacent to c_n.
Therefore, there is a Hamiltonian path from s_1 to s_2 in A and cost $C = n < \infty$. On the other hand, if there exits a Hamiltonian path $s_1 c_1 \dots c_n s_2$ from s_1 to s_2 in array A, a Hamiltonian path from c_1 to c_n also exists. Now by the inverse transformation $f^{-1}: A \to G$, it is seen that there exists a Hamiltonian path $f^{-1}(c_1)\dots f^{-1}(c_n)$ from $f^{-1}(c_1)$ to $f^{-1}(c_\mathrm{n})$. Moreover, $f^{-1}(c_1)$ and $f^{-1}(c_\mathrm{n})$ are two adjacent vertices. Therefore, there exists a Hamiltonian cycle $f^{-1}(c_1)\dots f^{-1}(c_\mathrm{n})f^{-1}(c_1)$ in G.

(2) *Proof for Case 2.* If there exist no adjacent cells on the boundary of A, we place s_1 and s_2 together next to one boundary cell c_1. This implies that in any path from s_1 to s_2, $c_1$is visited at least twice. Therefore, there exists no Hamiltonian path in A for this case and $C = \infty$. Similarly in G, since there are no adjacent vertices on the boundary, some boundary vertices have only degree one. This violates the necessary condition for the existence of a Hamiltonian cycle, i.e., every node should have a degree of at least two. Hence there is also no Hamiltonian cycle in G.

Thus we have shown that any instance of HC-GG is polynomial-time reducible to an instance of D-PP ($N = 1$ and $D = \infty$). Since HC-GG is NP-complete, D-PP is at least NP-hard. Moreover, since D-PP is in NP, it is

also NP-complete. The optimization version of D-PP, i.e., the Optimal Partitioning Problem is therefore NP-hard.

6 Integer Linear Programming Model for OPP

Although OPP has been proven in Sect. 5 to be NP-hard, we show in this section that it can be solved exactly using integer linear programming (ILP) for a microfluidic array of modest size. An ILP model can be described as follows:

$$\begin{aligned} &\text{Minimize}: \ Ax \quad \text{(objective function)} \\ &\text{Subject to}: \ Bx \leqslant C \ \text{(constraint inequalities)}, \end{aligned}$$

where x is a vector of variables, A is an objective function vector, B is a constraint matrix and C is a column vector of constraints. We used a popular public domain ILP solver called *lpsolve* for our work [2].

We formulate the ILP model for OPP as follows. It is obvious that when $N = 1$, OPP is equivalent to the Hamiltonian path problem for a single source and a single sink described in the earlier section.

For $N > 1$, we define a binary variable S_{ik} as follows:

$$S_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ is in subgraph } k \text{ i.e., microfluidic cell } i \text{ belongs to} \\ & \textit{partition } k. \\ 0 & \text{otherwise} \end{cases}$$

where $1 \leqslant k \leqslant N$. Since every vertex only belongs to one subgraph, $\sum_{k=1}^{N} S_{ik} = 1 \forall i$.

Definition 3. *Vertex j is the connected neighbor of vertex i, if there is an edge between i and j, and either $e_{ij} = 1$ or $e_{ji} = 1$.*

Next we impose the constraint that vertex i is in partition k if and only if its connected neighbor is also in partition k. This is expressed as follows:

$$S_{ik} = 1 \text{ if and only if} \sum_{j=1}^{n} e_{ij} S_{jk} = 1 \Rightarrow S_{ik} = \sum_{j=1}^{n} e_{ij} S_{jk}.$$

The existence of Hamiltonian paths in non-overlapping partitions ensures that, for every cell i in array, $\sum_{j=1}^{n} e_{ij} = \sum_{j=1}^{n} e_{ji} = 1$.

Finally, we incorporate the objective function into the ILP model. The objective of this optimization problem is to minimize the total cost $C = \max_k \{C_k\} = \max_k \{n_k\}, \ k = 1, 2 \ldots N$, where n_k is the number of vertices

visited by Hamiltonian path k. It is easily seen that $n_k = \sum_{i=1}^{n} S_{ik}$. Therefore, $C = \max_{1 \leqslant k \leqslant N} \sum_{i=1}^{n} S_{ik}$.

We now have a mathematical programming model for OPP, described as follows.

$$\text{Objective: minimize} \quad C = \max_{1 \leqslant k \leqslant N} \sum_{i=1}^{n} S_{ik}$$

$$\text{Subject to:} \quad (1) \ \sum_{k=1}^{N} S_{ik} = 1 \forall i.$$

$$(2) \ S_{ik} = \sum_{j=1}^{n} e_{ij} S_{jk} \forall i, \ 1 \leqslant k \leqslant N.$$

$$(3) \ \sum_{j=1}^{n} e_{ij} = \sum_{j=1}^{n} e_{ji} = 1 \forall i.$$

In order to solve the above mathematical programming model using *lp-solve*, its objective function and some constraint inequalities must be linearized to match the canonical form of an ILP model. First, the objective function is linearized as: Minimize C, subject to C $\geqslant \sum_{i=1}^{n} S_{ik}, \ 1 \leqslant k \leqslant N$. The set of constraints in (2) above contains the non-linear term $e_{ij} S_{jk}$, which can be linearized by introducing a binary variable $Z_{ijk} = e_{ij} S_{jk}$, with the following additional constraints [25]:

(1) $e_{ij} + S_{jk} - Z_{ijk} \leqslant 1$.
(2) $e_{ij} + S_{jk} - 2Z_{ijk} \geqslant 0$.

This transformation is verified as follows: If $S_{jk} = 0$, from (1) and (2), $Z_{ijk} + 1 \geqslant e_{ij}$ and $2Z_{ijk} \leqslant e_{ij}$; since $e_{ij} \leqslant 1, \ Z_{ijk} = 0$. If $S_{jk} = 1$, we get $Z_{ijk} \geqslant e_{ij}$ and $2Z_{ijk} \leqslant e_{ij} + 1$. Therefore, $Z_{ijk} = e_{ij}$.

We now describe the ILP model for OPP, which includes the new variable and constraints.

Objective: Minimize C

$$\text{Subject to:} \quad (1) \ C \geqslant \sum_{i=1}^{n} S_{ik} 1 \leqslant k \leqslant N.$$

$$(2) \ \sum_{k=1}^{N} S_{ik} = 1 \forall i.$$

$$(3) \ S_{ik} = \sum_{j=1}^{n} Z_{ijk} \quad \forall i 1 \leqslant k \leqslant N.$$

$$(4) \ e_{ij} + S_{jk} - Z_{ijk} \leqslant 1 \quad \forall i, \ \forall j, \quad 1 \leqslant k \leqslant N.$$

$$(5)\ e_{ij} + S_{jk} - 2Z_{ijk} \geqslant 0 \quad \forall i, \forall j, \quad 1 \leqslant k \leqslant N.$$

$$(6)\ \sum_{j=1}^{n} e_{ij} = \sum_{j=1}^{n} e_{ji} = 1 \quad \forall i.$$

The above ILP model can now be solved using *lpsolve*. The complexity of this model, measured by the number of variables and the number of constraints, is $O(n^2 \times N)$, where n is the number of cells in an array and N is the number of source/sink pairs.

The following example illustrates an optimal partitioning of a 4×4 microfluidic array with two sources a1 and a2, and two sinks b1 and b2, respectively; see Fig. 6. The result is obtained using *lpsolve*. It took 10 min of CPU time on a 1.6 GHz Pentium-IV PC with 392 MB of RAM. An optimal partitioning generated by *lpsolve* is as follows: Partition 1 = {1–3, 5, 7, 9, a1, b1}, Partition 2 = {4, 6, 8, 10–12, a2, b2}; see Fig. 7. Based on this test plan, the total time cost C is max {8, 8} = 8.

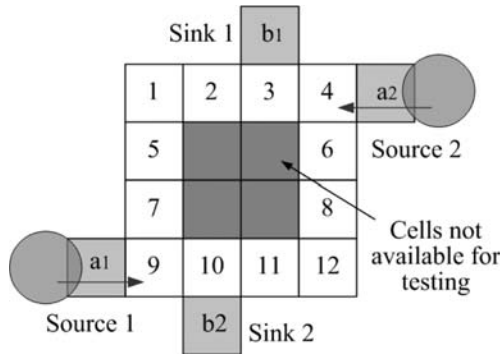


Fig. 6. An example of a 4×4 microfluidic array

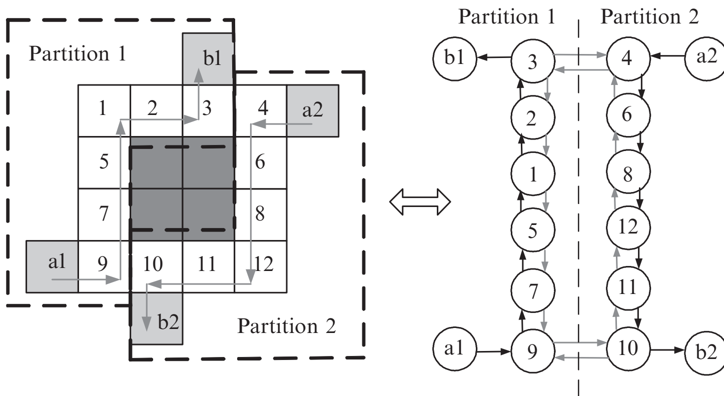


Fig. 7. An optimal partition and test stimuli droplet flow path for the 4×4 microfluidic array in Fig. 6

We have shown that an ILP model can be used to solve this optimization problem exactly for a microfluidic array of modest size. However, there exist several major limitations inherent in OPP:

(1) Sometimes there exists no Hamiltonian path in the array. Even if Hamiltonian paths exist, optimal partitioning obtained by solving OPP may not be the best solution for optimal test planning. The suboptimal nature of the test plan derived from the optimal solution to OPP results from the property of a Hamiltonian path that every node in the path should be visited exactly once. Lower-cost solutions can be obtained if we allow a cell to be visited more than once.
(2) The partitioning in OPP does not take into account the constraint that droplets can never be in a cell directly adjacent or diagonally adjacent to another droplet. The optimal solution to OPP may be not a feasible test plan because perhaps some test stimuli droplets would be adjacent to each other, thereby leading to the mixing of these droplets.

Moreover, due to the inherent complexity of the model, there is a need for heuristic algorithms that can be applied to a large array and that can eliminate the limitations inherent in the OPP problem.

7 Heuristic Algorithms

One possible heuristic method is motivated by the similarity of the test planning problem for a microfluidic array to the robot motion planning problem, where we view every test stimuli droplet as a mobile robot. However, there are a number of important differences:

(1) The test planning problem can be considered as a combination of both the navigation problem and the full coverage problem. It attempts to minimize the total time cost from the starting point (droplet source) to the end point (droplet sink), while it also requires all available cells to be covered in the droplet path. Therefore it is more complicated than either the navigation problem or the coverage problem alone.
(2) A major constraint in the application of multiple test stimuli droplets is that droplets can never be in a cell directly adjacent or diagonally adjacent to another droplet except in the case of mixing of two droplets. This restriction increases the complexity of the problem of test planning and resource optimization.

7.1 Simple Monte-Carlo Search Algorithm (SMC)

Monte-Carlo based search algorithms have been proposed in the literature for problems with a large number of constraints [16]. The key idea underlying these algorithms is that random points are generated in the search space and

the point with the lowest value for the objective function is taken to be the global optimum. In this modified random walk method, a large number of simulation runs are carried out to generate enough samples. First we apply the simple Monte-Carlo search algorithm to heuristically solve the problem of test planning and optimization. In each run, the test stimuli droplet starts from the cell directly adjacent to the droplet source and ends in the droplet sink. It randomly moves to the neighboring cell with some probability p. We mark the cell if it has been visited, then the larger p is assigned to the motion towards the unmarked cell. After randomly selecting the new positions of test stimuli droplets, the procedure checks if no two droplets are directly adjacent or diagonally adjacent in their new positions. If this restriction is satisfied, test stimuli droplets move to these new positions. Otherwise the new positions are selected again. If all available cells have been visited and test stimuli droplets have reached the droplet sinks, the test process is concluded. Here we assume that each droplet move only once in each time slot. Therefore, the test plan with the smallest number of total time slots, i.e., total test time, is selected as the optimal solution.

7.2 Modified Real-Time Algorithm (MRT)

We can further leverage real-time search algorithms and incorporate them into the heuristic algorithm for test planning. While the previous Monte-Carlo search algorithm simply marks the cell with a binary variable (0/1) based on whether it has been visited, this modified algorithm associates an evaluation function U with each cell. It always decides which neighboring cell to move to based only on the U-values of its neighbors. That is, the droplet always greedily moves to an adjacent cell with the smallest U-value. Ties due to same U-value neighbors are broken randomly. Similar to the Monte-Carlo search algorithm, the new positions of test stimuli droplets should be verified to satisfy the physical restriction. Then the U-value of the current cell is updated according to a predefined rule. We study four different U-value update rules, which been used successfully in robot motion planning, as listed in Table 1. Each rule assigns a different meaning to the U- value. For example, Node Counting interprets the U-value as the number of times the location has been visited, while LRTA* interprets U-value as approximations of the goal distances of the location [12,21]. The introduction of the evaluation function U decreases the arbitrariness of the selection of new positions in the

Table 1. Different U-value update rules

Value-update rules	Real-time search algorithms
$U(\text{current}) = 1 + U(\text{current})$	Node counting [1]
$U(\text{current}) = 1 + U(\text{New})$	Learning Real-time A*(LRTA*) [12]
If $U(\text{current}) \leqslant U(\text{New})$, $U(\text{current}) = 1 + U(\text{current})$	Wagner's value-update rule [24]
$U(\text{current}) = \max(1 + U(\text{current}), 1 + U(\text{New}))$	Thrun's value-update rule [23]

Monte-Carlo search algorithm and therefore increases the possibility of finding a better solution for the same number of simulation runs.

7.3 Proposed Improved Heuristic Algorithm for Multiple Droplets (PIH-MD)

When multiple test droplets are used, the above heuristic algorithms might move two droplets closer to each other. Additional effort may therefore be needed to prevent droplets from being directly or diagonally adjacent to each other. Moreover, if these two droplets are too close, the overlap of their coverage areas might increase, consequently leading to low efficiency in searching. Therefore we modify the heuristic algorithm for multiple test droplets by attempting to separate two droplets. We add a new evaluation function ΔP to approximate the relative distance between two droplets. When ties for new positions with the lowest U-value are encountered, we evaluate the ΔP function for every two possible positions of these droplets and select the new positions with smallest value of ΔP. Instead of breaking such ties arbitrarily as in MRT, this approach adds more guidance to heuristically find the near-optimal solution for test planning. Simulation results presented in the next section show that it provides better performance than the simple Monte-Carlo search algorithm and the modified real-time search algorithm for multiple test stimuli droplets. The procedure is outlined in Fig. 8.

8 Experimental Results

In this section, we report simulation results on test planning and resource optimization for droplet-based two-dimensional microfluidic arrays. Note that there exists an inherent tradeoff between test hardware overhead, i.e., test stimuli droplet source/sink pairs, and test application time. The location and the number of droplet sources and sinks can affect the time cost of the associated test plan [19]; a higher test hardware overhead usually leads to less testing time. Here we attempt to minimize the test application time for a given test hardware overhead. In addition, the concurrent test plan for a microfluidic array is also affected by the array configuration, i.e., the usage of cells in normal biomedical assays. In fact, design-for-testability (DFT) techniques can be used to optimize the assay schedule and array configuration to increase the efficiency of the corresponding concurrent test plan. In order to facilitate the evaluation and comparison, a set of given array configurations are deployed here for the different proposed test planning methods.

In the following experiments, two sets of cases are analyzed:

(1) A single source and a single sink;
(2) Two sources and two sinks.

```
Loop: For n = 1 to N (the maximum number of simulation runs)
   Initialization: Status initialization:
        All cells available for testing are set to '0';
        All cells not available for testing are set to '2'.
           (Here '0' denotes that the cell is not visited yet.
                 '1' denotes that the cell has been visited
                 '2' denotes that the cell is not available for testing)
        Evaluation function value initialization:
        The U values of all cells are set to 0.
   Starting point:
        The cell adjacent to source is set to be '1' when t = 1
    Loop:  For t = 2 to T (maximum index of time-slot)
           1.  Select new location of test stimuli droplet:
             Droplet moves to its neighbor cell with smallest U-value. That is,
             U(new location)=min(U(neighbors of current location). When
             there are ties, we evaluate ΔP between two droplets.
          2. Verify relative distance between new locations:
             We select the new locations which satisfy the restriction and
              have lowest ΔP.
         3. Update U-value of current location, then go on to next time-slot.
         If all available cells have been visited and test stimuli droplets have
          reached the sink, (Test finished)
         Record the time cost;
         Record test planning;
         Break; End;
     End
  If time cost < minimum cost
      minimum cost = time cost.
      Record the  best test planning.
End
```

Fig. 8. Sketch of the improved heuristic algorithm for multiple droplets

The configurations of the microfluidic arrays, e.g., the assignment of cells used for biomedical assays, as well as the locations of the source and the sink used in both sets of experiments, are shown in Fig. 9.

For arrays of modest size, optimal solutions can be obtained using ILP model. Therefore, we can compare the result of the heuristic algorithms with the optimal solution (OPT). However, for arrays of the larger size, optimal solutions are not available. The performance of heuristic algorithms in these cases can only be compared with a lower bound (LB) and an upper bound (UB) on the optimal solution as described next.

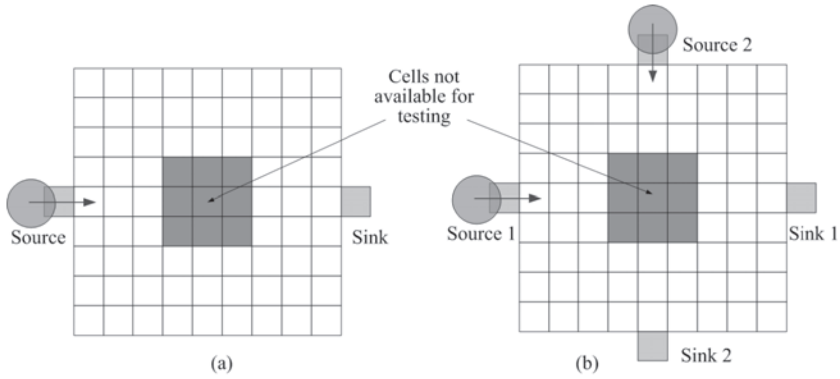


Fig. 9. Microfluidic array configurations in (**a**) the first set of experiments; (**b**) the second set of experiments

Table 2. Simulation results for Case (1)

	3×3	3×5	4×4	5×5	6×6	7×7	8×8	9×9
OPT	8	12	14	23	N/A	N/A	N/A	N/A
LB	8	10	14	22	30	41	52	64
UB	16	20	28	44	60	82	104	128
SMC	8	12	14	30	39	54	84	91
NC	8	12	14	23	34	47	66	77
LRTA*	8	12	14	25	34	47	66	81
Wagner	8	12	14	25	34	49	70	78
Thrun	8	12	14	23	32	47	62	77

The entries in the table denote testing time (in time-slots)

In an ideal case, the available cells of the array can be partitioned evenly. In each partition, there exists a Hamiltonian path from one droplet source to one droplet sink. Multiple tests can be run in non-overlapping parts in parallel without violating the restriction on droplet motion. Therefore, we have a lower bound LB on optimal solution of $\lceil n/k \rceil$, where n is the number of available cells in the system and k is the number of source-sink pairs. The tightness of this lower bound is determined by the topological configuration of the microfluidic array. In addition, an upper bound on the optimal solution can be shown to be $2 \times n$, which results from the depth-first search on a grid graph [6].

In the first set of experiments, we determined the test time for two different heuristic algorithms, i.e., the simple Monte-Carlo algorithm and the MRT (four different U-value update rules) for Case (1). We assigned 10,000 runs to the simple Monte-Carlo algorithm and 1,000 runs to the MRT. Table 2 shows the simulation results. Some optimal solutions obtained from the ILP model, as well as lower bounds and upper bounds, are also listed. The results show that heuristic algorithms provide close-to-optimal solutions for small

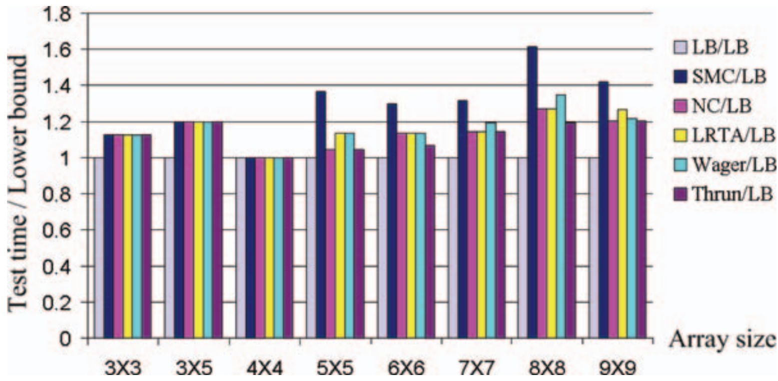


Fig. 10. Comparison of different heuristic approaches

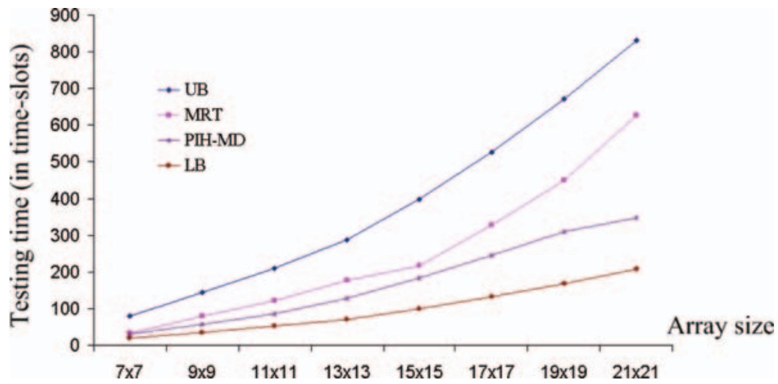


Fig. 11. Simulation results for Case (2)

array sizes, such as 3×3, 4×4 and 5×5. When the array size increases, the results for heuristic algorithms are still contained between the lower bound and upper bound of the optimal solution. The results for MRT are much closer to the lower bound than the simple Monte-Carlo algorithm; see Fig. 10. These experimental results highlight the advantage of adding the evaluation function U-value.

In the second set of experiments for multiple test stimuli droplets (Case 2), we compare the MRTs to the proposed improved heuristic algorithm (PIH-MD). Here the arrays of larger sizes are considered. Simulation result shows that the improved heuristic algorithm significantly outperforms the MRT for larger array sizes; see Fig. 11. The ratio of the actual testing time to the lower bound is always under 1.8 for the improved heuristic algorithm, while this ratio for the MRT increases with the array size; see Fig. 12.

Finally, we study the number of available solutions for each heuristic algorithm when the number of simulation runs is fixed, i.e., 500. Figure 13 shows that the proposed improved heuristic algorithm (PIH-MD) generates many

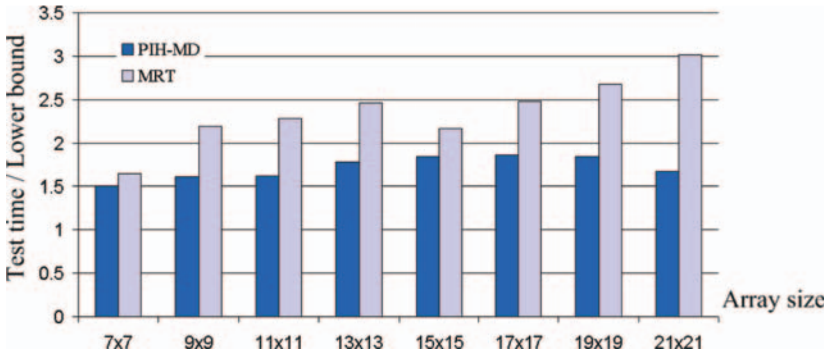


Fig. 12. Scalability of PIH-MD compared to MRT

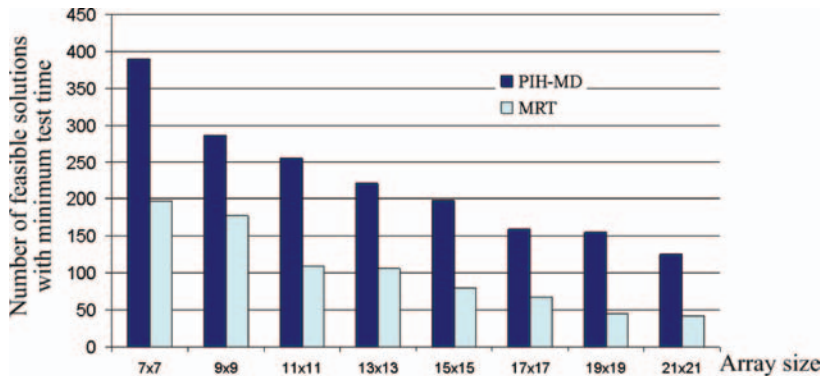


Fig. 13. Comparison of the number of available solutions for 500 simulation runs

more available solutions than the MRT. This advantage results from adding a new evaluation function ΔP to reduce the overlap between the coverage areas of the two test stimuli droplets, and it leads to a better solution for test planning and resource optimization.

9 Conclusions

In this chapter, we have presented an analysis of the test planning problem for droplet-based microfluidic systems. Due to NP-hard nature of the problem, heuristic approaches are needed. We have developed heuristic algorithms that are applicable to droplet-based microfluidic arrays of large sizes. Experiment results have shown that the heuristic solutions are close to the lower bounds on the optimal solutions. The advantage of the improved heuristic algorithm for multiple test stimuli droplets has been evaluated. In our ongoing work, we are investigating fault tolerance techniques based on the concurrent testing and reconfigurability of the droplet-based microfluidic systems.

References

1. T. Balch and R. Arkin, "Avoiding the past: a simple, but effective strategy for reactive navigation", *Proc. Int. Conf. Rob. Autom.*, pp. 678–685, 1993
2. M. Berkelaar. lpsolve. Eindhoven Univ. Technol., Eindhoven, The Netherlands. [Online]. Available: ftp://ftp.ics.ele.tue.nl/pub/lp_solve
3. K. F. Böhringer, "Optimal strategies for moving droplets in digital microfluidic systems", *Int. Conf. Miniaturized Chem. Biochem. Anal. Syst. (MicroTAS'03)*, pp. 591–594, 2003
4. N. Deb and R. D. Blanton, "Analysis of failure sources in surface-micromachined MEMS", *Proc. IEEE Int. Test Conf.*, pp. 739–749, 2000
5. J. Gross and J. Yellen, *Graph Theory and its Applications*, Boca Raton, FL: CRC Press, 1999
6. C. Icking, T. Kamphans, R. Klein and E. Langetepe, "Exploring an Unknown Cellular Environment", *Proc. Eur. Workshop Comput. Geometry*, pp. 140–143, 2000
7. A. Itai, C. H. Papadimitriou and J. L. Szwarcfiter, "Hamilton paths in grid graphs", *SIAM J. Computing*, vol. 11, pp. 676–686, 1982
8. H. G. Kerkhoff, "Testing philosophy behind the micro analysis system", *Proc. SPIE: Design, Test Microfabrication MEMS MOEMS*, vol. 3680, pp. 78–83, 1999
9. H. G. Kerkhoff and M. Acar, "Testable design and testing of micro-electro-fluidic arrays", *Proc. IEEE VLSI Test Symp.*, pp. 403–409, 2003
10. H. G. Kerkhoff and H. P. A. Hendriks, "Fault modeling and fault simulation in mixed micro-fluidic microelectronic systems", *J. Electron. Testing: Theory Appl.* (JETTA), vol. 17, pp. 427–437, 2001
11. A. Kolpekwar and R. D. Blanton, "Development of a MEMS testing methodology", *Proc. IEEE Int. Test Conf.*, pp. 923–931, 1997
12. R. E. Korf, "Real-time heuristic search", *Artif. Intell.*, vol. 42, pp. 189–211, 1990
13. W. Menz and A. Guber, "Microstructure technologies and their potential in medical applications", *Minimally Invasive Neurosurgery*, vol. 37, pp. 21–27, 1994
14. S. Mir, B. Charlot and B. Courtois. "Extending fault-based testing to microelectromechanical systems", *J. Electron. Test. Theory Appl.* (JETTA), vol. 16, pp. 279–288, 2000
15. S. Mir, H. Kerkhoff, R. D. Blanton, H. Bederr and H. Klim, "SoCs with MEMS? Can we include MEMS in the SoCs design and test flow?", *Proc. IEEE VLSI Test Symp.*, pp. 449, 2002
16. S. D. Nigam and J. U. Turner, "Review of statistical approaches to tolerance analysis", *Computer-Aided Des.*, vol. 27, pp. 6–25, 1995
17. C. H. Papadimitriou, *Computational Complexity*, Reading, MA: Addison Wesley, 1993
18. M. G. Pollack, R. B. Fair and A. D. Shenderov, "Electrowetting-based actuation of liquid droplets for microfluidic applications", *Appl. Phys. Lett.*, vol. 77, pp. 1725–1726, 2000
19. F. Su, W. Hwang, A. Mukherjee and K. Chakrabarty, "Defect-oriented testing and diagnosis of digital microfluidics-based biochip", *Proc. IEEE Int. Test Conf*, 2005
20. F. Su and K. Chakrabarty, "Design of fault-tolerant and dynamically-reconfigurable microfluidic biochips", *Proc. Des. Automation Test Eur. (DATE) Conf.,*, pp. 1202–1207, 2005

21. F. Su and K. Chakrabarty, "Architectural-level synthesis of digital microfluidics-based biochips", *Proc. IEEE Int. Conf., CAD*, pp. 223–228, 2004
22. F. Su, S. Ozev and K. Chakrabarty, "Testing of droplet-based microelectrofluidic systems", *Proc. IEEE Int. Test Conf.*, pp. 1192–1200, 2003
23. S. Thrun, *Efficient exploration in reinforcement learning.* Technical Report CMU-CS-92–102, Carnegie Mellon University, 1992
24. I. Wagner, M. Lindenbaum and A. Bruckstein, "On-line graph searching by a smell-oriented vertex process", *Proc. AAAI Workshop on On-Line Search*, pp. 122–125, 1997
25. H. P. Williams, *Model Building in Mathematical Programming*, New York: Wiley, 1999
26. T. Zhang, K. Chakrabarty and R. B. Fair, *Microelectrofluidic Systems: Modeling and Simulation*, Boca Raton, FL: CRC Press, 2002
27. International Technology Roadmap for Semiconductor (ITRS), http://public.itrs.net/Files/2003ITRS/Home2003.htm

Chapter 11: Testing and Diagnosis of Realistic Defects in Digital Microfluidic Biochips

F. Su, W. Hwang, A. Mukherjee, and K. Chakrabarty

1 Introduction

Over the past decade, research in integrated circuit testing has broadened from digital test to include the testing of analog and mixed-signal devices. More recently, new test techniques for mixed-technology microelectromechanical systems (MEMS) are also receiving attention [1–5]. As MEMS rapidly evolve from single components to highly integrated systems for safety-critical applications, dependability is emerging as an important performance parameter. Fabrication techniques such as silicon micromachining lead to new types of manufacturing defects in MEMS [2]. Moreover, due to their underlying mixed technology and multiple energy domains (e.g., electric, mechanical, and fluidic), such composite microsystems exhibit failure mechanisms that are significantly different from those in electronic circuits. In fact, the 2003 International Technology Roadmap for Semiconductors (ITRS) recognizes the need for new test methods for disruptive device technologies that underly composite microsystems, and highlights it as one of the five difficult test challenges beyond 2009 [6].

Microfluidics-based biochips constitute an emerging category of mixed-technology microsystems [7]. Recent advances in microfluidics technology have led to the design and implementation of miniaturized devices for various biochemical applications. These microsystems, referred to interchangeably in the literature as microfluidics-based biochips, lab-on-a-chip and bioMEMS [8, 9], promise to revolutionize biosensing, clinical diagnostics, and drug discovery. Such applications can benefit from the small size of biochips, the use of microliter/nanoliter sample volumes, lower cost, and higher sensitivity compared to conventional laboratory methods.

The first generation of microfluidics-based biochips was based on the manipulation of continuous liquid flow through fabricated microchannels [7]. Liquid flow was achieved either by external pressure sources, integrated mechanical micropumps, or by electrokinetic mechanisms such as electroosmosis. Recently, a novel microfluidics technology has been developed to

manipulate liquids as discrete microliter/nanoliter droplets. Following the analogy of digital electronics, this technology is referred to as "digital microfluidics" [8]. Compared to continuous-flow systems, digital microfluidics offer the advantage of dynamic reconfigurability and architectural scalability.

The level of system integration and the complexity of digital microfluidics-based biochips are expected to increase in the near future due to the growing need for multiple and concurrent bioassays on a chip [9]. However, shrinking processes, new materials, and the underlying multiple energy domains will make these biochips more susceptible to manufacturing defects. Moreover, some manufacturing defects are expected to be latent, and they may manifest themselves during field operation of the biochips. In addition, harsh operational environments may introduce physical defects such as particle contamination during field operation. Consequently, robust off-line and on-line test techniques are required to ensure system dependability as biochips are deployed for safety-critical applications such as field diagnostics tools to monitor infectious disease, and biosensors to detect biochemical toxins and other pathogens.

Although research in the design of digital microfluidics-based biochips has gained considerable momentum in recent years [8–10], only limited work has been reported thus far on biochip testing. A cost-effective test methodology for digital microfluidic systems was first described in [11]. Likely physical defects in such systems were analyzed and faults were classified as being either catastrophic or parametric. Faults are detected in [11] by electrically controlling and tracking the motion of test droplets. An optimal test planning method for the detection of catastrophic faults in digital microfluidic arrays was investigated in [12]. It is based on a graph model of the microfluidic array and a problem formulation based on Hamiltonian paths in a graph. An efficient concurrent testing method that interleaves test application with a set of bioassays was proposed in [13]. Reconfiguration and defect tolerance techniques for biochips were described in [14, 15].

Prior work on the testing of digital microfluidics-based biochips is based on invalid assumptions regarding the impact of certain defects on droplet flow. For example, a common defect seen in fabricated microfluidic arrays is a short-circuit between two adjacent electrodes [11]. It was assumed in [11–13] that this defect causes a droplet to be stuck at one of the two electrodes irrespective of the orientation of liquid flow. No attempt was made in prior work to experimentally validate this assumption. Experiments show however that the effect of this short-circuit defect on droplet flow depends on whether the droplet flow path is perpendicular to the two shorted electrodes or aligned with them. A test procedure for such defects should therefore not only test single cells as in [11–13], but it should also focus on pairs of cells and the traversal of droplets from one cell to all its neighbors. In addition, no systematic attempt has been made to relate defects to fault models and observable errors.

No attempt has been made in prior work to account for the hardware cost of droplet sources and sinks. The locations of droplet sources and sinks are

determined manually, and the problem of determining these locations is not incorporated in the test planning problem. Moreover, as shown in [14, 15], digital microfluidic biochips offer dynamic reconfigurability to support defect tolerance, whereby groups of cells in a microfluidic array can be reconfigured to change their functionality in order to bypass defective cells. To facilitate this reconfiguration, we not only need a pass/fail test, but we also need to locate faulty cells. However, prior work has not addressed the issue of fault diagnosis in microfluidic arrays.

In this chapter, we attempt to address the above issues for digital microfluidics-based biochips. First we relate some realistic defects to fault models and observable errors. We next set up an experiment to evaluate the manifestation of electrode shorts at the fluidic behavioral level. Motivated by the experimental results, we present a testing methodology based on graph theory to detect catastrophic faults, including those caused by electrode shorts. While this method can easily determine a test droplet flow path for off-line testing, we show that it can be extended to support on-line testing, whereby the test procedure is performed concurrently with a set of bioassays. This methodology can also automatically determine the location of test droplet sources/sinks to optimize the test plan. In addition, we investigate the problem of fault diagnosis. We apply this methodology to a real-life biochip performing multiplexed biochemical assays, and compare our results with the results reported in [13].

2 Prior Work

MEMS is a relatively young field compared to microelectronics. The heterogeneity inherent in MEMS, resulting from the use of interacting mechanical and electronic devices, gives rise to many possible failure mechanisms and failure modes that are quite different from those in microelectronics. Thus efficient fault models and test generation methods for MEMS remain a major challenge. Recently, fault modeling and fault simulation for surface-micromachined MEMS have been analyzed [1–4]. In [1, 2], a comprehensive testing methodology for surface micromachined sensors has been presented. High-reliability and safety-critical markets for MEMS, e.g., accelerometers used in automobiles, are driving the integration of efficient built-in self-test and on-line monitoring functions. Design-for-manufacturing (DFM) and design-for-testability (DFT) methodologies have been incorporated in the design flow for MEMS [16].

However, test techniques for classical MEMS cannot be directly applied to microfluidic systems, since they differ in the underlying energy domains and in their working principles. The techniques and tools currently in use for the testing of classical MEMS (e.g., comb-drive microresonator) mainly aim at mechanical defects such as stiction; they do not handle fluids. Thus new testing techniques are required for microfluidics-based biochips. Very limited work has been reported in this area. Recently, fault modeling and fault simulation for

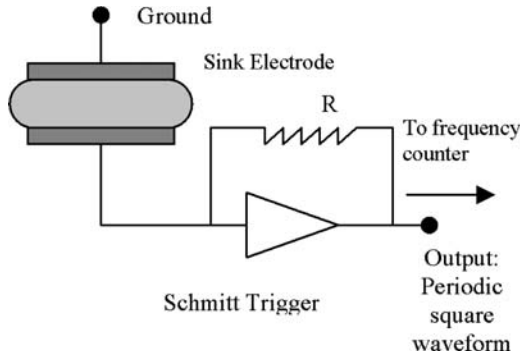


Fig. 1. Simple capacitive sensing circuit

continuous-flow microfluidic biochips have been proposed in [17, 18]. Also, a DFT technique for microfluidic systems based on electro-osmotic flow has been discussed in [19].

The first attempt to address testing problem of droplet-based "digital" microfluidic biochips is described in [11]. In the proposed testing methodology, test stimuli droplets containing the normal conductive fluid (e.g., 0.1 M KCL) are released into a two-dimensional microfluidic array from on-chip reservoirs, and are guided through the system following the designed testing scheme. Both catastrophic and parametric faults are detected by electrically controlling and tracking the motion of these test stimuli droplets. This testing method is minimally invasive and easy to implement, thus it alleviates the need for expensive and bulky external testing devices.

The proposed unified detection mechanism consists of a simple RC oscillator circuit formed by the sink electrodes and the fluid between them as an insulator; see Fig. 1 [12]. The capacitance of this structure depends on the presence of the droplet since the filler medium and the droplet have distinct permittivities. By sensing the capacitance of this structure using a simple frequency counter, one can determine whether a droplet has reached the sink. This mechanism can be electronically implemented and easily integrated on-chip. In order to provide a unidirectional and unambiguous detection mechanism, the pass/fail criterion has to be determined based on the presence of the droplet at the sink electrode and this criterion should be applied for all test cases, i.e., the fault-free operation is associated with the presence of the droplet at the sink electrode and faulty operation with its absence.

In [12, 13], the test planning problem was formulated in terms of the graph partitioning and the Hamiltonian path problems from graph theory. The key idea underlying this optimization approach is to model the two-dimensional microfluidic array as a directed graph, and then partition it into non-overlapping subgraphs. In each subgraph, a Hamiltonian path from one source to one sink is determined as the flow path of test droplets such that each cell of a microfluidic array can be traversed. If one single cell becomes

faulty, test droplet would be stuck during its transportation. The faulty status can be easily determined by observing the corresponding test points (i.e., sink electrodes connected with capacitive sensing circuits).

3 Fault Modeling

Like microelectronic circuits, a defective microfluidic biochip is said to have a failure if its operation does not match its specified behavior. In order to facilitate the detection of defects, fault models that efficiently represent the effect of physical defects at some level of abstraction are required. These models can be used to capture the effect of physical defects that produce incorrect behaviors in the electrical or fluidic domain. As described in [11], faults in digital microfluidic systems can be classified as being either catastrophic or parametric. Catastrophic faults lead to a complete malfunction of the system, while parametric faults cause degradation in the system performance. Table 1 lists some common failure sources, defects, and the corresponding fault models for catastrophic faults in digital microfluidic biochips.

Table 1. Some failure sources, corresponding defects, fault models, and observable errors in digital microfluidic biochips

Failure source	Defect	Fault model	Observable error
Excessive voltage applied to electrode	Dielectric breakdown	Short between the droplet and the electrode	Droplet undergoes electrolysis, which prevents its further transportation
Abnormal metal layer deposition and etch variation during fabrication	Metal connection between two adjacent electrodes	Electrode short	A droplet resides in the middle of these two shorted electrodes, and its transport along one or more directions cannot be achieved
	Broken control wire to control source	Electrode open	A failure in activating the electrode for droplet transport
Particle contamination	Fluidic high-impedance between plates	Fluidic open	A droplet cannot move across the obstacle
Abnormal environment – temperature variation during operation	Unexpected viscosity change of fluids (droplet and filler medium)	Fluidic open	A droplet cannot move due to the increased friction at the droplet/medium interface

In addition, physical defects that cause parametric faults include geometrical parameter deviations. The deviation in insulator thickness, electrode length, and height between parallel plates may exceed their tolerance value during fabrication. Note that, even though the methodology described in this chapter is applied to only catastrophic faults, it can easily be extended for the detection of parametric faults based on the unified detection mechanism proposed in [11].

It is evident that catastrophic faults can lead to a complete cessation of droplet transportation. However, there exist differences between their corresponding erroneous behaviors. For instance, to test for the electrode-open fault, it is sufficient to move a test droplet from any adjacent cell to the faulty cell. The droplet will always be stuck during its motion due to the failure in charging the control electrode. On the other hand, if we move a test droplet across the faulty cells affected by an electrode-short fault, the test droplet may or may not be stuck depending on its flow direction. In the next section, we design a defect-oriented experiment to evaluate the behavioral impacts of electrode-short faults.

4 Defect-Oriented Experiments

4.1 Microfluidic Biochip Description

The microfluidic biochip discussed in this chapter is based on the manipulation of microliter–nanoliter droplets using the principle of electrowetting-on-dielectric (EWOD) [8, 20]. Electrowetting refers to the modulation of the interfacial tension between a conductive fluid and a solid electrode coated with a dielectric layer by applying an electric field between them. An imbalance of interfacial tension is created if an electric field is applied to only one side of the droplet; this tension gradient forces the droplet to move.

The basic cell of an EWOD-based digital microfluidic biochip consists of two parallel glass plates, as shown in Fig. 2. The bottom plate contains a patterned array of individually controllable electrodes, and the top plate is coated with a continuous ground electrode. The control electrodes in the bottom plate are coated with a dielectric insulator, e.g., parylene C, for insulation. A hydrophobic thin film is also added to the top and bottom plates

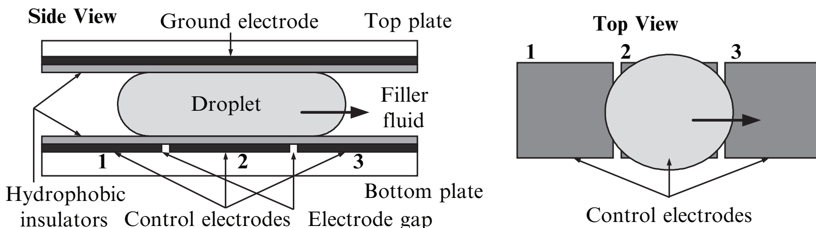


Fig. 2. Basic cell used in a digital microfluidic biochip

to decrease the wettability of the surface and to add capacitance between the droplet and the control electrode. The droplet containing biochemical samples and the filler medium, such as the silicone oil, are sandwiched between the plates; the droplets travel inside the filler medium.

In order to move a droplet, a control voltage is applied to an electrode adjacent to the droplet (e.g., electrode 3 in Fig. 2) and at the same time the electrode just under the droplet (e.g., electrode 2 in Fig. 2) is deactivated. Thus, the charge in the droplet/insulator interface that is accumulated over the activated electrode results in an interfacial tension gradient, which consequently causes droplet transport. By varying the electrical potential along a linear array of electrodes, microliter/nanoliter-volume droplets can be transported along this line of electrodes. The velocity of the droplet can be controlled by adjusting the control voltage ($0 \sim 90\,\mathrm{V}$), and droplets have been observed to move with velocities up to $20\,\mathrm{cm\,s^{-1}}$ [8]. Furthermore, based on this principle, droplets can be transported freely to any location on a two-dimensional array without the need for micropumps and microvalves that are required in continuous-flow systems.

Using a two-dimensional microfluidic array, many common operations for different bioassays can be performed, such as sample movement (*transport*), temporary sample preservation (*store*), and the mixing of different samples (*mix*). For instance, the *store* operation is performed by applying an insulating voltage around the droplet. The *mix* operation is used to route two droplets to the same location and then turn them about some pivot points. Note that these operations can be performed anywhere on the array, whereas in continuous-flow systems they must operate in a specific micromixer or microchamber. This property is referred to as the reconfigurability of a digital biochip. The configurations of the array, i.e., the droplet transport routes and their rendezvous points, are programmed into a microcontroller that controls the voltages of electrodes in the array.

The in vitro measurement of glucose and other metabolites, such as lactate, glutamate, and pyruvate, is of great importance in clinical diagnosis of metabolic disorders. A colorimetric enzyme-kinetic glucose assay has been recently demonstrated in lab experiments on a digital microfluidic biochip [10, 21, 22]. This biochip uses a digital microfluidic array, which moves and mixes droplets containing biochemical samples and reagents, and an integrated optical detection system consisting of a LED and a photodiode; see Fig. 3 [10, 21, 22]. We further envision that a disposable microfluidic biochip will be easily plugged into a controller circuit board that can be programmed and powered via a standard USB port, as shown in Fig. 4.

4.2 Experiment Design

To evaluate the effect of an electrode short on microfluidic behavior, we design an experiment using a 2×4 microfluidic array as shown in Fig. 5a. This experiment includes two steps. First, we impose the condition that two electrodes

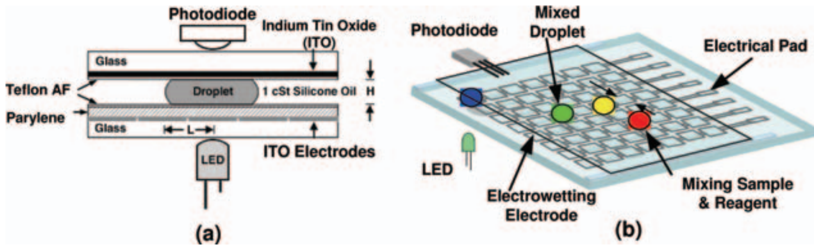


Fig. 3. Schematic of a digital microfluidic biochip used for colorimetric assays: (**a**) basic cell; (**b**) top view of microfluidic array

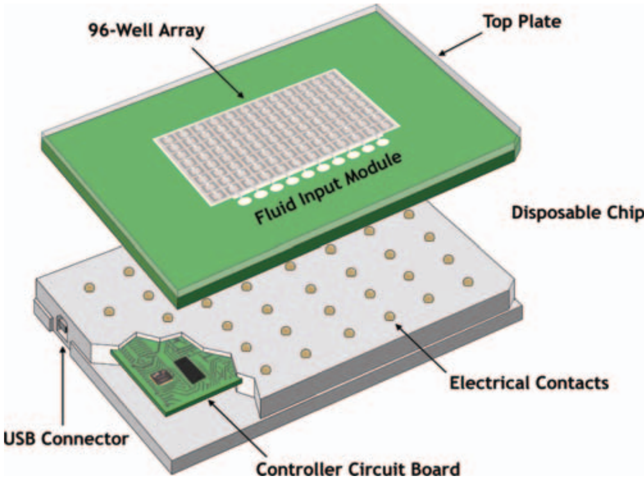


Fig. 4. The concept of a commercial disposable microfluidic biochip

adjacent in the X-direction, e.g., electrode 6 and 7 in Fig. 5b, are shorted. A horizontal flow path, e.g., 5 → 6 → 7 → 8, is used to guide a test droplet across the shorted cells. The effect of the short between two adjacent electrodes can be simulated by simultaneously changing the voltages on these two electrodes. In the second step, two electrodes adjacent in the Y-direction, e.g., electrode 2 and 6 in Fig. 5c are considered to be shorted. As in the first step, a test droplet traverses the faulty cell (electrode 6) following a flow path in the X-direction (e.g., 5 → 6 → 7). For both steps, we use optical devices such as CCD cameras to visually inspect if the test droplet is stuck during its transportation.

4.3 Chip Fabrication

The 2×4 microfluidic array used in the experiment was fabricated using standard microfabrication techniques. The detailed fabrication process is described in [20]. The control electrodes in the bottom glass plate are formed

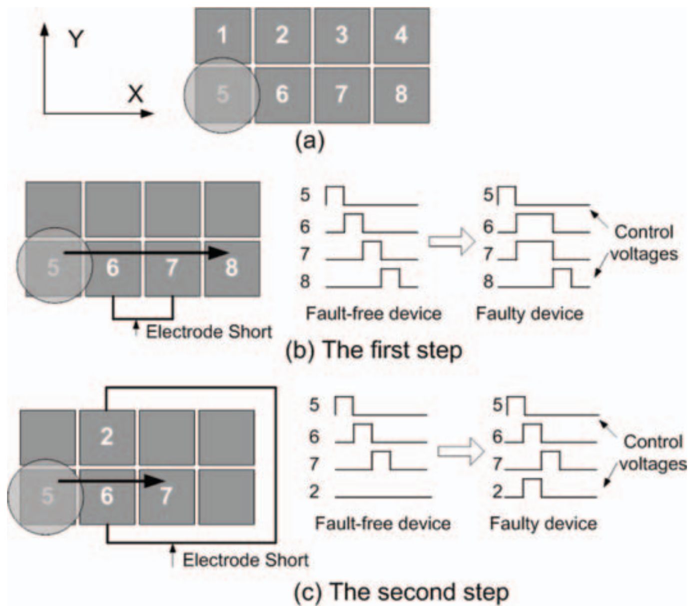


Fig. 5. Design of an experiment to study microfluidic behavior in the presence of the electrode-short fault

by a 200 nm thick layer of chrome, which is further coated with a layer of parylene C (800 nm) as a dielectric insulator. This microfluidic array uses a 1.0 mm electrode pitch size. A layer of optically transparent indium tin oxide (ITO) in the top glass plate is used as the continuous ground electrode. In addition, a 50-nm-thick film of Teflon AF 1600 is added as the hydrophobic coating on both the top and the bottom plates. The 600 μm gap between the top and bottom plates is set using a glass spacer.

4.4 Experimental Setup

The experimental setup for testing the 2×4 microfluidic array is shown in Fig. 6. The chip-under-test was mounted on a custom-assembled platform. We use a custom-made electronic unit to independently control the voltages of each control electrode in the array by switching them between ground and a DC actuation voltage. In our experiments, the actuation voltage was set at 50 V. A 1-microliter test droplet containing 0.1 M KCL was dispensed onto the chip using a micropipettor; the filler fluid medium, i.e., 1 cSt silicone oil was introduced after droplet dispensing. Images of droplet transportation during the experiment were obtained with an industrial microscope (VZM 450i, Edmund Industrial Optics) and a color CCD camera (Sony XC-999). Images were either captured directly to a PC using a frame grabber (MicroDC30, Pinnacle Systems) or were video-recorded with a super-VHS videocassette recorder (JVC-S4600).

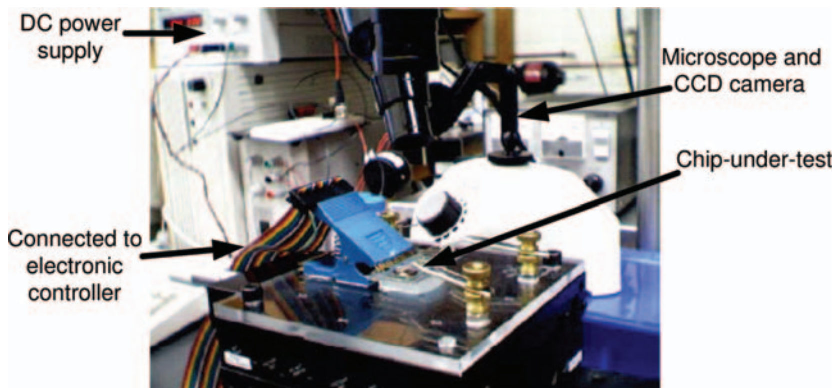


Fig. 6. Experimental setup

4.5 Results and Analysis

In the first step of the evaluation experiment, we let a test droplet move through two electrodes that are adjacent in the X-direction. As indicated before, these two electrodes are effectively shorted by setting them to identical voltages. A droplet aligns itself with the charged electrode to maximize the area of overlap and therefore the electrostatic energy stored in the effective capacitors between the droplet and the electrode. Thus the test droplet resides around the middle of two shorted electrodes as shown in Fig. 7. Since there is no overlap between this droplet and neighboring electrode (i.e., electrode 8), the test droplet cannot be further moved to electrode 8; it is stuck between electrode 6 and electrode 7 in the experiment.

The second step of the experiment is to investigate what happens when there is a short between two electrodes that are adjacent in the Y-direction. Interestingly, our experiment shows that in this case, the test droplet can still move across electrode 6, even though this electrode is shorted with electrode 2; see Fig. 8. We can explain this phenomenon on the basis of the fact that there still exists sufficient overlap between the test droplet and electrode 7, even though the droplet tends to move towards the middle of electrodes 6 and 2. Thus, the test droplet is not stuck if it follows the test plan $5 \rightarrow 6 \rightarrow 7$.

The above experimental results provide useful insights on how testing should be carried out for microfluidic arrays. We find that electrode short faults lead to an error only when the droplet flow path is aligned with the orientation of the electrode shorts. In addition to electrode short, there exist other physical defects that lead to similar erroneous behavior. For example, particle contamination between two adjacent cells also produces an error under specific droplet flow paths. In order to detect these defects, a test plan should guide the test droplet to move from a cell in the array to all its neighbors.

These experimental results also highlight a major deficiency of prior work on the testing of microfluidic arrays [12,13]. The previous approaches map the

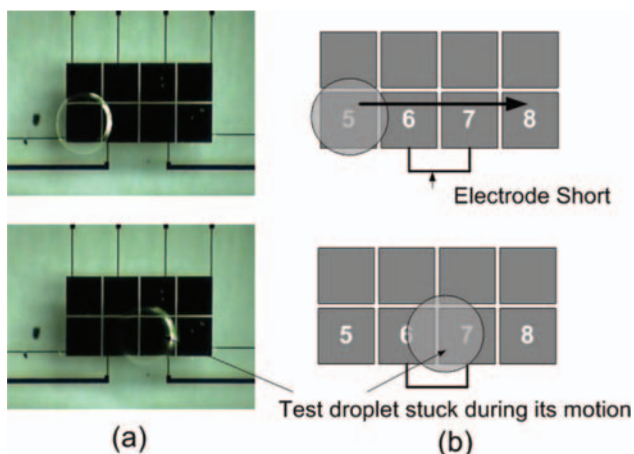


Fig. 7. Experimental results and analysis for the first step

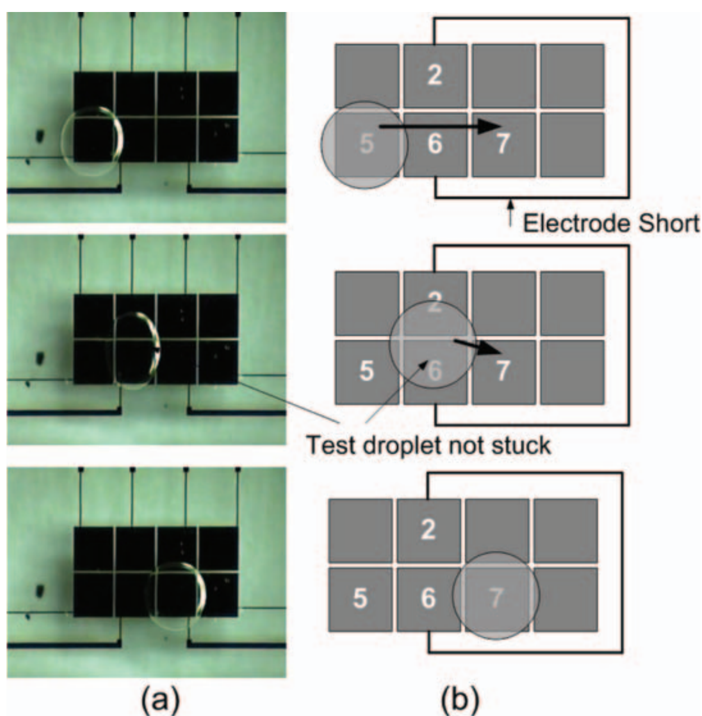


Fig. 8. Experimental results and analysis for the second step

droplet flow path problem to that of finding a Hamiltonian path in a graph model of the array. In other words, the test droplet is routed through the array such that it visits every cell exactly once. While this approach guarantees

the detection of faults involving only one electrode or cell, it is not sufficient to detect electrode-short and fluidic-open faults that affect two adjacent electrodes. This is highlighted in the next section.

5 Testing and Diagnosis

The "edge-dependent" nature of some defects (e.g., electrode shorts), as seen in Sect. 4, indicates that test planning methods proposed in [12, 13], which are based on the notion of the Hamiltonian path from graph theory, are not sufficient for fault detection. For example, in Fig. 5c the test droplet path $5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$ fails to detect an electrode short fault between electrodes 2 and 6, even though this Hamiltonian path-based flow visits each cell exactly once. Thus, a new test planning method is required to deal with this problem. Since this type of defect can be introduced into microfluidic biochips not only during fabrication (e.g., electrode shorts due to manufacturing problems), but also during in-field operation (e.g., due to particle contamination and electrode metal migration), both off-line and on-line testing techniques are necessary. In addition, to support defect tolerance based on reconfiguration, a diagnosis technique is needed to locate candidate fault sites in a microfluidic array that is deemed to be faulty by the testing procedure.

5.1 Off-Line Testing

Test droplets are first dispensed onto the microfluidic array from the droplet source (i.e., on-chip reservoir and dispensing port). They are then routed through the biochip-under-test, i.e., traversing all the cells and cell boundaries. If there exists a catastrophic fault on the chip, the test droplet gets stuck at an intermediate point. Otherwise, it is eventually guided back to the droplet sink. The sink electrode is connected to a capacitive detection circuit that can determine the presence of the test droplet [11]. In this way, we can easily determine the faulty or fault-free status of the microfluidic biochip from the electrical output of the detection circuit.

We formulate the test planning problem in terms of the Euler circuit and Euler path problems from graph theory [21]. The key idea underlying this approach is to model the digital microfluidic array under test as an undirected graph, and then "eulerize" this graph. On the basis of Euler's theorem [21], a flow path for the test droplet can be easily obtained, which allows us to detect shorts between any two directly adjacent electrodes in the array.

First, we model the array of microfluidic cells using an undirected graph $G = (V, E)$ where the set of vertices V represents the set of microfluidic cells in the array, and each edge is an unordered pair of vertices. The edge $\{u,\ v\} \in E$ if and only if vertex u and vertex v represent two directly adjacent

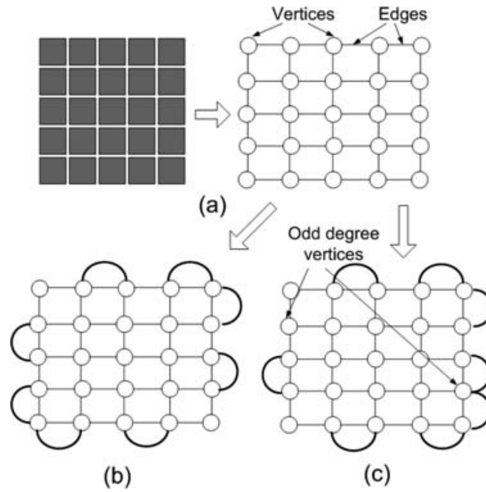


Fig. 9. (**a**) Graph model for a 5×5 microfluidic array; (**b**) eulerized graph containing an Euler circuit; (**c**) eulerized graph containing an Euler path

microfluidic cells. Figure 9a shows an example of the graph model for a 5×5 microfluidic array.

An Euler path in a graph G is defined as a path that traverses all the edges of G exactly once [23]. Similarly, an Euler circuit is a cycle that traverses all the edges of the graph exactly once. We know from [23] that an undirected graph has an Euler circuit if and only if it is connected, and each vertex has even degree. Moreover, an undirected graph has an Euler path if it is connected and has exactly two vertices of odd degree. The Euler path must start at one of the odd-degree vertices and must end at the other odd-degree vertex [23].

Euler's theorems give us the means for finding efficient ways in which to traverse all the edges of an undirected graph. However, we notice that a graph model of a microfluidic array usually has more than two vertices of odd degree. Thus we have to retrace some of the edges in order to traverse all edges at least once. In order to apply existing Euler cycle/path algorithms, we need to convert some or all the vertices of odd degree to even degree by adding additional edges. The process of eliminating odd degree vertices by adding additional edges is called *eulerizing* the graph. There are two different ways for eulerizing the graph model of a microfluidic array, depending on whether an Euler circuit or an Euler path is desired. For example, as shown in Fig. 9b, there exists an Euler circuit in the eulerized graph model for a 5×5 microfluidic array since each vertex becomes to be even degree. On the other hand, another eulerized graph in Fig. 9c contains an Euler path starting from one odd-degree vertex, e.g., cell (2,1) and ending at another odd-degree vertex, e.g., cell (4,5).

Although both these eulerizing methods can provide an edge tour as the feasible flow path of a test droplet, we use the first method (i.e., to find an Euler circuit) here. There are two main reasons for this choice. First, in the second eulerizing method we must use the node with odd degree as the starting or the ending point. Thus, to find an Euler path between another pair of cells, a different eulerized graph is required. In contrast, since any vertex can be used as the start and end point of an Euler circuit, we can locate the test droplet source/sink adjacent to any boundary cell using the same eulerized graph in the first method. Thus, this method is especially suitable when we try to determine the optimal location of droplet sources and sinks. Second, we are motivated by considerations of physical implementation. If we merge the test droplet source and sink, i.e., connect the electrode of the dispensing port to the capacitive detection circuit, it not only reduces the area overhead of the test hardware, but it can also conserve the liquid volume of on-chip reservoir by recycling test droplets. This reduces the cost of manual maintenance. This feature is especially desirable for in-field testing.

Using the selected Eulerizing method, a graph model for the microfluidic array under test is modified to $G' = (V,\ E')$, where the new set of edges E' includes all edges from E as well as the additional edges. The following theorem quantifies the number of additional edges that are necessary.

Theorem 1. *The minimum number of additional edges Na required to eulerize an $m \times n$ microfluidic array such that an Euler circuit exists in the corresponding graph, is given by:*

$$Na = \begin{cases} m + n - 4, & \text{if } m \text{ and } n \text{ are even;} \\ m + n - 2, & \text{otherwise.} \end{cases}$$

Proof. Since in an $m \times n$ array all internal vertices have even degree, i.e., 4, we only need to add additional edges to the boundary vertices. Then this theorem can easily be proven using three different cases. 1) if m and n are both odd, $Na = 2\left\lfloor \frac{m-1}{2} \right\rfloor + 2\left\lfloor \frac{n-1}{2} \right\rfloor = m + n - 2$; 2) if m or n is even and another one is odd, $Na = \left\lfloor \frac{m-1}{2} \right\rfloor + \left\lfloor \frac{m}{2} \right\rfloor + \left\lfloor \frac{n-1}{2} \right\rfloor + \left\lfloor \frac{n}{2} \right\rfloor = m + n - 2$; 3) if m and n are both even, $Na = 2\left\lfloor \frac{m-1}{2} \right\rfloor + 2\left\lfloor \frac{n-1}{2} \right\rfloor = m + n - 4$. □

Based on Theorem 1, we find that the total number of edges of an eulerized graph model $G' = (V,\ E')$ for an $m \times n$ microfluidic array is as follows:

$$N(E') = N(E) + Na = (2mn - m - n) + Na$$
$$= \begin{cases} 2\ mn - 4, & \text{if } m \text{ and } n \text{ are even;} \\ 2\ mn - 2, & \text{otherwise.} \end{cases}$$

We next define the length of a time slot to be equal to the time during which a test droplet moves from one cell to an adjacent one. Thus, the total testing application time is $N(E')$ time slots, if a test droplet follows an Euler circuit-based path.

Procedure FLEURY'S ALGORITHM

1 Make sure the graph is connected and all vertices have even degree
2 Start at any vertex
3 Travel through an edge that is not visited if
 a) it is not a bridge for the part not visited, or
 b) there is no other alternative
4 Label the edges in the order in which they were visited
5 When there is no edge not visited, an Euler circuit is found.

Fig. 10. Pseudocode of Fleury's algorithm [23]

To find an Euler circuit in the eulerized graph, we use the well-known Fleury's algorithm; its pseudocode is shown in Fig. 10 [23]. The advantage of this algorithm is that since it is a real-time search algorithm, it can be easily modified to handle both multiple test droplets and the concurrent testing problem.

The identification of an edge as a bridge, i.e., cut edge,[1] in Fleury's algorithm can be achieved by applying depth-first search to check the connectivity of the untested part of the graph [24]. Although it works well for a microfluidic array of modest size, its complexity is $O(n+e)$, where n and e are the number of vertices and edges in the part of an undirected graph that has not been visited, respectively. This amounts to high computation cost because of the need for iterative connectivity checking during the search for an Euler circuit. Therefore, we modify Fleury's algorithm by replacing bridge checking with a probabilistic search procedure based on some simple rules of complexity $O(1)$. We probabilistically select the edge to visit. The probability assignment is based on some simple rules, which can be used as guidelines to find Euler circuits; some of these rules are listed as follows:

1. *Do not use an edge to go to a vertex unless there is another edge available to leave that vertex* (*except for the last step*). An example of probability assignment based on this rule is shown in Fig. 11a.
2. *An edge that belongs to a loop is not a bridge.* Note that if there exist two "not visited" edges between two adjacent vertices, they form a loop. Thus, we can select one such edge with a higher probability compared to other edges; see Fig. 11b.

Although this rule-based search cannot guarantee the identification of an Euler circuit in one run, an appropriate number of simulation runs can easily lead to the desired result. This method is scalable to large problem sizes. In addition, the starting point, i.e., the location of droplet source and sink, can be selected at random, which is especially important for multiple test droplets and for concurrent testing. The pseudocode of this probabilistic modified Fleury's algorithm (PMF) is shown in Fig. 12.

[1] A cut edge (bridge) of a graph G is an edge whose removal disconnects G.

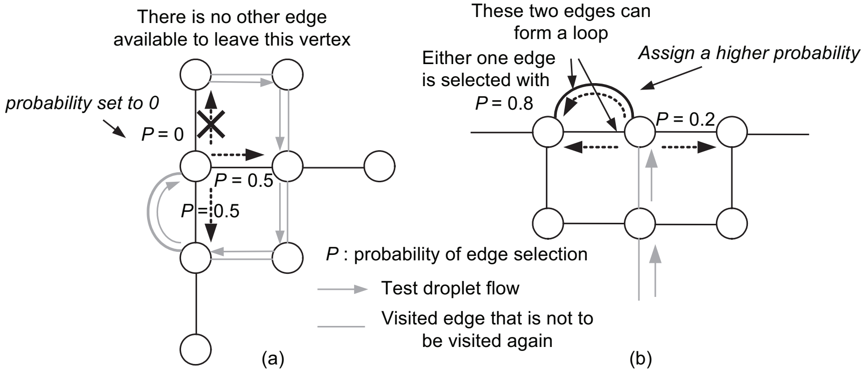


Fig. 11. Illustration of simple rules

Procedure PMF ALGORITHM

/* Probabilistic modified Fleury's algorithm */
1 **Loop**: For $n = 1$ to N (maximum number of simulation runs)
2 Select vertex v_n (1) as the starting point at random
$\{v_n$ (1) $\in V$: it represents the boundary cell on the array$\}$
3 **Repeat** {/* test one "not visited" edge at each time step t*/
4 Determine candidate edges $E(t) =$
$\{e \in E$: it is not visited and one of its end vertex is $v_n(t)\}$
5 Select $e \subseteq E(t)$ with probability $P(e)$
/* $P(e)$ is assigned to edge e based on simple rules*/
6 Visit e, and set $v_n(t + 1) =$ another end vertex of e
7 $t = t + 1\}$
8 **Until** ($E(t)$ is empty)
9 **If** (all edges have been tested)
10 /*An Euler circuit-based test plan found*/
11 Record a test plan $\{v_n(t)\}$
12 **Else** Search for an Euler circuit failed
13 **End if**
14 Record the location of source and sink, i.e., $v_n(1)$
15 **End loop**

Fig. 12. Pseudocode of the PMF algorithm

The Euler circuit-based method can be further extended to find a test schedule for more than one test droplet. We first partition the graph model of a microfluidic array into subgraphs, and then eulerize them individually such that there exists an Euler circuit in each subgraph. In this way, multiple test droplets can perform the edge-tour testing simultaneously in different parts of the microfluidic array. The total testing application time is the maximum of the testing time for any of these subgraphs. This leads to the reduction of the testing time at the expense of test hardware overhead, corresponding to multiple droplet sources/sinks. Figure 13 shows an example of two test

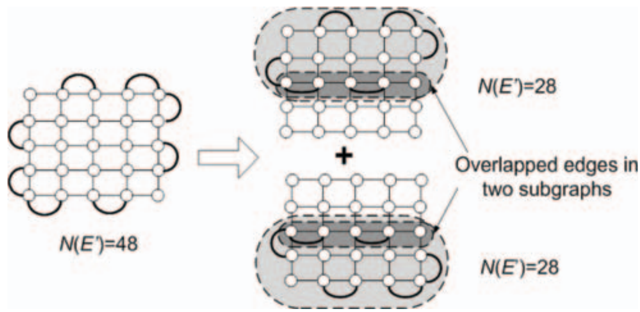


Fig. 13. Application of two test droplets to a 5×5 microfluidic array

droplets that are applied to a 5×5 microfluidic array. The testing time can be reduced significantly, i.e., from 48 time slots to 28 time slots. Note that there exist overlaps between the different subgraphs in order to cover all edges in the graph, as shown in Fig. 8. However, we must not allow two test droplets to traverse an edge at the same time. In addition, an important constraint arising from fluidic considerations is that a droplet should never be in a cell directly adjacent or diagonally adjacent to another droplet; otherwise, these two droplets will mix together. This restriction increases the complexity of test planning problem and it may introduce waiting time (stall cycles) for some test droplets. The proposed PMF algorithm can be easily modified to solve the above problem. To ensure that fluidic constraints are satisfied, we assign a random (but distinct) priority to each test droplet; the test droplet movements are planned in prioritized order, whereby in each time step the test droplet with higher priority is scheduled first, and the droplet with lower priority attempts to avoid the droplet with higher priority.

5.2 On-Line Testing

Some cells in a digital microfluidic biochip may be rendered faulty during in-field operation. Therefore, on-line concurrent testing, which allows testing and normal bioassays to run simultaneously on a chip, can play an important role in alerting the user to an unpredictable faulty status.

We can easily modify the PMF algorithm to derive a test plan that support on-line concurrent testing. We assume that the schedule of a bioassay performed on the microfluidic biochip is known a priori, e.g., using methods described in [9]. The goal of a desirable test plan is to avoid conflicts with the normal assay operation while traversing all the edges in the array. Thus, an additional evaluation step is added to the search procedure in the PMF algorithm, i.e., in each time step we need to check the other endpoint (vertex) of each candidate edge. If this vertex represents the cell that is occupied by the assay operation at this time slot or adjacent to an assay droplet, the corresponding edge cannot be visited. If no edges are available at this time

step, the test droplet must wait at the current cell until there is an available edge to visit. The total concurrent testing time equals Euler tour time, i.e., $N(E')$ time slots, plus the waiting time. Different locations of test droplet sources and sinks can affect the on-line testing time. By randomly selecting the starting point, the PMF algorithm attempts to find the best location of test droplet sources and sinks to minimize the testing time. Moreover, as in off-line testing, multiple test droplets can be applied to reduce the testing time, whereby each test droplet is guided to traverse the partition and also does not conflict with the bioassay in this region.

5.3 Diagnosis

In order to increase the reliability and system lifetime of digital microfluidic biochips, defect tolerance based on reconfiguration can be used to bypass faulty cells [14, 15]. We implement the diagnosis procedure using multi-step and adaptive Euler circuit-based testing methods. In each step, we divide the candidate faulty region into two partitions, and then test each partition to determine whether it is a candidate faulty region. Under single fault assumption [14], we can simply check either one binary partition to determine the faulty candidate region. By using a series of adaptive testing steps, we can eventually determine the location of candidate faulty cells. Assume that such a diagnosis procedure includes a series of testing steps, i.e., $T_1,\ T_2, \ldots, T_k$, where T_i $(i = 1 \sim k)$ denotes an Euler circuit-based traversal of the candidate faulty region at step i, and the final testing step T_k is to traverse a 2×2 array, i.e., the minimum candidate faulty region that can be located by Euler circuit-based approach. The number of steps k for a given microfluidic array size is given by using the following theorem.

Theorem 2. *To locate any single fault (including electrode-short faults) in an* $m \times n$ *microfluidic array* $(m,\ n > 2)$*, the number of Euler circuit-based testing steps* k *in the proposed diagnosis scheme is* $k = \lceil \log_2(m-1) \rceil + \lceil \log_2(n-1) \rceil$.

Proof. We can prove this theorem by using the two-phase partitioning schemes. In the first phase, we split the array in half with a cutting line in the Y-direction (North–South). The binary partition is recursively applied until each partition contains only one edge in the row of the corresponding sub-array. The number of steps in recursive binary partitioning is $\lceil \log_2(n-1) \rceil$. Next, a similar partitioning scheme is applied to the $m \times n$ array with a cutting line in the X-direction, until each partition only has one edge in the column; the number of binary partitioning steps is $\lceil \log_2(m-1) \rceil$ in this phase. Through these two phases, we are able to locate any single fault to a minimum candidate faulty region. The total number of partitioning steps is $\lceil \log_2(m-1) \rceil + \lceil \log_2(n-1) \rceil$, which is a sufficient number of adaptive testing steps to locate any single fault. Thus $k = \lceil \log_2(m-1) \rceil + \lceil \log_2(n-1) \rceil$. □

We denote the time needed for each testing step T_i by $Tt(T_i)$; it includes the Euler traversal time in the candidate faulty region described in Sect. 5.1,

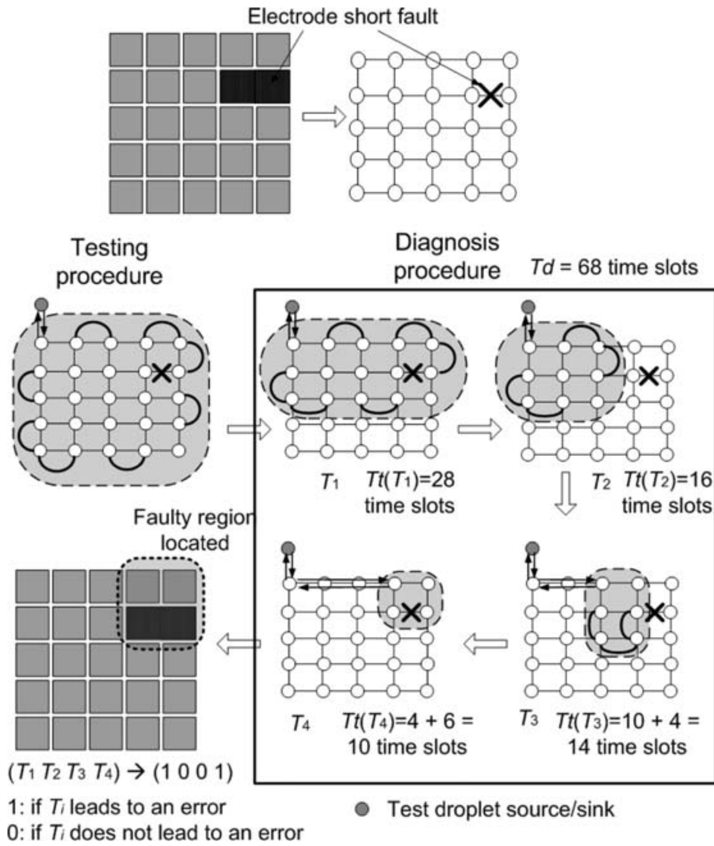


Fig. 14. An example of fault diagnosis for a 5×5 microfluidic array

and the droplet transportation time between the droplet source/sink and the testing region (if droplet source and sink are not adjacent to this testing region). Thus, the total diagnosis time Td is $Td = \sum_{i=1}^{k} Tt(T_i)$.

Figure 14 illustrates the adaptive diagnosis procedure for an array with an electrode-short fault. Based on the single fault assumption, we can easily locate the faulty region caused by the electrode-short fault through a series of testing steps, i.e., $T_1 \sim T_4$. If some bioassay operations are scheduled in this region, they must be remapped to other faulty-free regions on the microfluidic array to avoid erroneous assay results. This diagnosis method can locate not only single faults, but it can also easily be extended to locate multiple faults by using multiple test droplet sources and sinks.

Furthermore, a comprehensive testing and diagnosis procedure can be developed for digital microfluidic biochips. As digital microfluidic biochips become widespread in safety-critical biochemical applications, the reliability of these systems will emerge as a critical performance parameter. These

systems need to be tested adequately not only after fabrication (off-line), but also continuously during in-field operation (on-line). Once the testing procedure determines the faulty status of biochips, the operation of the normal bioassay is stopped. Diagnosis techniques are applied to determine the location of faulty region. Then reconfiguration techniques are applied to tolerate operational faults; the biochip is redesigned with the help of the proposed system-level design automation tools.

6 Real-Life Application

In this section, we use the real-life application example from [13], i.e., multiplexed glucose assay and lactate assay, to illustrate how Euler circuit-based method can be used for off-line testing, on-line testing and diagnosis in digital microfluidic biochips.

The glucose assay performed on digital microfluidic biochips is based on Trinder's reaction, a colorimetric enzyme-based method. In addition to glucose assays, the detection of other metabolites such as lactate, glutamate, and pyruvate using digital microfluidics has also been demonstrated recently [10, 21, 22]. Furthermore, all these assays can be integrated to form a set of multiplexed bioassays that are performed concurrently on a microfluidic platform. Figure 15 illustrates a fabricated microfluidic biochip prototype used for multiplexed bioassays [6]. For example, Sample 1 can be assayed for glucose using Reagent 1, which contains glucose oxidase and other chemicals. Similarly, Sample 2 can be assayed for lactate using Reagent 2, which consists of lactate oxidase and other chemicals. In this way, both glucose assay and lactate assay can be carried out concurrently. To demonstrate multiplexed assays, only cells and electrodes used for the bioassay have been fabricated.

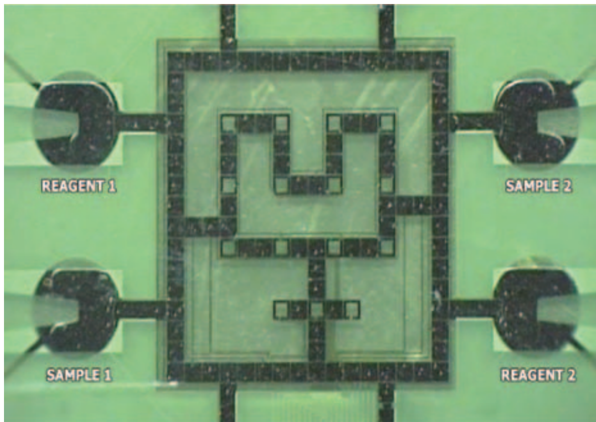


Fig. 15. Fabricated microfluidic array used for multiplexed bioassays [13]

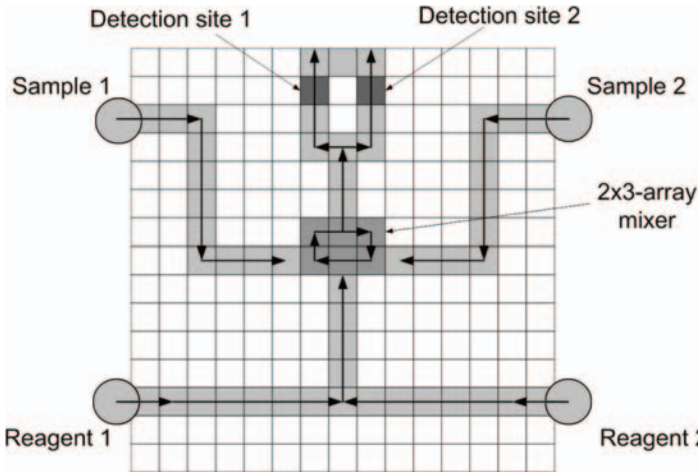


Fig. 16. A 15 × 15 microfluidic array used for multiplexed bioassays

In our example, the digital microfluidics-based biochip used for the multiplexed biochemical assay operations contains a 15 × 15 microfluidic array, as shown in Fig. 16. This example is a full-packed array design compared to the prototype shown in Fig. 15. The fabricated prototype chip can be embedded in this array; the latter is expected to be the next-generation prototype for demonstrating more complex bioassays. Note that, unlike previous work, we do not manually assign the location of test droplet sources and sinks here. Instead, the proposed PMF algorithm can be used to determine the optimal location of the test hardware. The schedule of the set of bioassays, determined using the techniques in [9], is listed in Table 2; one procedure of the multiplexed assays takes 25.8 s. The movement of droplets (including test droplets) is controlled using a 50 V actuation voltage with a switching frequency of 16 Hz. The details of these colorimetric enzymatic reactions as well as the fabricated prototype can be found in [13].

We first apply the PMF algorithm described in Sect. 5 to obtain an off-line testing plan for the 15 × 15 microfluidic array. Its eulerized graph model for a single test droplet is shown in Fig. 17a; next a test plan based on an Euler circuit is found using the PMF algorithm. The total testing time involves 448 time slots (i.e., 28 s), where the length of a time slot equals the droplet transportation time between two adjacent cells, i.e., 62.5 ms. The test droplet sources and sinks can be located at any boundary cell other than dispensing ports for sample and reagent droplets. Next, we consider on-line testing for this example. The optimized concurrent test plan obtained using the PMF algorithm takes 480 time slots (i.e., 30 s); compared to off-line testing, the test time is slightly higher due to the waiting time that is necessary to avoid conflicts with the normal bioassay. The optimal location for the test droplet source and sink is shown in Fig. 17a. The test plan for the same biochip in [13]

Table 2. Schedule of multiplexed biomedical assay (Sample 1 and Reagent 1 are for Glucose assay; Sample 2 and Reagent 2 are for Lactate assay)

Time (s)	Operation
0	Sample 2 and reagent 2 start to move towards the mixer.
0.8	Sample 2 and reagent 2 begin to mix together and turn around in the 2×3 array
6.0	(1) Sample1 and reagent 1 start to move towards the mixer. (2) Sample 2 and reagent 2 continue the mixing.
6.8	(1) Sample 2 and reagent 2 finish the mixing and product2 leave the mixer to optical detection location 2. (2) Sample 1 and reagent 1 begin to mix in 2×3 array mixer.
12.8	(1) Sample 1 and reagent 1 finish the mixing and product1 leave the mixer to the optical detection location 1. (2) Product 2 continues the absorbance detection.
19.8	(1) Product 2 finishes optical detection and leaves the array to the waste reservoir. (2) Product 1 continues the absorbance detection.
25.8	Product 1 finishes optical detection and leaves the array to the waste reservoir. One procedure of the multiplexed biomedical assay ends.

is only 18.7 s. Although the Euler circuit-based test plan requires more testing time, it provides higher defect coverage, since it can detect defects such as electrode shorts that affect two adjacent cells. For safety-critical applications, defect coverage is more important than a slight increase in the test application time.

We further consider the application of multiple test droplets for this example. If we partition 15×15 microfluidic array into two 8×15 arrays as shown in Fig. 17b, we can obtain an off-line test plan that allows two test droplets to traverse each partition while adhering to the constraints on droplet motion. The test application time for two test droplets is 238 time slots (i.e., 14.9 s), which is 47% less than that for a single test droplet. An optimized test plan for concurrent testing requires a total test time of 332 time slots, i.e., 20.8 s. Using the PMF algorithm, we find that the first partition requires 332 time slots for testing, while the second partition requires 308 time slots. The locations of two test droplet sources and sinks are also shown in Fig. 17b.

Finally, we apply the proposed diagnosis technique to this example. Assume that the cell used as the first optical detection site is shorted to its adjacent cell. Thus the product droplet of the glucose assay cannot be transported to the appropriate location for optical detection, thus leading to a measurement error. The adaptive diagnosis scheme proposed in Sect. 5.3 can be applied to locate faulty regions, as shown in Fig. 18. There are in all $(\lceil \log_2(15-1) \rceil + \lceil \log_2(15-1) \rceil)$, i.e., eight steps of adaptive testing procedures. Following the diagnosis procedure, we can reschedule the detection operation for the product of the glucose assay to another optical detector to avoid the error.

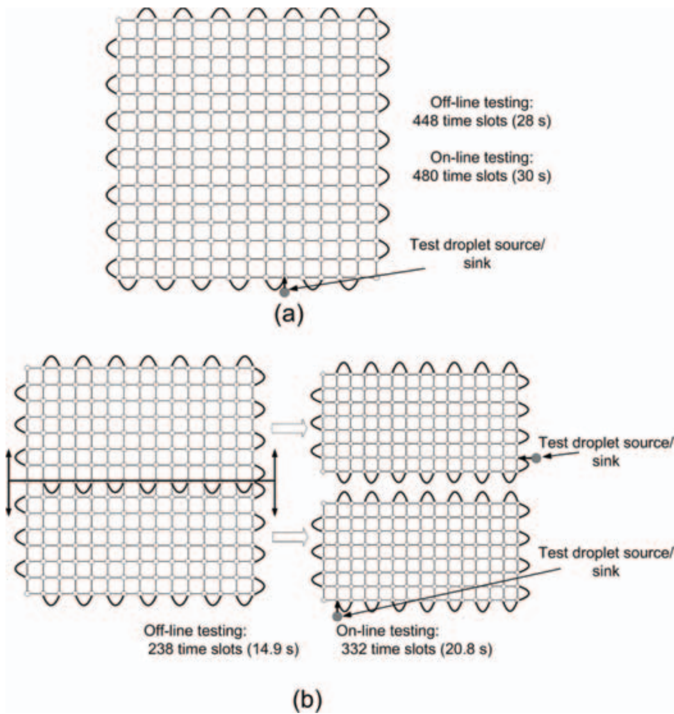


Fig. 17. Testing of a 15×15 microfluidic array: (**a**) eulerized graph for the application of the single test droplet; (**b**) partitions and eulerized graphs for the application of two test droplets

7 Conclusions

We have presented a defect-oriented testing and diagnosis methodology for digital microfluidics-based biochips. Experimental results have highlighted a major deficiency of prior work on the testing of microfluidic arrays; faults such as electrode shorts that affect two consecutive cells are not always detected by prior methods. To address this issue, we have formulated test planning in terms of the Euler circuit problem from graph theory. Both off-line and on-line testing methods have been presented. Diagnosis techniques to locate faulty cells in the microfluidic array have also been implemented using multi-step and adaptive Euler circuit-based testing procedures. The testing and diagnosis methods have been evaluated for a set of real-life bioassays. This work is expected to facilitate defect tolerance of digital microfluidics-based biochips, thereby increasing the reliability and system lifetime of these composite microsystems.

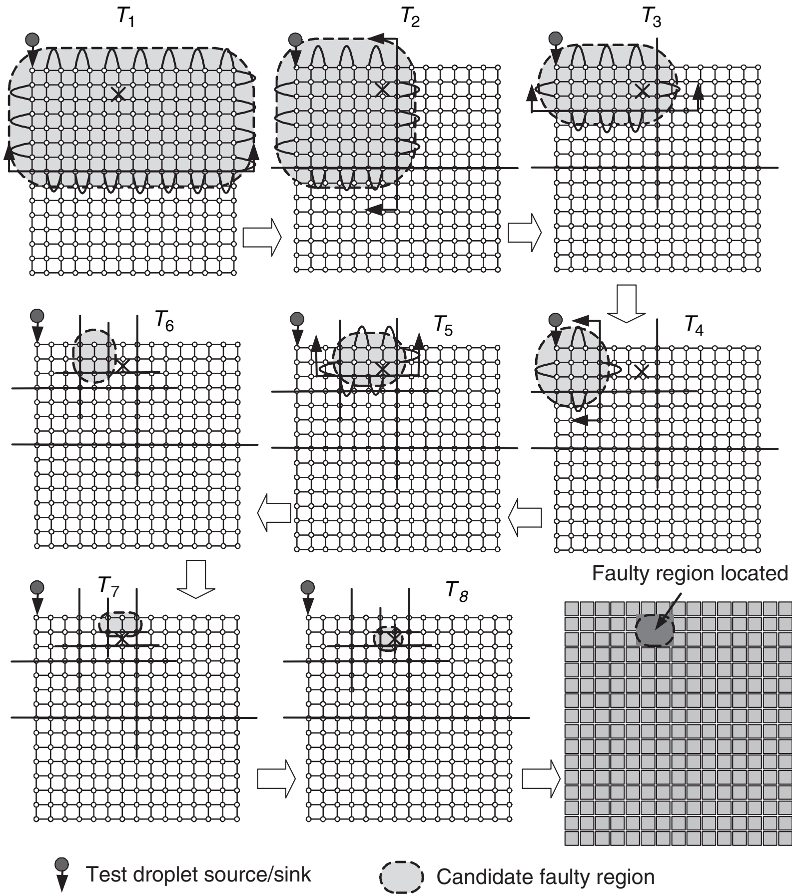


Fig. 18. Diagnosis procedure for a 15 × 15 microfluidic array

Acknowledgement

The authors thank Phil Paik of Duke University for help in carrying out the experiments involving electrode shorts.

References

1. A. Kolpekwar and R. D. Blanton, "Development of a MEMS testing methodology", Proc. IEEE Int. Test Conf., pp. 923–93, 1997.
2. N. Deb and R. D. Blanton, "Analysis of failure sources in surface-micromachined MEMS", Proc. IEEE Int. Test Conf., pp. 739–749, 2000.

3. N. Deb and R. D. Blanton, “Multi-modal built-in self-test for symmetric microsystems”, Proc. IEEE VLSI Test Symp., pp. 139–147, 2004.
4. S. Mir, B. Charlot and B. Courtois, “Extending fault-based testing to microelectromechanical Systems”, Journal of Electronic Testing: Theory and Applications, vol. 16, pp. 279–288, 2000.
5. A. Dhayni, S. Mir and L. Rufer, “MEMS built-in-self-test using MLS”, Proc. IEEE Eur. Test Symp., pp. 66–71, 2004.
6. International Technology Roadmap for Semiconductor (ITRS), http://public.itrs.net/Files/2003ITRS/Home2003.htm.
7. E. Verpoorte and N. F. De Rooij, “Microfluidics meets MEMS”, Proc. IEEE, vol. 91, pp. 930–953, 2003.
8. M. Pollack, A. D. Shenderov and R. B. Fair, “Electrowetting-based actuation of droplets for integrated microfluidics”, Lab on a Chip, vol. 2, pp. 96–101, 2002.
9. F. Su and K. Chakrabarty, “Architectural-level synthesis of digital microfluidics-based biochips”, Proc. IEEE Int. Conf. on CAD, pp. 223–228, 2004.
10. V. Srinivasan V. K. Pamula and R. B. Fair, “An integrated digital microfluidic lab-on-a-chip for clinical diagnostics on human physiological fluids”, Lab on a Chip, vol. 4, pp. 310–315, 2004.
11. F. Su, S. Ozev and K. Chakrabarty, “Testing of droplet-based microelectrofluidic systems”, Proc. IEEE Int. Test Conf., pp. 1192–1200, 2003.
12. F. Su, S. Ozev and K. Chakrabarty, “Test planning and test resource optimization for droplet-based microfluidic systems”, Proc. IEEE Eur. Test Sym., pp. 72–77, 2004.
13. F. Su, S. Ozev and K. Chakrabarty, “Concurrent testing of droplet-based microfluidic systems for multiplexed biomedical assays”, Proc. IEEE Int. Test Conf., pp. 883–892, 2004.
14. F. Su, K. Chakrabarty and V. K. Pamula, “Yield enhancement of digital microfluidics-based biochips using space redundancy and local reconfiguration”, accepted for publication in Proc. DATE Conference, 2005.
15. F. Su and K. Chakrabarty, “Defect tolerance for gracefully-degradable microfluidics-based biochips”, accepted for publication in Proc. IEEE VLSI Test Symp., 2005.
16. S. K. Tewksbury, “Challenges facing practical DFT for MEMS”, Proc. Defect and Tolerance in VLSI Systems, pp. 11–17, 2001.
17. H. G. Kerkhoff, “Testing philosophy behind the micro analysis system”, Proc. SPIE: Design, Test and Microfabrication of MEMS and MOEMS, vol. 3680, pp. 78–83, 1999.
18. H. G. Kerkhoff and H. P. A. Hendriks, “Fault modeling and fault simulation in mixed micro-fluidic microelectronic systems”, Journal of Electronic Testing: Theory and Applications, vol. 17, pp. 427–437, 2001.
19. H. G. Kerkhoff and M. Acar, “Testable design and testing of micro-electro-fluidic arrays”, Proc. IEEE VLSI Test Symp., pp. 403–409, 2003.
20. M. G. Pollack, “Electrowetting-Based Microactuation of Droplets for Digital Microfluidics”, PhD thesis, Duke University, 2001.
21. V. Srinivasan, V. K. Pamula, M. G. Pollack and R. B. Fair, “A digital microfluidic biosensor for multianalyte detection”, Proc. IEEE MEMS Conference, pp. 327–330, 2003.

22. V. Srinivasan, V. K. Pamula, M. G. Pollack and R. B. Fair, "Clinical diagnostics on human whole blood, plasma, serum, urine, saliva, sweat, and tears on a digital microfluidic platform", Proc. Micro Total Analysis Systems, pp. 1287–1290, 2003.
23. D. B. West, Introduction to Graph Theory, Prentice Hall, NJ, 1996.
24. T. H. Cormen, S. Clifford, C. E. Leiserson and R. L. Rivest, Introduction to Algorithm, MIT, 2001

Section 4: Reliability for Nanotechnology Devices

M. Tehranipoor

As devices and operating voltages are scaled down, future circuits will be plagued by higher soft error rates, reduced noise margins, increased circuit sensitivity, and defective devices. A key challenge for the future is retaining high reliability in the presence of faulty devices and noise. To further scaling CMOS technology, new device models and circuit design methods are much needed.

Transient and time-related faults also impose reliability concerns and must be taken into account in nanotechnology designs. Several methodologies have been reported recently on mapping computation onto chemically self-assembled defect-prone molecular nanofabrics. These methodologies use defect-mapping followed by defect-avoidance techniques. Most of these defect-mapping techniques can only generate defect maps for deterministic defect models and fail to alleviate the susceptibility of nanofabrics to transient faults.

This section contains three interesting chapters that cover various reliability challenges in current CMOS and future molecular electronics-based nanotechnologies.

Chapter 12, entitled "Designing Nanoscale Logic Circuits Based on Principles of Markov Random Fields", suggests that probabilistic computing would be one possible approach to future reliability problems. In this chapter the authors describe an approach for mapping circuits onto CMOS using principles of probabilistic computation. In particular, they demonstrate how Markov random field elements may be built in CMOS and used to design combinational circuits running at ultra low supply voltages. They show that with their new design strategy, circuits can operate in highly noisy conditions and provide superior noise immunity, at reduced power dissipation. If extended to more complex circuits, the proposed approach could lead to a paradigm shift in computing architecture without abandoning the dominant silicon CMOS technology.

"Towards Nanoelectronics Processor Architectures" is the second chapter in this section (Chap. 13) that focuses on reliability, one of the most fundamental and important challenges, in the nanoelectronics environment. For a

processor architecture based on the unreliable nanoelectronic devices, fault tolerance schemes are required so as to ensure the basic correctness of any computation. Since any fault tolerance approach demands redundancy either in the form of time or hardware, reliability needs to be considered in conjunction with the performance and hardware tradeoffs. The authors propose a new computational model for the nanoelectronics-based processor architectures, that provides flexible fault tolerance to deal with the high and time varying faults. The model guarantees the correctness of instruction executions, while dynamically balancing hardware and performance overheads. The correctness of every instruction is confirmed by multiple execution instances through a hybrid hardware-time redundancy approach. To achieve high system performance, multiple unconfirmed computation branches are exploited in a speculative manner. Hardware resource growth that these speculative computations entail is controlled so that the utilization of hardware is balanced between the two competing goals of performance and fault tolerance. In addition, the authors examine the impact on the proposed computational model of other nanoelectronic characteristics such as the necessity for localization of interconnections and the regularity of nanofabric structures on the proposed computational model. They set up an experimental framework to validate the effectiveness of the proposed scheme as well as to investigate multiple tradeoff points within the proposed approach. Simulation data confirm that the proposed computational model achieves the goal of providing flexible fault tolerance under a wide range of fault occurrence rates, while at the same time guaranteeing high system performance and efficient utilization of hardware resources.

Finally, Chap. 14, entitled “Design and Analysis of Fault-tolerant Molecular Computing Systems”, addresses the issue of defect-mapping by (1) developing a non-deterministic probabilistic defect map generation scheme that extends a fail-stop defect model-based defect-mapping technique proposed by Dwyer et al., (2) enhancing Jacome et al.’s hierarchical methodology to design structural redundancy-based architectures that can mask transient faults, and (3) developing a framework that integrates these techniques and provides system designers an environment for the design and analysis of molecular systems that are more tolerant towards permanent and transient faults. The authors have used this framework for generating defect maps for different molecular nanofabrics to evaluate the performance of their defect-mapping scheme and for designing and analyzing permanent and transient fault-tolerant signal and image processing computing systems.

Chapter 12: Designing Nanoscale Logic Circuits Based on Principles of Markov Random Fields

K. Nepal, R.I. Bahar, J. Mundy, W.R. Patterson, and A. Zaslavsky

1 Introduction

As Si CMOS devices are scaled down into the nanoscale regime, current microarchitecture approaches are reaching their practical limits. Thus far, the semiconductor industry has successfully overcome many hurdles, including the current transition to silicon-on-insulator (SOI) technology [1]. Looking to the future, the next major challenges to Si CMOS include new materials (high-κ and low-κ dielectrics [2]), new device geometries (dual-gate or fin-FET devices [3]), and further downscaling of devices and supply voltages with attendant difficulties in manufacturing, power dissipation, and economics of commodity manufacturing [2]. The longer-term prospects of digital computation then diverge into two interrelated areas. On the system side, there are the computer architecture issues arising from the problem of integrating billions of transistors at the lowest possible supply voltage, with tremendous constraints on total power dissipation and device reliability. On the device integration front, there is hope that hybrid systems will emerge, combining CMOS FET-based digital logic with any number of alternative devices, ranging from analog circuits, to more exotic alternatives (optical sources and detectors, quantum or molecular transistors, carbon nanotube devices, *etc.*) all on the same chip [4].

While there is no clear consensus on how far and how fast CMOS will downscale and which of the emerging hybrid technologies will eventually enter production, it is certain that future nanodevices will have high manufacturing defect rates. Further, it is clear that the supply voltage, V_{DD}, will be aggressively scaled down to reduce dynamic power dissipation – $V_{DD} = 0.5V$ is the current prediction for low-power CMOS in 2018 [5], although extrapolations to even lower $V_{DD} = 0.3V$ have appeared in the literature [4]. The resulting reduction in noise margins will expose computation to higher soft error rates.

Probabilistic computing provides a new approach towards building fault-tolerant nanoarchitectures and systems. We propose a new CMOS-compatible approach, based on principles of Markov Random Fields, to the design and operation of logic circuits. In this approach, the logic states are considered to

be random variables whose values can vary over the range of the logic signal level between 0 V and V_{DD}. Under this framework, one no longer expects a correct logic signal at all nodes at all times, but only that the joint probability distribution of signal values has the highest likelihood for valid logic states. The random logic variables for a circuit interact through a distribution representing their joint probability. Circuit design is guided by the formulation of a multivariate distribution on vectors of logic variables, aiming for a distribution that attains maximum probability for valid states of the circuit.

In this chapter, we describe how logic circuits may be designed using CMOS elements, based on principles of Markov Random Fields, such that correct logic operation may be obtained even under extremely noisy conditions. We show that with our new design strategy, the circuits provide superior noise immunity and at reduced power dissipation compared to standard CMOS counterparts.

2 Markov Random Fields: Theory

Before presenting our MRF style circuits, we first provide a brief overview of the Markov random field theory. Consider a set of random variables called *sites*, $X = \{x_1, x_2, \ldots, x_k\}$ where each variable, x_i can take on various values called *labels*. The sites in X are related to one another via a neighborhood system ($\mathcal{N}$) defined by a set of variables from $X - \{x_i\}$. This collection of random variables is called a *Markov Random Field* (MRF) if and only if:

$$P(x) > 0, \quad \forall x \in \mathbf{X} \quad (Positivity), \tag{1}$$

$$P(x_i|\{\mathbf{X} - x_i\}) = P(x_i|\mathcal{N}_i) \ (Markovianity). \tag{2}$$

In other words, a set of random variables form a MRF if all sites have a finite positive probability and the probability of a particular site in the neighborhood depends only on its immediate neighbors to which it is connected by an edge. The edges in the neighborhood represent the conditional dependence between the connected variables in the neighborhood. The joint probability of a given set of sites can be formulated in terms of the associated clique of the graph structure. Figure 1 shows one such neighborhood with one 1^{st}order clique and one 2nd order clique.

The Hammersley–Clifford theorem [6] asserts that X is a Markov random field on graph G if and only if it has a Gibbs distribution with respect to G. Using this theorem, the joint probability can be written as,

$$P(x) = \frac{1}{Z} \prod_{c \in C} e^{\frac{-U(x_c)}{kT}} \tag{3}$$

where X is the set of all nodes in the neighborhood, C is the set of cliques, x_c is the set of nodes in a clique c and $U(x_c)$ is the *clique energy function.*

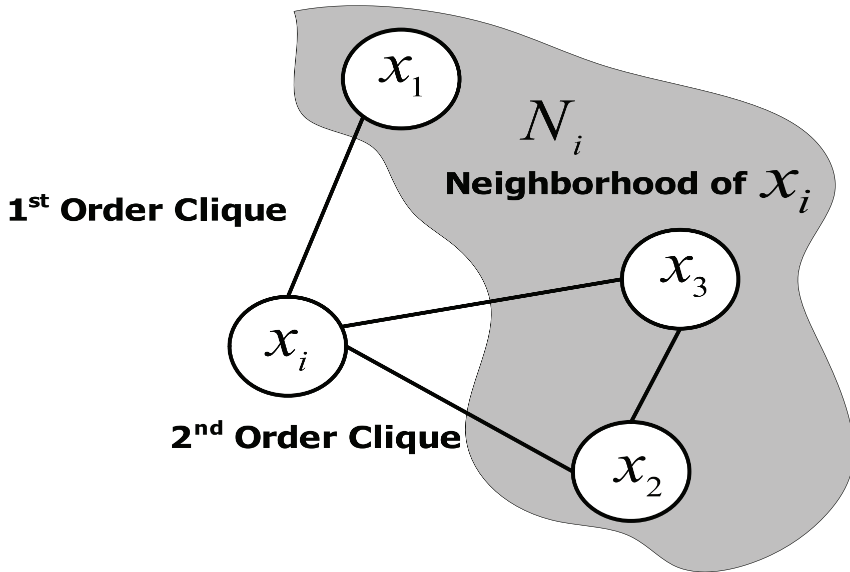


Fig. 1. The MRF neighborhood system

The term Z is called the *partition function* and is a constant required to normalize the probability function to [0,1]. The term kT can be interpreted as thermal energy from the physical point of view, but here it is merely treated as a constant in proportion to the clique energy that controls the sharpness of the probability distribution. This form of the distribution is called the Gibbs distribution. This Gibbs formulation of the Markov random field is an attractive representation for computation, since the physical interpretation of the probabilities in terms of entropy of computation is likely to find ready interpretation in the physical device characteristics.

3 Markov Random Fields and Circuit Networks

Circuit networks can be expressed in terms of neighborhoods shown in Fig. 1 and the interaction of the logic states and variables can be represented as a dependence graph. Figure 2 shows a simple multi-level circuit and its corresponding dependence graph. In this case, the graph is equivalent to a Markov random field, where the nodes are random logic variables that can hold values ranging from 0 V to V_{DD} and the edges are the conditional dependencies between the variables. Importantly, there is no notion of directed logic flow and causality, just statistical dependence. For instance, if the output of the first NAND gate is at logic 0, then both the inputs are constrained to be at logic 1 (i.e., there is a backward statistical dependency between the output state and the input states).

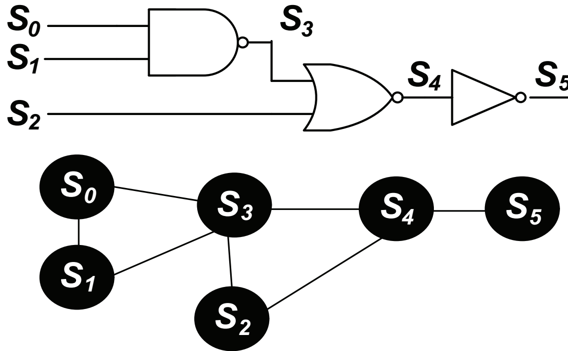


Fig. 2. A logic circuit and its dependence graph for a simple Markov random field

All the logic variables, $\{s_0, s_1, s_2, s_3, s_4, s_5\}$, in the example, are varying in a random manner over the range of the voltage levels. The correct logic states are those that maximize their joint probability, i.e., the correct logic operation for the example corresponds to the variables that maximize, $p(s_0, s_1, s_2, s_3, s_4, s_5)$. In large logic networks with hundreds of logic variables it is impractical to directly consider a joint probability distribution. The number of constraints required to enforce maximum probability for the valid states grows exponentially with the dimension of the random vector space and so the computation quickly becomes intractable. Fortunately there exists a representation for high dimensional joint distributions that can be factored into low dimensional distributions [7, 8]. The Hammersley–Clifford assertion that an MRF is equivalent to Gibbs distribution allows for this representation. Using this important relationship, the joint probability distribution $P(S)$ can be factored into terms each of which depends only on the variables covered by a set of cliques. In the graph of Fig. 2, three distinct sets of *cliques* (i.e., the sets of fully connected subsets of the nodes in the graph) $\{s_0, s_1, s_3\}$, $\{s_2, s_3, s_4\}$, $\{s_4, s_5\}$ are observed. These cliques represent the local statistical dependencies of the logic states. The joint probability $p(s_0, s_1, s_2, s_3, s_4, s_5)$ can now be decomposed into:

$$\begin{aligned} p(s_0, s_1, s_2, s_3, s_4, s_5) &= \prod_{c \in C} U_c(s_c) \\ &= U_{013}(s_0, s_1, s_3) U_{234}(s_2, s_3, s_4) U_{45}(s_4, s_5) \end{aligned} \quad (4)$$

This simple decomposition means that the maximum overall probability corresponds to the individual maximization of each clique function. This is crucial for probabilistic circuit design where the full set of logic variables (nodes) in the circuit can now be factored into a product of joint probabilities in the set of cliques that describe the local interactions.

4 State Propagation in MRF Networks

In this section we briefly describe the nature of computation in the MRF framework. The general algorithm for finding individual site labels that maximize the probability of the overall network is called Pearl's *belief propagation* [9] and provides an efficient means of solving inference problems by propagating marginal probabilities through the network.

In a Markov random network, we have two types of nodes - the **observable** nodes and the **hidden** nodes. The observable nodes (e.g., inputs to a logic circuit) have label probabilities pre-assigned to them by the problem setup. The probabilities of the hidden nodes (e.g., internal nodes and output nodes of a logic circuit) are not defined and need to be computed by the belief propagation algorithm.

The goal of belief propagation is to compute the marginal probability distribution $p(x_i)$ at each node i of the network. The probability of state labels at a given node in the network can be determined by marginalizing (summing) over the joint probabilities for the node state given just the probabilities for site labels in the Markov neighborhood, $\mathcal{N}_i$. This marginalization establishes the label probabilities for the next propagation step.

Belief propagation is an incremental/iterative process. Messages are passed from each node to its neighbors and the process repeats until a convergence is reached. If the Markov random network is acyclic (i.e., it contains no loops), it has been shown that the propagation algorithm converges to the maximum probability site label assignment for the entire network [10]. This incremental algorithm has computational complexity on the order of the number of nodes.

Most practical MRF networks are not acyclic and contain loops. In such networks, the marginalization must be done combinatorially over a region of the network that bounds the loops in order to guarantee maximum probability solutions. That is, one would partition the network into a loop-free network of blocks which internally contain loops. Yedidia *et al.* [10] showed that the belief propagation algorithm works well in graph with loops and usually converges to the maximum probability state even in presence of such loops.

Consider the circuit network from Fig. 2. For belief propagation, we start from the primary inputs of the circuit network and work our way to the outputs. As a first step, one of the input nodes with a known probability, (e.g., s_0) is taken and the node is marginalized out by summing the entire distribution over all possible states of that node. Then other input nodes are marginalized out. After all the input nodes have been eliminated, some of the hidden nodes now have marginal probabilities that are known. After a few iterations, the marginal probabilities are propagated to the output. Here, we outline the basic steps of the propagation algorithm in the network of Fig. 2 to determine the marginal probability $p(s_5)$ of the output node s_5:

Inputs: $f_0(s_0), f_1(s_1), f_3(s_2)$
Cliques: $f_2(s_0, s_1, s_3),\ f_4(s_2, s_3, s_4),\ f_5(s_4, s_5)$
Goal: $p(s_5)$

STEP 1: Eliminate s_0
Eliminated: $f_0(s_0),\ f_2(s_0, s_1, s_3)$
New: $f_6(s_1, s_3)$

$\Rightarrow p(s_5)=f_1(s_1)f_6(s_1, s_3)f_3(s_2)f_4(s_2, s_3, s_4)f_5(s_4, s_5)$

STEP 2: Eliminate s_1
Eliminated: $f_1(s_1),\ f_6(s_1, s_3)$
New: $f_7(s_3)$

$\Rightarrow p(s_5)=f_7(s_3)f_3(s_2)f_4(s_2, s_3, s_4)f_5(s_4, s_5)$

STEP 3: Eliminate s_2
Eliminated: $f_3(s_2),\ f_4(s_2, s_3, s_4)$
New: $f_8(s_3, s_4)$

$\Rightarrow p(s_5)=f_7(s_3)f_8(s_3, s_4)f_5(s_4, s_5)$

STEP 4: Eliminate s_3
Eliminated: $f_7(s_3),\ f_8(s_3, s_4)$
New: $f_9(s_4)$

$\Rightarrow p(s_5)=f_9(s_4)f_5(s_4, s_5)$

STEP 5: Eliminate s_4
Eliminated: $f_9(s_4),\ f_5(s_4, s_5)$
New: $f_{10}(s_5)$

$\Rightarrow p(s_5)=f_{10}(s_5)$

This example illustrates that achieving the correct state configuration in the network corresponds to propagating state values through the network and updating each node assignment with a node state having the maximum probability. Now we show how the mathematical framework of MRF can be related to logical elements suitable for computation.

The Markov random field framework can be represented by a number of models of computation such as the auto model, the multi-level logistic model, the smoothness prior model, the FRAME model and the hierarchical Gibbs random field model among many others [7]. These models help encode the clique energy function in the MRF framework. In [11] Bahar *et al.* used the **auto-model** to represent computation in Markov random networks. For combinational logic, this model, serves as a simple yet powerful representation of the clique energy function (logic compatibility function). In the auto-model, the clique energy function is represented as the summation of the interaction between different sites in a given clique.

$$U_c(s_i, s_j, s_k) = \xi + \sum_{i \in C_0} \alpha_i s_i + \sum_{\{i,j\} \in C_1} \beta_{i,j} s_i s_j + \sum_{\{i,j,k\} \in C_2} \gamma_{i,j,k} s_i s_j s_k \quad (5)$$

where the constant ξ is just an energy offset while α, β and γ are called *interaction coefficients*.

Boolean logic is represented in the MRF framework with nodes that interact and can take on labels from the set $\mathcal{L}=\{0,1\}$. A site s_i with label 1 (logic high) is written as s_i and a site with label 0 (logic low) is represented as s_i'[1]. Hence, if three interacting sites s_0, s_1 and s_2 have the site labels 1, 0, 1, respectively, the overall interaction between the sites is written as:

$$U_c(s_0, s_1, s_2) = s_0 s_1' s_2 \quad (6)$$

Consider the inverter from Fig. 2 with input s_4 and output s_5. Successful operation of this inverter is designated by the compatibility function $f(s_4, s_5)$ as shown in Table 1. This compatibility function or the clique energy function syntactically describes the interaction of the neighboring nodes represented in the circuit dependence graph.

Here we list all possible states (input/output pairs) : valid states with $f = 1$ and invalid states with $f = 0$. In [11] Bahar et al. proved that for a combinational logic circuit, the energy of correct logic state is always less than that of the invalid state by a constant. Hence, the valid input/output pair should have a lower energy than the invalid pairs. Thus, the clique energy expression is obtained by a negative sum over minterms from the valid states,

$$\begin{aligned} U_c(s_4, s_5) &= -\sum_i f_i(s_4, s_5) \\ &= -(s_4 s_5' + s_4' s_5) \end{aligned} \quad (7)$$

For the two valid states $\{01, 10\}$ of the inverter, the clique energy is -1 while for the two invalid states $\{00, 11\}$ the clique energy is 0. As long as the energy of the correct logic state configurations is less than that of the invalid state configurations, the logic element will operate correctly. Although the example of a two-state inverter may appear trivial, similar clique energy expressions may be written down for all elementary logic elements.

Table 1. The logic compatibility function for an inverter with all possible states

s_4	s_5	f
0	0	0
0	1	1
1	0	1
1	1	0

[1] In [11] Bahar *et al.* used Boolean ring to represent a site with label 0 (logic low) as $1 - s_i$.

5 Building MRF Elements in CMOS

As described in the previous section, the key underpinning of the MRF circuit behavior is that the logic states of network nodes need to depend, in a probabilistic fashion, on the logic states of some finite number of neighboring nodes. For the purposes of probabilistic computation based on interacting nanodevices, we need to find a physical embodiment of interacting logic levels and the clique energy function. In principle, these could be encoded in many physical variables, from occupation of quantum dots by single electrons with occupation probability of neighboring dots mutually influenced by their Coulomb repulsion [12], to the orientation of magnetic spins influencing each other via the exchange interaction. However in this chapter, we present the MRF computational paradigm in CMOS Si technology. By choosing the CMOS route, we can use proven device and circuit simulation techniques to more easily examine the higher-level architectural implications of probabilistic computing, including the power consumption, speed and fault tolerance of our circuits.

In the preceding section we discussed clique energy minimization to obtain correct circuit operation. This energy minimization can be achieved by a device or device configuration that produces a bi-stable energy function. A binary flip-flop circuit possesses this desired energy behavior where the required asymmetry of state energy is created by the summing mechanism just described. As such, the mapping of MRF model into CMOS circuitry requires the following two essential ingredients [13]:

- Each logic state, s_i, should be represented as a *bistable storage element*, taking on logical values of “0” an “1” with equal probability. The probability for any other signal value should be low.
- The constraints of each logic graph clique should be *enforced by feedback* to the appropriate storage elements, implementing the logic compatibility functions to maximize the joint probability of the correct logical values.

The first requirement ensures that the MRF logic states are maintained so that the conditional probabilities among the neighboring elements can propagate. The feedback paths, required by the second design principle, are based on conditional probabilities and ensure that the correct logic states are the most probable states. Each logic state, valid or invalid, has a probability distribution associated with it. In the absence of feedback the probability distribution of the circuit would be uniformly distributed between all states. Whereas the bistable element allows us to maintain a particular logic state at a given node, the feedback mechanism allows us to model the belief propagation and the dependence of a node on the state of its neighborhood, as described in Sect. 4.

5.1 MRF Inverter

For combinational circuits, this notion of feedback can be enforced by realizing the relationship between inputs and outputs of each gate or function.

For example, consider the inverter of Fig. 2 with input variable s_4 and output variable s_5 and logic compatibility function or clique energy described by (7). Following the recipe for mapping MRF networks into CMOS structures, we can create a bistable structure with feedback reinforcement that represents the clique energy function of the inverter relation using CMOS logic gates. For the mapping, we create a bi-stable element for each minterm of the logic compatibility equation – i.e., one bi-stable element for $s_4 s_5'$ and the other bi-stable element for $s_4' s_5$. The feedback for each node comes from the output of the bi-stable element containing its complemented counterpart. The MRF implementation of the inverter is shown in Fig. 3. The existence of two inverters in a chain between the bistable element and the input/output nodes might seem redundant. The cascade of inverters serves two purpose – firstly it allows us to reduce the capacitive load strain on the bistable output node and secondly it allows us to size the feedback paths to the input nodes and the output nodes differently. This sizing is crucial to make sure that the circuit does not stay at some metastable state or oscillate between the valid states. In our implementation, as illustrated in, the output feedback path is sized to have double the drive-current strength compared to the input feedback path.

The circuit consists of two "storage nodes", one for s_4 and one for s_5. The stable states of the nodes correspond to the maximum probability configurations of the variables. For example, suppose that $s_4 = 0$ and $s_5 = 1$. Then the top NAND-inverter gate is active and feeds the logic state "1" back to the inputs, thereby reinforcing the expected output value. The other NAND-inverter gate feeds back the logic "0" state. These feedback values are consistent with the input values $\{s_4, s_5\}$ and the overall circuit latches into this state. The other configuration, $s_4 = 1$ and $s_5 = 0$, corresponding to the other valid inverter logic state, is also stable.

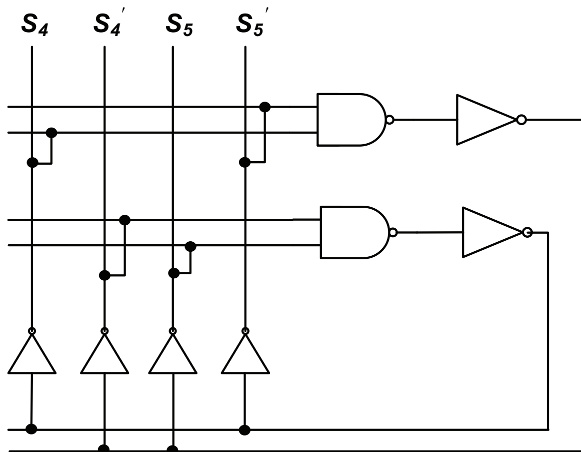


Fig. 3. A circuit for encoding the clique function of two logic variables defining an inverter

The MRF inverter and all of the more complex MRF gates and circuits described later in this chapter were simulated in SPICE using the 70 nm Berkeley predictive technology model [14] at $T = 100°C$. In order to simulate the aggressively scaled V_{DD} of future circuits, as well as noisy environment that will plague the end-of-the-roadmap CMOS, we operated our MRF gates at a supply voltage of 0.15 V – a voltage level below the threshold voltages of our transistor models, which are V_{TH}=0.2 and –0.22 V for NMOS and PMOS, respectively.

We ran two sets of simulations. First, we simulated the output of the circuit for a noisy input signal in comparison with the standard CMOS gates. Second, we simulated the effect of V_{TH} variation on the MRF element. We emphasize that the sources of signal noise in ultimate transistors are a subject of current research. Some noise sources, *e.g.*, hot-electron effects, cannot be treated analytically even for standard supply voltages but rather require Monte–Carlo techniques. On the basis of such simulations, some authors have argued that current noise models will underestimate noise levels in nanodevices [15]. Since we propose to run our circuits at very low V_{DD}, both thermal noise and hot-electron effects, as well as power supply and electromagnetic coupling noise will significantly degrade the logic voltages, while substantial and unavoidable V_{TH} variation [16] between transistors will reduce the noise margins.

An estimate of the noise on a typical signal arising from thermal noise aggravated by threshold variation can be obtained in SPICE by transient simulation of a chain of standard CMOS inverters. A sample of bandwidth-limited random noise of magnitude and spectrum determined from the steady-state noise of the Berkeley transistor model was added to the output of each of 10 inverter stages in tandem, with thresholds V_{TH} of individual transistors allowed a random variation of $\pm$ 10%. The resulting noise was roughly Gaussian with 30 mV RMS standard deviation. However, the Berkeley model deals with 70 nm planar bulk devices, whereas the future Si technology relies on fully depleted SOI with substantially lower node capacitances. Since noise is inversely proportional to the square root of the node capacitance [17], it is expected to be higher. In addition, our thermal model leaves out crosstalk noise, which will also have a significant effect. While research is underway in trying to accurately model the noise sources in nanoscale CMOS designs [18], we have added Gaussian noise of zero mean and 60 mV RMS standard deviation to our 0.15 V and zero voltage levels – a value we believe to be a reasonable estimate for the true signal noise seen by ultimate transistors operated at low V_{DD}.

With this choice of noisy input signals, we have compared the noise immunity for the MRF and CMOS inverters, initially assuming no V_{TH} variation. The inverters are compared in Fig. 4, where it is evident that the noisy input causes the standard CMOS inverter to switch between correct and incorrect output values, due to the small noise margin at low V_{DD} compared to the

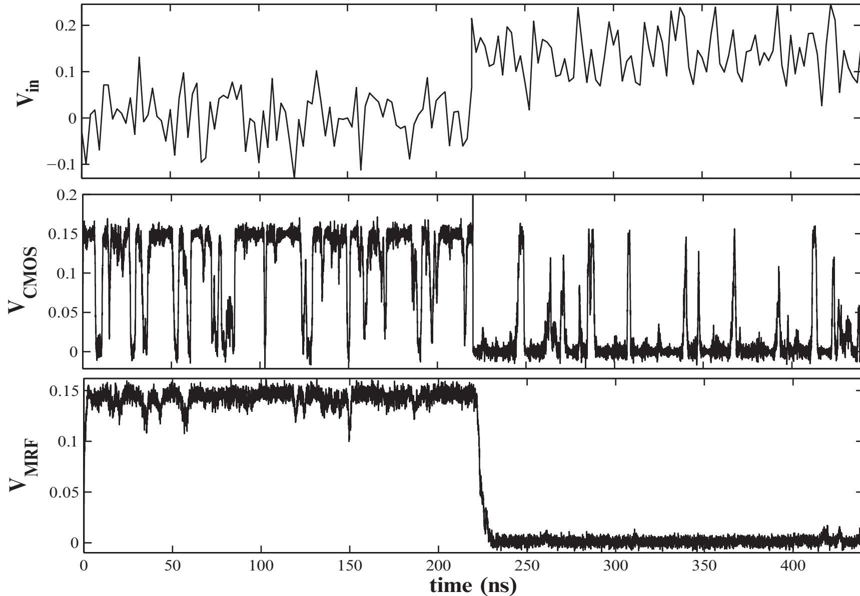


Fig. 4. Simulation of standard CMOS inverter and MRF inverter operation at subthreshold supply voltage

input noise amplitude. The MRF inverter, on the other hand, provides excellent noise immunity.

We emphasize that simulation illustrated in Fig. 4 assumed noisy input signals, without any V_{TH} variation from transistor to transistor (expected to reduce the noise margins in any large-scale circuit). The expected threshold voltage variation in ultimate CMOS transistors will depend on how the threshold is controlled. Current expectation is that they will have fully depleted undoped Fin-FET channels [2, 5] and V_{TH} will be controlled by the appropriate mid-gap gate material. In order to maintain effective gate control over the potential along the channel, the channel thickness W will need to be smaller than the gate length L_G, so $W < 10$ nm for ultimate CMOS devices. At the same time, W cannot be made too small because size quantization in the channel renders V_{TH} very sensitive to any variation in W [19, 20]. A monolayer fluctuation in W would lead to several mV variation in V_{TH}. As a result, in the following simulations we chose a worst-case ±10% (that is, ±20 mV) variation in V_{TH}.

Given the larger transistor counts, the immunity of the MRF inverter in Fig. 3 to V_{TH} variation is not self-evident, but our preliminary simulations, shown in Fig. 5 are reassuring. Figure 5 compares MRF inverter operation for $V_{TH} = 0.2$ and –0.22 V model values with the worst-case situation of ΔV_{TH} = 20 mV in all transistors but with N and P devices changing in opposite senses. In all cases, the MRF inverter operates correctly.

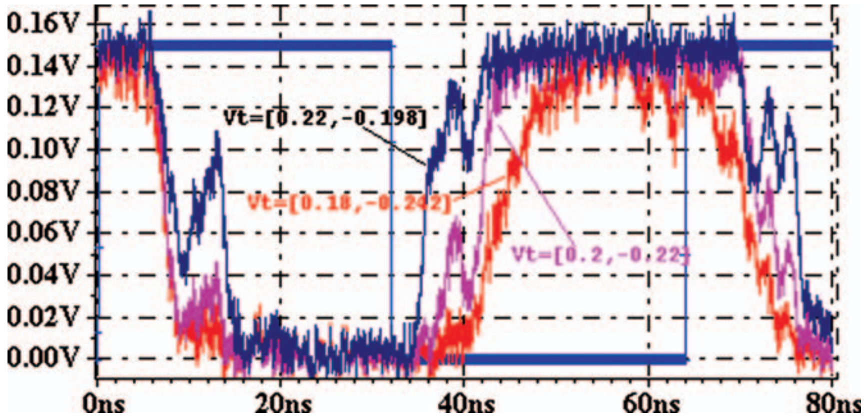


Fig. 5. MRF inverter with variable transistor V_{TH}. Comparison of V_{TH} values = 0.2 and –0.22 V (standard) with worst-case ± 20 mV variation (10% of V_{TH}) for all transistors

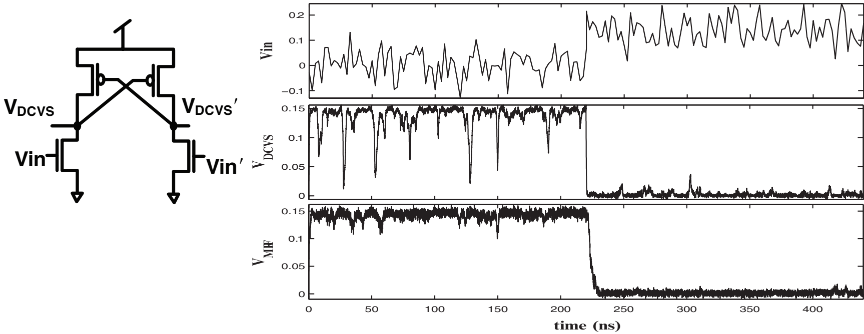


Fig. 6. Comparison of DCVS (top, with inset showing the DCVS transistor layout) and MRF inverters operated at VDD = 0.15 V, given noisy voltage inputs (Gaussian noise with zero mean and 60 mV RMS amplitude). Note that DCVS provides some noise immunity over standard CMOS, but not as much as MRF

The MRF implementations analogous to Fig. 3 provide correct probabilistic operation at low V_{DD} in the presence of noise that would ordinarily defeat standard CMOS. Nevertheless, it is instructive to compare this implementation with other implementations that also have noise-immunity characteristics. For example, consider a gate based on differential cascode voltage switch (DCVS) logic. By virtue of its differential operation and positive feedback, DCVS has some built-in noise immunity. Figure 6 compares the DCVS inverter (see inset for layout) to our MRF inverter of Fig. 3, in the presence of the same noisy input signals as in Fig. 4 (i.e., Gaussian voltage noise). We find that the DCVS inverter has much better noise immunity than a standard CMOS inverter, but is still not as stable as our MRF inverter. At the same time, a

DCVS inverter requires twice the transistor count of standard CMOS, while our MRF inverter is an order of magnitude higher.

5.2 MRF NAND Element

The layout of Fig. 3 suggests a programmable logic array style encoding where different functions can be achieved by varying feedback paths. Logic functions with more variables are implemented by feedback paths involving NAND/NOR gates with larger fan-in, and complex feedback elements. Here we use a 2-input NAND gate to illustrate the design of an MRF element with a three-node clique function.

Consider the truth table of a two-input NAND gate shown in Table 2. Again all valid states in the table are labeled with f=1. The clique energy function of this three-node gate can be obtained as:

$$U_c(x_0, x_1, x_2) = x_0' x_1' x_2 + x_0' x_1 x_2 + x_0 x_1' x_2 + x_0 x_1 x_2' \quad (8)$$

The clique energy function shows that there are a total of four minterms for the NAND2 element. Each minterm is a valid input–output pair whose probability must be be maximized using a bistable storage element. The feedback circuitry becomes slightly more complicated compared to the previous example. The feedback to x_2' comes from the first three minterms containing x_2, while the feedback to x_2 comes only from the final minterm containing x_2'. Since more than one minterm can determine the state of a logic variable, a more complex feedback network consisting of NOR logic gates are needed as shown in Fig. 7.

The circuit suggests that a bistable element is required for each minterm. If explicit enumeration of all valid input–output pairs were necessary, creating a MRF element with a larger fan-in would cause an explosion in the transistor count, severely limiting the applicability of this approach. Fortunately, an alternate mapping of the MRF elements described below provides better

Table 2. The logic compatibility table for a two-input NAND gate as a function of all possible input–output pairs

x_0	x_1	x_2	f
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

The inputs are x_0 and x_1, the output is x_2

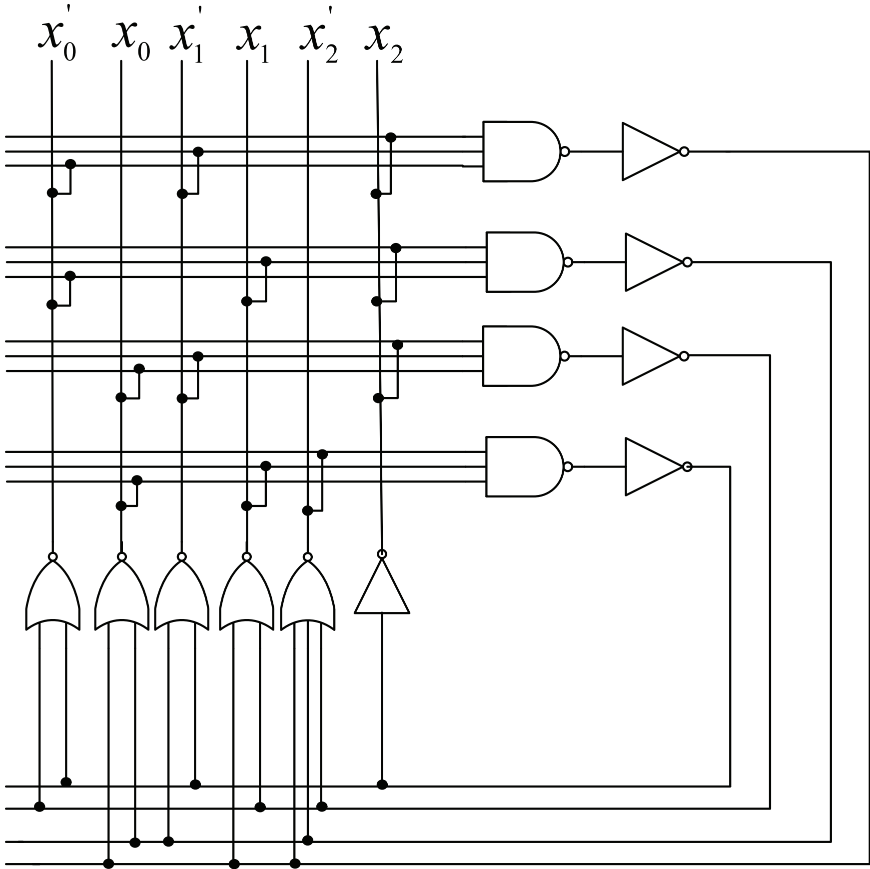


Fig. 7. Implementation of the MRF NAND2 element. The inputs are x_0 and x_1, the output is x_2

efficiency in terms of area and power and allows for creation of larger fan-in elements.

The clique energy function of (8) for the NAND gate can be re-expressed as:

$$U_c(x_0, x_1, x_2) = (x_0' + x_1')x_2 + x_0x_1x_2' \tag{9}$$

Using this factored form of (8), a more efficient mapping of the NAND gate can be created as shown in Fig. 8.

The new mapping consists of an OAI (OR-AND-INV) gate implementing the first term $(x_0' + x_1')x_2$ and a 3-input static CMOS NAND gate implementing the second term $x_0x_1x_2'$. The number of bistable elements required decreased from 4 *(for the four minterms)* to just 2. This decrease also reduced the complexity of the feedback path. In our earlier approach, the feedback to x_2' came from the output of a NOR gate whose inputs were three elements

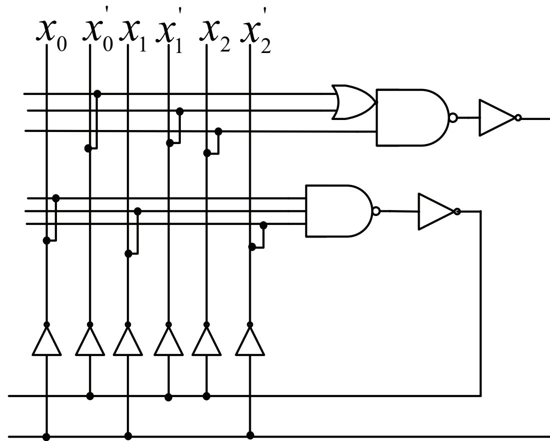


Fig. 8. Area efficient MRF NAND gate implementation (total transistor count is 28 compared to 60 of the mapping shown in Fig. 7). The inputs are x_0 and x_1, the output is x_2

representing the minterms containing the term x_2 (see (8)). This feedback complexity reduced from a three-input NOR (or its DeMorgan's equivalent) to a simple inverter that takes the output of the topmost complex gate and feeds back to x_2'. Similarly, the feedback to other nodes are also reduced. Mapping the simplified equation now produces a circuit that uses only 28 transistors, compared to the 60 transistors shown in Fig. 7.

The simulation of the optimized MRF NAND element of Fig. 8 and its comparison to standard CMOS when subjected to uncorrelated noisy inputs is shown in Fig. 9. As can be seen from the figure, the output of a regular static CMOS NAND gate is very noisy, rendering the gate unusable. However, the MRF NAND gate provides stable and correct voltage operation and excellent noise immunity.

It should be noted that, for all MRF elements, the presence of a feedback loop in the circuit can result in the circuit oscillating between valid states. This oscillation behavior (although not desirable in an electrical circuit) is consistent with the theory of belief propagation in a loop, where the loop is not guaranteed to settle in a particular state but can oscillate between valid states [21]. The feedback components must be sized properly to ensure that no oscillation or metastable states are possible at the supply voltage being used. In our circuits, all feedback to the circuit output are sized slightly larger to eliminate the possibility of any metastable states that might arise due to contention between the input and feedback.

Using this factorization technique, higher fan-in circuits can be created without exponentially increasing the circuit area and complexity. Table 3

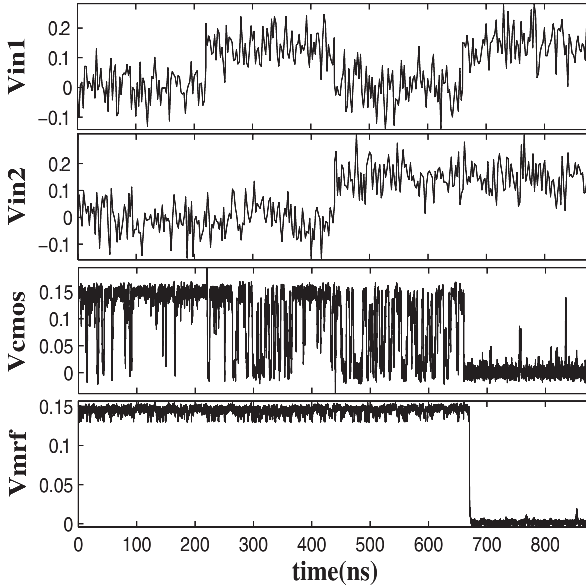


Fig. 9. Simulation of regular static CMOS NAND and optimized MRF NAND gate in presence of noise

Table 3. Comparison of transistor counts for multiple-input standard MRF elements

Std. gates	MRF mapping
1-input	20
2-input	28
3-input	36
4-input	44
5-input	48

shows the transistor counts for different gates (given as a function of its inputs) mapped into MRF elements.

The fan-in of the MRF elements is limited only by the maximum number of transistors connected in series in their transistor-level implementations. For instance, in the 2-input MRF NAND element shown in Fig. 8, a 3-high stack is required to implement the OAI gate within the element. In general, an N-input MRF element would require at most $(N + 1)$ transistors in series in the transistor-level implementation. While larger logic functions could be realized using higher transistor stacks, for practical purposes this is generally not preferred. When all the devices in the stack are turned on and conducting, the threshold voltage of each device effectively increases due to the stack effect and causes the drive current to decrease. This, coupled with the fact that all our circuits are operating at a subthreshold voltage regime prompted us to limit the maximum stack size of all our circuits to four transistors.

5.3 Circuits with Multiple Outputs

So far we have looked at simple logical elements. Often in real designs we encounter circuits that have multiple outputs. Usually these multiple outputs are all a function of the same primary inputs of the circuit. Consider a circuit with inputs p, q and outputs x and y. The output x is defined by the logical AND of the two inputs, i.e., $x = p.q$ and y is defined as $y = p + q'$. The clique energy function for these two relations can be written as:

$$U_c(p, q, x) = pqx + x'(q' + p') \tag{10}$$

$$U_c(p, q, y) = p'qy' + y(p + q') \tag{11}$$

The dependence graph in Fig. 10 shows the relationship between the inputs and the respective outputs. The solid lines in the graph show a dependence of the two separate outputs x, and y on the inputs. There is also a dashed line between outputs x and y. The dashed line between the two outputs represents an implied dependence between the two outputs. For example, the logical state of output x is directly dependent on inputs p and q. But the state of inputs p and q is also directly dependent on y. That means if output y was set to a 0 then inputs p and q would have to be logic 0 and 1, respectively. These two inputs being in those states means that x has to be at logic 0. This implied dependence between all the nodes of the dependence graph adds some degree of complexity but it also has an advantage. The main advantage here is that instead of treating these two outputs as two separate entities with two different clique functions, we can treat the entire system as one large entity governed by a fourth-order clique function consisting of nodes p, q, x and y.

$$U_c(p, q, x, y) = pqxy + p'qx'y' + q'x'y \tag{12}$$

Using this combined clique energy function, we can now create an MRF mapping for the circuit the same way as we did for the MRF inverter and NAND2 elements. The circuit encoding is shown in Fig. 11. The total number of transistor required to implement this combined clique energy function is 50 compared to 84 if the individual clique energies were separately mapped.

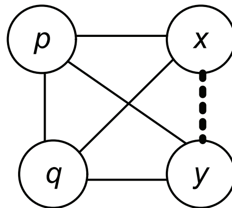


Fig. 10. A graph representing a circuit with multiple outputs. p, q are the inputs and x, y are the outputs

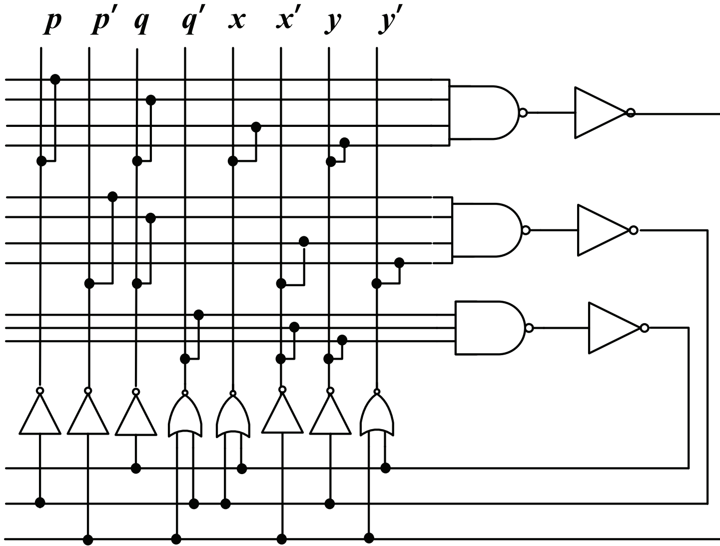


Fig. 11. MRF encoding of clique energy function shown in (12). p, q are the inputs and x, y are the outputs

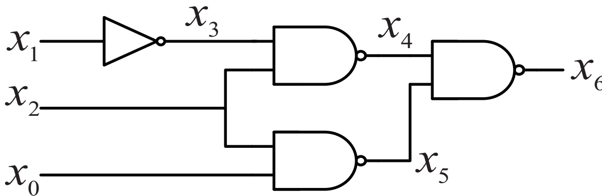


Fig. 12. A multi-level circuit

5.4 Building Larger Circuits

The MRF approach can be generalized to larger combinations of logic gates. Consider the circuit shown in Fig. 12 which implements the function $x_6 = x_2(x_0 + x_1')$. Larger multilevel circuits such as the one shown here can always be built by cascading MRF elements like the MRF inverter and the MRF NAND gate introduced earlier. The noise tolerance of the individual MRF element cascaded to form such multi-level elements will result in a reliable signal at the output x_6. However, the total cost of this reliability in terms of transistor count is 104, which is a large area penalty to pay for what is a 14 transistor circuit in regular static CMOS.

Instead of cascading individual MRF elements naively, one can take advantage of the fact that not all internal nodes are important. If the circuit

designer were interested only in the primary inputs and primary outputs of the circuit and did not care about the internal nodes, one could do away with cascading MRF gates. In the circuit shown in Fig. 12, the important nodes are just the inputs x_0, x_1, x_2 and the output x_6. As such, one can write the clique energy function directly for the circuit, ignoring all internal nodes, as: $U_c = x_6 x_2(x_0 + x'_1) + x'_6(x'_0 x_1 + x'_2)$. Implementing this clique energy function directly requires a total of only 36 transistors compared to 104.

While this decrease in transistor count is impressive, one cannot always ignore the internal nodes of a circuit. Consider a cascade of two inverters as shown in Fig. 13 for an application where the output of the first inverter as well as the second inverter is equally important. In such a case, one cannot ignore the middle node y. One possible solution is again to take just two MRF inverter elements and cascade them together. The total cost of this implementation would be 40 transistors. However, instead of cascading the two MRF inverter elements, we can again look at this circuit in terms of implied dependence. Signals x and y have an explicit dependence while y and z also have an explicit dependence. In such a case, the dependence of x and z is implicitly defined. Hence, an MRF encoding can be built by sharing the common label y as shown in Fig. 14. Here the total cost in terms of area is just 28 transistors.

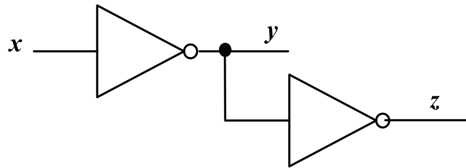


Fig. 13. An inverter cascade

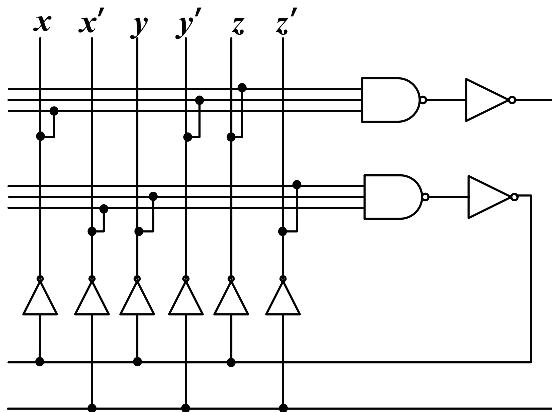


Fig. 14. Clique encoding of the inverter cascade

6 Power, Area and Delay Analysis

In this section, we compare the power dissipation and area overhead in terms of transistor counts of MCNC'91 combinational benchmark circuits mapped onto our optimized MRF elements operating at 150 mV and regular static CMOS gates running at 1 V (the expected V_{DD} for 70 nm technology) [5].

Table 4 shows the comparison between the standard CMOS and MRF implementations in terms of the number of transistors and power consumption under noisy conditions. Also shown in the table are the number of *first-stage* transistors (i.e., the number of transistors gated by primary inputs) and the maximum number of gates along any path from primary input to output (i.e., the *depth* of the circuit). For all our circuit simulations, because of the subthreshold operation of our circuits, we limit the maximum stack size to four transistors. Hence, the MRF approach used two and three input MRF elements when mapping the benchmark circuits to gates.

The table shows that the average increase in area for the MRF implementation is about 4.7 times the static CMOS implementation. At the same time, other redundancy approaches like Triple Modular Redundancy (TMR) are not that far lower (3X overhead for a fine-grained implementation). Regardless of whether a fine-grained or coarse-grained approach is used, the TMR method needs a reliable and a noise-free final majority gate. The MRF elements, however, have no such requirements. The effectiveness of the MRF approach compared to the TMR and the cascaded TMR approach was shown in [22]. In the presence of extreme noise and single-bit errors, the MRF approach produced correct output and showed superior immunity to noise compared to TMR.

The results in Table 4 also show that the MRF mapping provides a power advantage compared to static CMOS gates, particularly for circuits with larger depth and many transistors in the first stage. Specifically, the MRF implementation dissipates on average 33% less power than the standard CMOS implementation for these larger circuits (e.g., *alu4, cordic, ex5*, and *table5*).

Table 4. Comparison of transistor counts and power for MCNC'91 benchmark circuits

Circuit	in	out	CMOS V_{DD}=1V				MRF mapping	
			# tran	1st*-stage*	*depth*	*power(μW)*	*# tran*	*power(μW)*
5xp1	7	10	568	25	10	101.4	2,756	151.2
alu4	14	8	6,928	153	23	875.2	33,416	612.1
con1	7	2	78	6	6	16.5	356	16.9
cordic	23	2	604	32	15	89.8	2,612	54.7
ex5	8	63	5,448	135	13	692.5	25,964	506.9
misex1	8	7	356	11	7	69.6	1,700	82
o64	130	1	520	65	8	24.7	2,752	44.5
rd53	5	3	232	6	9	40.7	1,012	46.3
squar5	5	8	346	10	8	55.6	1,532	70.1
table5	17	15	10,192	237	23	1,522.5	47,948	936.1

Table 5. Delay of CMOS and MRF elements normalized to the delay of a CMOS inverter operated at $V_{DD} = 0.15V$

Circuit	Delay
CMOS INV	1.0
CMOS NAND2	1.6
MRF INV	7.1
MRF NAND2	8.6

This is significant, since this implies that our MRF elements may be used more effectively in larger circuit designs. For circuits with shallower depth, there is not as much flexibility available in the MRF mapping, so a power advantage may not always exist. In these cases, as a power/reliability tradeoff, it might be advantageous to evaluate the circuit areas most vulnerable to defects and noise, and selectively introduce MRF elements as needed to achieve desired reliability.

For sake of completeness, a quick glance at the propagation delay of the MRF elements is also provided. As the power supply is decreased into the subthreshold operation region, the propagation delay of a circuit increases significantly. The increase in delay for the MRF elements shown in the previous section is even more obvious because of the increased level of circuitry and the existence of feedback paths. The feedback paths add capacitance at the output node and the contention between the input and the feedback values causes an increase in the latency of the circuit. Table 5 shows the delay of the MRF elements normalized to a CMOS inverter operated at the same subthreshold voltage of 150 mV.

Depending on where and how these MRF elements are used, some path in the circuit may be able to tolerate the increase in delay. However, the problem will remain one with reliability, power and delay tradeoffs.

7 Quantifying Noise Immunity

In Sect. 5, we showed the noise tolerance of MRF elements compared to their CMOS counterparts. Here we quantify the circuit's tolerance to noise. An appropriate measure of the discrepancy between the actual output signal probability of a logical element or circuit P_{real} and the ideal (correct) output P_{ideal} is the Kullback–Leibler distance (KLD) [23]. For a digital system with two levels ("0" and "1"), the KLD is the measure of the distance between P_{real} and P_{ideal} (where output is sampled and noise leads to some probability of finding an incorrect output value):

$$KLD(P_{ideal}, P_{real}) = \sum_{states} P_{ideal} \log_2\left(\frac{P_{ideal}}{P_{real}}\right) \tag{13}$$

Table 6. Comparison of Kullback–Leibler distance from correct (noise-free) output of unloaded CMOS, DCVS, and MRF logic elements fed with noisy input voltages

	INV	NAND
CMOS	3.404	3.7947
DCVS	2.1832	3.6608
MRF	0.5878	0.4126

Table 7. Comparison of Kullback–Leibler distance from correct (noise-free) output of MRF and standard CMOS benchmark circuits (run at $V_{DD} = 0.15\,\mathrm{V}$, $T = 100°\mathrm{C}$)

Circuit	CMOS	MRF
5xp1	1.23	0.23
alu4	0.76	0.39
con1	1.03	0.20
cordic	0.60	0.33
ex5	1.18	0.50
misex1	1.00	0.24
o64	0.85	0.37
rd53	0.98	0.11
squar5	1.13	0.28
table5	0.90	0.34

where the smaller the KLD, the better the noise immunity of the circuit. By sampling the output voltage at discrete points we can quantitatively compare the noise immunity of our simple logic elements. For the KLD calculation the voltage values are sampled at 0.1 ns, because this time is much smaller compared to the switching time of our MRF elements. A comparison of standard CMOS, DCVS and MRF inverters and NAND gates is shown in Table 6. Clearly, the MRF implementations have much better noise immunity as measured by the KLD (for perfectly correct operation, the KLD is zero; see (13)).

We have also carried out the same noise immunity simulations for several larger benchmark circuits, each with two different implementations; one based on our MRF circuits and the other based on "standard" CMOS gates. The KLD values were computed by creating a probability distribution averaged over all primary outputs. As can be seen in Table 7, the KLDs for the MRF circuits are much smaller than those of the standard CMOS circuits, indicating that the probability distributions of the MRF gates more closely mimic the ideal output probability distributions.

8 Conclusions and Future Work

As devices are sized down to the nanoscale and supply voltage is scaled down below 0.5 V, circuit designs will need to account for significant signal noise in order to guarantee reliable computation. The MRF probabilistic model

provides a framework for designing CMOS circuits that can operate effectively under conditions of ultra-low supply voltage and extreme noise conditions. We have demonstrated that probabilistic computation based on MRF principles may be implemented in CMOS circuitry with much greater reliability in the presence of noisy inputs, but at a cost of larger transistor counts. The MRF circuits may be operated at much lower supply voltage, leading to reduced power dissipation along with improved reliability.

Our immediate goal in the future is to further reduce the area overhead to make the MRF design methodology more viable for large circuits. Our ultimate goal is to develop a noise-aware logic synthesis and technology mapping tool. Given a functional description of a circuit, the tool will produce an error-tolerant design that balances area, power, delay, and reliability constraints when generating the final mapped circuit. In addition, work is also underway in trying to accurately model the noise sources in nanoscale CMOS designs.

References

1. G. K. Celler and S. Cristoloveanu. Frontiers of silicon-on-insulator. *Journal of Applied Physics*, 93:4955–4978, 2003.
2. S. Luryi, J. M. Xu, and A. Zaslavsky, eds. *Future Trends in Microelectronics: The Nano, the Giga, and the Ultra.* New York: Wiley, 2004.
3. H. S. P. Wong. Beyond the conventional transistor. *IBM Journal of Research and Development*, 46(2-3):133–168, 2002.
4. H. Iwai. *The future of CMOS downscaling*, paper in: S. Luryi, J. M. Xu, and A. Zaslavsky, eds., *Future Trends in Microelectronics: The Nano, the Giga, and the Ultra*, pages 23–33. Wiley, New York, 2004.
5. International Technology Roadmap for Semiconductors. The latest update is at http://www.public.itrs.net.
6. J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society, Series B*, 36(3):192–236, 1994.
7. S. Z. Li. *Markov Random Field Modeling in Computer Vision.* Berlin Heidelberg Newyork: Springer, 1995.
8. R. Chellappa. *Markov Random Fields: Theory and Applications.* New York: Academic, 1993.
9. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* San Francisco, CA: Morgan Kaufmann Publishers, 1988.
10. J. Yedidia, W. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *International Joint Conference on AI*, 2001. Distinguished Lecture.
11. R. I. Bahar, J. Mundy, and J. Chen. A probabilistic-based design methodology for nanoscale computation. In *Proceedings of International Conference on Computer Aided Design*, November 2003.
12. K. K. Likharev. Single-electron devices and their applications. *Proceedings of the IEEE*, 87(4):606–632, April 1999.

13. K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky. Designing logic circuits for probabilistic computation in the presence of noise. In *Proceedings of Design Automation Conference*, June 2005.
14. Berkeley Predictive Technology Model. Available at http://www-device.eecs.berkeley.edu/∼ptm/.
15. V. M. Polyakov and F. Schwierz. Excessive noise in nanoscaled double-gate mosfets: A monte carlo study. *Journal of Semiconductor Science and Technology*, 19(4):145–147, 2004.
16. S. Narendra, V. De, S. Borkar, D. A. Antoniadis, and A. P. Chandrakasan. Full-chip subthreshold leakage power prediction and reduction techniques for sub-0.18 μm cmos. *IEEE Journal Of Solid-State Circuits*, 39:501–510, March 2004.
17. R. Sarpeshkar, T. Delbrueck, and C. A. Mead. White noise in mos transistors and resistors. *IEEE Circuits and Devices Magazine*, 6:23–29, November 1993.
18. H. Li, J. Mundy, W. R. Patterson, D. Kazazis, A. Zaslavsky, and R. I. Bahar. A model for soft errors in the subthreshold cmos inverter. In *Proceedings of Workshop on System Effects of Logic Soft Errors*, November 2006.
19. E. Suzuki, K. Ishii, S. Kanemaru, T. Maeda, T. Tsutsumi, T. Sekigawa, K. Nagai, and H. Hiroshima. Highly suppressed short-channel effects in ultrathin soi n-mosfets. *IEEE Transactions on Electron Devices*, 47(2):354–359, February 2000.
20. T. Ernst, S. Cristoloveanu, G. Ghibaudo, T. Ouisse, S. Horiguchi, Y. Ono, Y. Takahashi, and K. Murase. Ultimately thin double-gate soi mosfets. *IEEE Transactions on Electron Devices*, 50:830–838, March 2003.
21. K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: an empirical study. In *Proceedings of Uncertainty in AI*, 1999.
22. K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky. MRF Reinforcer: A Probabilistic Element for Space Redundancy in Nanoscale Circuits. *IEEE Micro*, 26(5):9–27, September–October, 2006.
23. S. Kullback. *Information Theory and Statistics.* New York: Dover, 1969.

Chapter 13: Towards Nanoelectronics Processor Architectures

W. Rao, A. Orailoglu, and R. Karri

1 Introduction

CMOS technology has moved beyond 80 nanometers in scale, and according to the *International Technology Roadmap for Semiconductors (ITRS)*, is projected to reach beyond 22 nanometers in the next several years [1, 2]. At nanometer scale, CMOS devices start to meet the physical limits and further shrinking in the CMOS feature sizes is checkmated by the insurmountable barriers of quantum effects, leakage current and power consumption.

A number of nanoelectronic devices, based on the quantum physical effects, have been proposed as highly promising to continue the scaling down of transistor sizes into the nanometer scale. Although no single nanoelectronic device can be expected to replace CMOS in the next generation, there are a number of promising device candidates, including Carbon Nanotube Electronics (CNT) [3], Resonant Tunnel Devices (RTD) [4], Quantum Cellular Automata (QCA) [5], Single-Electron Transistors (SET) [6, 7], Molecular devices [8–11] and Spin devices [12]. These nanoscale devices are promising due to their potential for high speed operation, low power consumption and device densities of the order 10^{12} device/cm^2 [1, 2]. Although these multiple candidates have different underlying characteristics, they do share a number of important characteristics, determined by their nanometer scale, that are significantly different from the ones exhibited by CMOS based devices [1, 2]. Particularly, unreliability, localized communication and regular structure of the nanofabrics, all essentially stem from the common nanoscale device dimensions:

1. *Fabrication.* Current CMOS systems utilize top-down fabrication, which is limited in obtaining precision when scaling down to the nano level. Aggressive approaches in lithographic fabrication do exist for the nanoelectronic environment, yet they are too expensive to be applied for massive production. On the other hand, a bottom-up approach is expected to prevail as the basic way to construct nanoscale circuits by building structures in a self-assembly manner. The main implications of

a bottom-up fabrication process are (1) *regularity in structures* imposed by the self-assembly process, (2) *massive defects* caused during the fabrication process, and (3) *post-fabrication reconfigurability* necessitated to define the circuits and bypass the defects.

It has been shown that the bottom-up self-assembly process of nanofabrics can be used to generate a crossbar-like structure, where a number of perpendicular nanowires form a grid with nanoelectronic devices located at the cross points [13–16]. Based on the crossbar structure, latch based storage elements and programmable logic array (PLA) like logic blocks have been investigated [14, 17, 18]. These approaches on the development of nanoelectronics based memory and logic, in conjunction with research work on the interface design between CMOS and nanodevices [17–20], have exhibited promising potential for the construction of functioning nanoelectronic systems.

2. *Interconnect.* Accessing the extremely small devices and delivering information at high speed and bandwidth is an essential challenge in the nanoelectronic environment. The small gain of the quantum devices strongly limits the number of fan-outs, while some devices, such as QCA, even rely on the interaction between neighboring devices to implement the transfer of signals. Interconnection essentially becomes the dominant issue in a nanoelectronic system in terms of area, delay, and power consumption. Communications between geographically distant devices through nanowires is extremely expensive and *localized interconnection* becomes a critical criterion in the nanoelectronic environment.
3. *Reliability.* Perhaps the most severe challenge as well as the most basic issue that needs to be addressed to build working nanoelectronic processor architecture is the reliability problem, which is commonly exhibited among all the emerging nanodevices [1,2,21,22]. Although the multiple nanodevice candidates exploit various underlying physical characteristics, the severe reliability challenge is caused in common by their highly shrinking device scale. In contrast to the current CMOS based systems, defect and fault tolerance is emerging as one of the most important design objectives in constructing any functional systems based on nanoelectronics. The unreliability of nanoelectronic devices exists mainly in two forms.
   - First, manufacturing defects increase significantly. Due to the device scale and the stochastic process involved in the bottom-up fabrication, the manufacturing defect rates in these emerging nanodevices are projected to be in the order of 10^{-3}–10^{-1}, in comparison to the defect rates of 10^{-9}–10^{-7} in CMOS technology [2, 21, 22].

     Related research in defect tolerance for the nanoelectronic systems includes system level approaches [23] and logic level solutions [24, 25]. These approaches reconfigure the redundant hardware to bypass the defective units according to a defect map in the post-fabrication stage.
   - Second, a high occurrence of transient faults is expected during runtime [2, 21, 26]. Due to the nanometer scale of devices which utilize

ultra low voltage, nanoelectronic transistors tend to be highly sensitive to environmental influences, such as temperature, cosmic ray particles, and background noise.

In fact, transient faults have been observed increasingly in the current CMOS based systems as the device scales down to the deep submicron stage. Single event upset caused by cosmic particles have already been observed in large amounts in the memory systems and sequential logic state elements. Similar transient faults have started to be observed in combinational logic as well [27–29]. The challenge of dynamic transient faults is furthermore severely aggravated in nanoelectronics based systems [21, 30]. The ultra low power utilized as well as the quantum effects nanoelectronic devices rely on, both result in significantly reduced noise margins and increased sensitivity to environmental effects, including cosmic particles, temperature, and crosstalk effects [2, 21, 26, 30]. In the nanoelectronic systems, transient faults are projected to be frequently occurring in not only memory elements, but also combinational logic blocks, since most of the logic functions are to be implemented by crossbar based PLAs and look-up tables.

In addition, the functionality of most nanoelectronic devices is extremely sensitive to a certain set of manufacturing parameters. However, the bottom-up manufacturing process cannot perfectly control such parameters to be precisely identical across all the devices fabricated, thus inevitably resulting in variations among the transistors. Such variations lead to significant differences in performance, robustness, as well as noise immunity among the devices in a chip, engendering clustered fault behavior in the system. Furthermore, such clustering behavior of the faults is exasperated by the factor that most environmental effects, including elevation in temperature generated by heavy computations, exhibit clustering and time varying behaviors as well.

The identification of a common set of new challenges among the multiple nanoelectronic devices provides an initial starting point to address the fundamental questions regarding the construction of nanoelectronic systems. Essentially, the highly shrunk dimensions and the severe unreliability in nanodevices introduce significant changes in the design optimization considerations of the current CMOS based systems.

Specifically, in nanoelectronic system design, on the one hand, the hardware constraint becomes less stringent due to the abundance of resources introduced by the shrinking of transistor sizes; on the other hand, reliability needs to be addressed as a fundamental issue, thus presenting an important new dimension into the design optimization space. The most fundamental challenge that has emerged in constructing a workable nanoelectronic system is to enable reliable computations despite the severe unreliability imposed by the underlying nanoelectronic devices. On top of the reliability challenge, a number of particular characteristics that are significantly distinct for nanoelectronics,

such as the strict interconnect constraint, the regularity in structure as a result of the bottom-up fabrication, the reconfigurability required to form essentially any operational functionality of the system, interacting with the fault tolerance aspect, will influence the design for the future nanoelectronic systems.

The emerging nanotechnologies are projected to offer a boosting of device density of the order 10^3–10^6 on the basis of current CMOS device scales [1]. This in turn can support a significant increase in the number of high-level computational units at the processor architecture level, which can be exploited for the dual competing goals of system performance and fault tolerance. Since processor architecture level fault tolerance in a nanoelectronic environment has direct impact on the performance and parallelism of the system, new computational models are required to support dynamically the tradeoff between hardware resources and system performance while efficiently guaranteeing the correctness of computations.

For nanoelectronic processor architectures, we focus on two core requirements that computational models should satisfy. First, correctness of computations is a fundamental requirement. The overall system should operate reliably even though the underlying nanodevices are highly unreliable. A second requirement is high performance. The large number of computation units should be efficiently organized to support system performance speedups. Overall, we need to understand (1) how can we translate the speedup afforded by nanoelectronic devices into system-level high performance; and (2) how can one organize the abundant computational resources to trade off fault tolerance against system performance in the presence of high rates of time varying faults.

We introduce a fault tolerance computational model according to the above considerations for the nanoelectronic processors. During the run time, each instruction is confirmed by multiple computation instances exploiting both hardware and time redundancy. On the performance aspect, through parallel speculative executions, the proposed computational model eliminates the severe performance deterioration typically caused by time redundancy approaches on data dependent instructions. On the hardware aspect, to avoid the exponential growth of resource allocation introduced by the hardware redundancy based speculations, a resource allocation framework is developed to control the hardware resource requirement, while preserving the low latency achieved through speculative executions. In addition to the fault tolerance, performance, and hardware resource tradeoff, we further provide a discussion on a number of other emerging issues in a nanoelectronic processor architecture, namely the localized interconnection constraint, CMOS/nano hybrid approach, and the decentralization over a regular structured nanofabric, based on the proposed computational model.

A simulation framework is setup to validate the effectiveness of the computational model and investigate the variations of the algorithm under different parameters. Simulation data confirm that, both in terms of hardware and

latency aspects, fault tolerance can be achieved by the new computational model under various fault rates, with high system performance and low hardware overhead. The simulation results also show that, for various fault rate ranges, a solution quite close to the optimal can be achieved through the proper selection of parameters in the proposed computational model.

2 Nanoelectronic Processor Fault Tolerance Overview

To tolerate the dynamically occurring faults at run-time, classic fault tolerance techniques such as N-Module Redundancy (NMR) and NAND multiplexing [31–34] have been examined and proposed for nanoelectronic systems. These hardware redundancy based techniques are typically applied at the logic gate level so as to mask the fault behavior through a majority vote. However, the reliance on applying these hardware redundancy based schemes at a low level in the design hierarchy is highly impractical, since it requires immense amount of hardware. It is shown that hardware redundancies of several orders are required to tolerate fault rates in the order of 10^{-2}. This is prohibitively expensive even for the resource abundant nanoelectronic environment [22, 30]. The dynamic faults caused by environmental factors exhibit clustered and time varying behavior. They exacerbate hardware replication based techniques, since the hardware redundancy approaches always need to aim at the worst case.

Consequently, the correctness of a nanoelectronic system based on the unreliable devices needs to be guaranteed by fault tolerance approaches at multiple levels in processor architecture. Various fault tolerance approaches, including hardware redundancy, time redundancy, and information redundancy can be exploited.

At the processor architecture level, execution of individual instructions is susceptible to runtime faults. In fact, the issue of fault tolerance for processor architectures has been addressed for CMOS based systems. In [35], the commit phase of the processor pipeline is augmented with a functional checker unit, which verifies the correctness of the core processor's computation, thus only permitting correct results to commit. In RAFT [36], fault tolerance in terms of diagnosis and recovery is achieved by duplicating the computation on different processors and comparing the signatures derived from their results. In [37], two modules are used to execute the same task, and compared for detection of faults. If a disagreement occurs, the two differing states are both stored. The state at the preceding checkpoint, where both processing modules agreed, is loaded into a third spare module to recompute the phase where faults had occurred. Concurrently, the task continues forward on the two active modules, beyond the checkpoint where the disagreement occurred. At the next checkpoint, the state of the spare is compared with the stored states of the two active modules and one which disagrees with the spare is identified to be faulty. When the faulty module is identified, the state of the faulty module

is restored to the correct state by copying the state from the other active module, which is fault-free. The spare is released to the pool after recovery is completed.

To address the problem of transient faults for multiprocessors, secondary versions of jobs are scheduled on the unused, or spare, processors of the system. Comparisons are used to detect faults [38, 39]. Redundant thread based recovery schemes have been proposed [40], in which fault tolerance is achieved by executing and comparing two copies of threads, namely leading and trailing threads, of a given application. For real-time multiprocessor systems, primary backup model is utilized, with two versions of the task executed serially on two different processors. An acceptance test is used to check the result, and the backup version is executed only if the output of the primary version fails the acceptance test [41]. Logic topologies of fault tolerant multiprocessor architecture have also been examined in [42].

Overall, for CMOS based processor architectures, fault tolerance is achieved by performing a computation with one or two extra redundant copies, either with hardware or time redundancy. This presumes a low and rather fixed rate of faults, which is true for CMOS based processors. Under limited hardware resources and low fault rates, these approaches are feasible. However, these schemes do not provide a satisfiable solution for the fault tolerant computation in nanoelectronic processor architectures. On the one hand, the significantly higher rates of dynamic faults cannot be handled efficiently based on the assumptions of low fault rates; on the other hand, the massive hardware resources offered by the nanoelectronic environment are not fully exploited. Due to the changes in the reliability challenge as well as the particularities of nanoelectronic environment, the fault tolerance issue in a nanoelectronic processor needs a rethinking from the perspectives of both component composition and the computational model.

In general, constructing a reliable processor architecture using highly unreliable components requires immense amounts of redundancy [22]. However, at the processor architecture level, multiple subsystems can be treated with various fault tolerance schemes. Furthermore, if a part of a system can be implemented with components of certain reliability guarantee, fault tolerance can be achieved with much lower overhead for the overall system. The backbone control of the system as well as the crucial components for fault tolerant purposes such as the majority voters, need to be implemented with components with higher reliability. Such a system construction is actually feasible, as CMOS and nanoelectronic hybrid structures are shown to be not only implementable, but also a necessary step towards the construction of nanoelectronics based systems. A number of research approaches have exhibited various ways to construct functional CMOS–nano hybrid electronic circuits, containing both logic and memory elements [17–20, 43]. Therefore, in a nanoelectronic system, a comparatively small number of reliable CMOS based components can be utilized to implement the critical components so as to construct efficient fault tolerance schemes.

At the processor architecture level, it is possible to exploit various forms of redundancies, such as hardware, time and information redundancy, and various subsystems can incorporate distinct fault tolerance schemes, since each strategy has distinct applicability according to the functionality of the subsystem. Typically the reliability of the memory and bus subsystem can be guaranteed by adopting information redundancy based schemes, since a number of well-developed error checking/correction coding theories are applicable in particular to these subsystems [44]. The reliability of the control subsystem is crucial for the functionality of the processor architecture as well as for guaranteeing the fault tolerance of other modules in a nanoelectronic processor architecture. We therefore envision the backbone control of a nanoelectronic processor architecture, as well as the control units for fault tolerance purpose to be implemented with reliable components such as CMOS devices, so as to carry out the fault tolerance schemes for the entire nanoelectronic processor.[1]

The arithmetic/logic computation subsystem of a nanoelectronic processor consists of a large number of ALU components, which perform the core task of instruction execution. These computation units exploit the parallelism supported by the large number of unreliable nanoelectronic devices, thus necessitating aggressive fault tolerance approaches to guarantee correctness. Information redundancy based error checking code approaches are hardly applicable for the arithmetic/logic computation subsystem since the computational components typically perform quite complex calculations, while the error checking code approaches rely on strict algebraic structures. Therefore, we focus our architecture-level fault tolerance approach on the faults occurring in the subsystem of computational components, with the assumption that control components can be implemented with reliable CMOS based devices, while data storage/transfer can be made reliable by applying error checking codes on memory and buses.

3 Motivation

According to the above analysis, applying hardware or time redundancy based fault tolerance schemes for the arithmetic/logic computation subsystem is the only choice. However, this is challenging due to the high cost in either hardware resource or performance. We motivate the proposed approach in this section by investigating a set of conflicts among hardware resources, system

[1] For performance reasons a number of control units close to the arithmetic units might be implemented with nanoelectronic devices. Special fault tolerance approaches need to be applied for these control units. In fact, a number of existing techniques can be used in enhancing the reliability of controllers and checkers [45]. For both FSM and RAM based controllers, coding based techniques have been developed to use redundant states or bits to provide error checking/correcting capabilities [45]. Additional coding based techniques such as the two-rail code are applicable for fault tolerant checker designs [45].

performance, and fault tolerance in a nanoelectronic architecture under high and time varying rates of faults.

Triple-Modular Redundancy (TMR) and in its generalized form, N-Modular Redundancy (NMR), have been some of the most commonly applied hardware redundancy based approaches for fault tolerance. To apply this straightforward strategy, an instruction can be computed by N distinct units in parallel and the result confirmed by a majority/plurality vote. This strategy is supported by the emerging nanotechnologies due to the abundant hardware resources.

A careful analysis reveals however that this approach is practical only if the fault rates are steady and the faults are evenly distributed, since the amount of redundancy is predefined and fixed. With high occurrence of time varying faults, however, a low predetermined number of computation units might generate distinct results and fail to confirm the computation. On the other hand, setting the redundant computation unit number high to match the worst case scenario consumes unnecessary hardware overhead. Therefore, the rigidity of the NMR fault tolerance strategy makes it extremely hard to match a predefined amount of redundancy with the high and varying fault occurrence in the nanoelectronic environment.

Consider a straightforward time redundancy strategy instead. Multiple clock cycles may elapse before the result of an instruction can be confirmed. For time redundancy based schemes, recompute with shifted operand (RESO) can be used to deal with dynamic permanent faults [46, 47]. However, the application of RESO is strictly limited to a small subset of functions, such as the addition operation. For general arithmetic/logic computation, a time-redundancy based scheme that reuses the same computation unit over multiple time slots is only effective for transient faults. If the component becomes permanently faulty or the transient fault lasts across multiple cycles, distinct computation units need to be allocated. Basically, both hardware and time redundancy are required in a complementary manner, so as to provide high flexibility. With time redundancy, an instruction is always confirmable despite the dynamically varying fault rate, since it can always allocate new computation units at the next cycle when the current results do not conform.

Severe compromises in system performance can be introduced by the time redundancy based approach if subsequent instructions need to wait and contend for a common centralized control unit, which performs both the instruction issue and the fault tolerance control task. Such a centralized control becomes the performance bottleneck in the system with time-redundancy based fault tolerance approaches. This problem can be resolved by introducing more parallelism. In fact, the control for fault tolerance purposes can be separated from the main architecture control unit, forming a second level distributed control for fault tolerance. Supported by the abundant hardware resource in nanoelectronic environment, multiple dedicated control units can be used in parallel, each performing a decentralized fault tolerance scheme for an instruction being executed. Consequently, the latency caused by the time

redundancy approach on one instruction does not block the next instruction from being issued. By adding a new layer of decentralized control units for fault tolerance purposes, instructions can be issued and dispatched without stalls.

Computations at the processor architecture level are not isolated. The existence of large number of dependencies among the instructions further causes complications in the performance overhead of time redundancy based approaches. The main performance bottleneck thus resides among the instructions with data dependencies. Basically, applying a time redundancy approach inevitably costs prolonged latency in the confirmation process. It might take an unpredictable number of cycles, as determined by the time varying fault occurrence, before an instruction can be confirmed. Therefore, any successor instructions that rely on the unconfirmed result of a current instruction have to be delayed, resulting in a domino effect on the subsequent dependent instructions and a tremendous number of stalls in an instruction pipeline. Consequently, the latency introduced by time redundancy becomes a severe problem when data dependency exists among instructions, especially in a pipelined environment.

To solve this problem, it can be observed that a dependent instruction need not wait for the confirmation of its predecessor results; additional units can be used to speculatively execute an instruction. In other words, a dependent instruction can use the unconfirmed results in a speculative manner. Multiple speculative branches may be formed for a dependent instruction. As results are confirmed, the correct branches of the dependent instruction are retained and the remaining branches are pruned. While speculation can speed up instruction execution in the presence of data dependencies, one has to carefully manage the amount of speculation. Speculative branches can grow exponentially and quickly exhaust the available hardware, furthermore compromising parallelism and performance in a processor system. We will present an allocation algorithm that carefully manages the speculative instruction execution by allocating hardware frugally.

4 Nanoelectronic Processor Computational Model

In a nutshell, the key features of a high-performance fault-tolerant computational model for the nanoelectronic processor architectures are:

- Decentralized fault tolerance control units (denoted as *voters*) with a large number of complex computation units (denoted as *C-units*)
- Fault tolerance scheme that utilizes hardware and time redundancy to guarantee the correctness of computations
- Support for speculative instruction execution of dependent instructions
- Hardware allocation algorithm that dynamically balances hardware resources and system performance when dealing with speculation branches for dependent instructions

4.1 Voter/C-unit Structure

In the nanoelectronic environment where the device densities can be 1–3 orders of magnitude higher than the current CMOS systems, a nanoelectronic processor can support a large number of computation units. Under such a scenario, parallel instruction improve system performance. However, the parallel instruction execution demands not only multiple execution units, but also decentralized control units to support fault tolerance to guarantee correctness of each instruction.

In the proposed architecture for a nanoelectronic processor, the system includes a pool of decentralized control units, denoted as *voters*. The control of the instruction is handled over to the voter once such a connection is established with the allocated voter. The centralized control performs the fetching, decoding and dispatching of the subsequent instructions in the instruction queue without delay. The voters support the parallel execution of the instruction and guarantee the correctness of the computations. Specifically, when an instruction is issued, a voter is allocated to it. The voter manages the reliable execution of the instruction by applying hybrid redundancy based fault tolerance approaches.

Computation units (*C-units*), which carry out the actual arithmetic and logic computations doing instruction execution, are managed by the voters. C-units are dynamically allocated by a voter, and receive from the voter the input values to perform a specific computation. Upon returning result back to the voter, the C-unit is released by the voter and is free for future allocations. In order to confirm the result of an instruction, a voter assigns the same computation to multiple C-units, and compares their results. Figure 1 shows a functional view of the instruction issue process. It can be seen that the Issue Machine constitutes the highest centralized control while the voters constitute the second level of decentralized control.

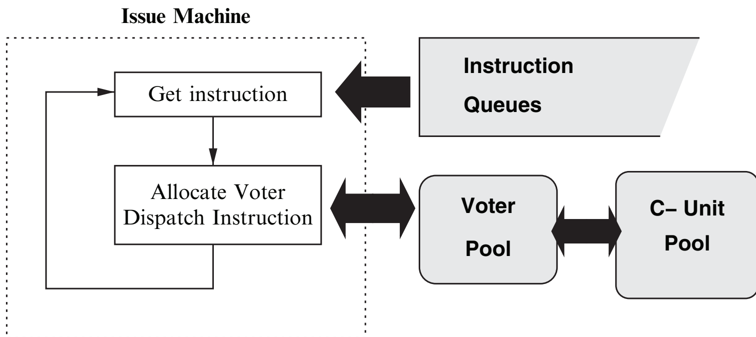


Fig. 1. Instruction issue process with voter/C-unit structure

4.2 Fault Tolerant Computation

In order to guarantee the correctness, an instruction should be confirmed by at least two results in agreement.[2] The fault tolerance computation is composed of two processes:

1. An initial NMR hardware redundancy approach, followed by
2. A time redundancy approach, which continues invoking new computation instances until two of the results conform

The main idea behind such a hybrid approach is two-folded (1) the hardware redundancy based NMR provides an initial trial to confirm the instruction, such that under the situation of low fault rate the computation can be quickly confirmed with minimum hardware resource required; (2) the time redundancy based approach provides full flexibility by trading off performance for reliability, so that high fault rates can be handled in a hardware-efficient way. In this section, we provide an exposition of the proposed scheme based on the minimum redundancy of two C-units.

Specifically, a voter accomplishes fault tolerance computation for each instruction by combining hardware and time redundancy in the following way. At the beginning, the voter allocates two C-units for an instruction for the initial hardware redundancy based NMR. Triple modular redundancy (TMR) is not necessary for this initial allocation, since the follow-on time redundancy can be invoked if an instruction cannot be confirmed with the initial C-units. After the execution, the unconfirmed values are stored by the voter while the C-units are released so as to support the computation requirement for other instructions. If the initial two results conform, then the instruction is deemed confirmed. Otherwise, the voter incrementally applies time redundancy by allocating one C-unit at a time until two of the results agree and the instruction can be confirmed. Similarly, each C-unit allocated during the time redundancy process is released once the result is stored back to the voter.

The voter performs a comparison each time a new result is returned. Since the previous stored results in a voter are all deemed distinct, the only possible agreement is between one of the previously stored results and the newly returned result. Therefore, only two-input comparators are needed in the voters and the number of comparisons performed inside a voter equals the number of existing results stored in the voter. Figure 2 depicts the comparisons needed to be performed upon the nth result being available.

Due to the involvement of time redundancy approach, the confirmation of an instruction might demand unpredictable latency. For a multi-cycle

[2] Note that in the computation of a processor architecture level the results typically consist of multiple bits, and it is presumed that faults in computation units exhibit themselves in distinct ways. Therefore, the faults occurring in multiple computation units tend to have negligible chances of generating a conforming yet faulty result. Further insurance against letting faulty computations slip through can be attained by increasing the threshold of agreement.

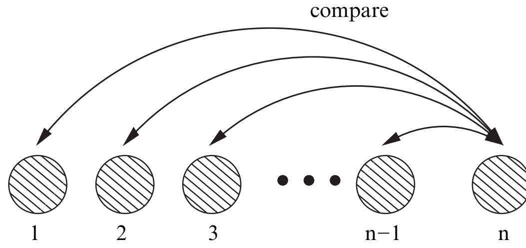


Fig. 2. Comparison performed in a voter when the nth result is returned

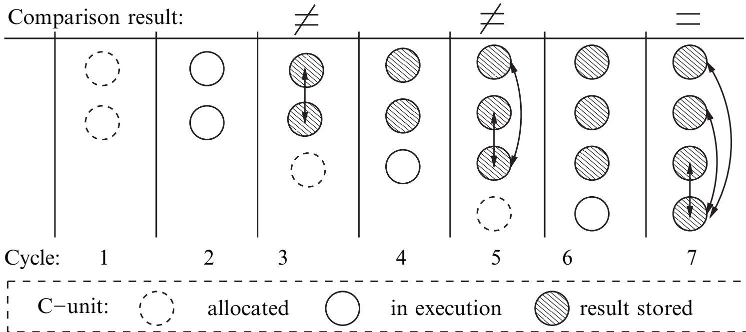


Fig. 3. An example of the instruction confirmation process

pipelined processor architecture, such a confirmation process can stride across multiple cycles. In implementing high performance processor architectures, pipelines of various depths have been utilized, varying from the simple five-stage pipelines in RISC architectures to the complex twenty-stage pipelines in Pentium 4 [48]. However, for the purpose of illustration, we make considerable simplification and divide an instruction confirmation process into three pipeline stages:

(a) Instruction decode and initial allocation of C-units by the voters
(b) Instruction execution carried out by C-units
(c) Result comparison and new C-unit allocation (if needed)

According to the above functionality division, the confirmation process of an instruction has a pattern of {a, b, c, (b, c)*}, which consists of a first initial allocation cycle (a), and followed by a number of recurrent (b, c) steps, due to possibly non-conforming comparisons.

Figure 3 shows an example of an instruction being confirmed in seven cycles with four C-units allocated throughout the process. In cycle 1 the voter initially allocates two C-units and the results are available in the third cycle. The two C-units are released at this point and the results stored in the voter. Since the results are not conforming, a new C-unit is allocated at the third cycle. After the execution stage in the 4th cycle, the new result is available

at the 5th cycle and is compared with the two stored results. Again the comparisons cannot achieve an confirmation, thereby a new C-unit is allocated in cycle 5. The instruction is finally confirmed at cycle 7 when one of the three previous stored results conforms with the newly returned result.

To carry out the fault tolerance approach, a voter needs a number of storage elements and comparators. Although multiple values need to be compared to check for possible conformity between two results, the comparisons are always performed pairwise, since they are only required between the returning result and the existing ones. Therefore, these comparisons can be performed in parallel with a number of two-input comparators, avoiding the necessity of multi-input comparators, which are expensive both in terms of hardware and latency. Furthermore, a tradeoff needs to be examined when designing the voters: a voter can utilize a large number of comparators for fast parallel comparisons; alternatively, a voter can perform the comparisons in serial with shared comparators, which might result in a prolonged multi-cycle comparison process in the pipelined environment.

The storage elements in a voter is for the purpose of keeping the non-conformable results. The number of storage elements set in a voter depends on the tradeoff between hardware and fault tolerance requirement. At the rare occasions where fault rate is extremely high and all the storage elements are used up in a voter, a simple strategy of discarding a fraction of the existing results can be used. Since all the results stored are unconfirmed and distinct, the chance of discarding a correct result is very low.

4.3 Speculative Computation to Improve Performance

We have discussed the fault tolerance computation scheme, which confirms each instruction through a hybrid redundancy of hardware and time. The main concern for system performance, consequently, hinges on the data dependencies on the instructions yet to confirm. Consider the situation where an instruction B takes as input the result of a yet unconfirmed instruction A (we denoted B as a *child instruction*, and A as a *parent instruction*), since the execution of B needs to be based on a correct input, B's execution is delayed until the confirmation of A's result.

Alternatively, in order to improve system performance, dependent instructions do not need to stall for the confirmation of their operands. A child instruction can speculatively use the results of the yet unconfirmed parent instruction. As a confirmed result of the parent instruction is obtained at a later cycle, it can be used to confirm or prune the corresponding speculative branches of the child instruction.

As an example, suppose instruction B uses the result of instruction A as an input, while A takes an exceedingly long time to confirm. B can start executing speculatively without delay, based on the multiple unconfirmed results of A. Multiple *speculative branches* for computing B can thus be formed. As the result of A is finally confirmed, the correct branches of B are retained and

the incorrect speculative branches are pruned. The conforming results within a correct branch of B's execution can further confirm the instruction B.

According to the above analysis, two extremal positions in terms of hardware vs. latency tradeoff can be envisioned.

- In a *no speculation* approach, if the input data of a child instruction depends on a parent instruction that is not yet confirmed, the execution of the child instruction is delayed and waits for the confirmed input from the parent instruction, thus necessitating no extra computation units for speculations.
  The *no speculation* approach is a simple scheme that uses a small constant number of C-units, yet inevitably results in significant delay among the dependent instructions. Particularly, in the case of a sequence of dependent instructions, the delay caused by the time redundancy approach is transferred to all the descendent instructions, resulting in a domino effect of latency accumulation. Consequently, the *no speculation* approach represents a mechanism with low hardware, yet high latency overhead.
- In a *full speculation* approach, an instruction can start execution by generating multiple speculative branches for every unconfirmed input. Since the initial NMR needs to be applied in each speculative branch for the purpose of local confirmation,[3] the formation of full speculative branches necessitates at least twice the number of the hardware resources to compute all the branches of a child instruction in parallel. Consequently, the *full speculation* approach represents a mechanism with low latency, yet high hardware overhead.

Figure 4 illustrates the *no speculation* and the *full speculation* approaches. When executing dependent instructions with the *no speculation* case as shown

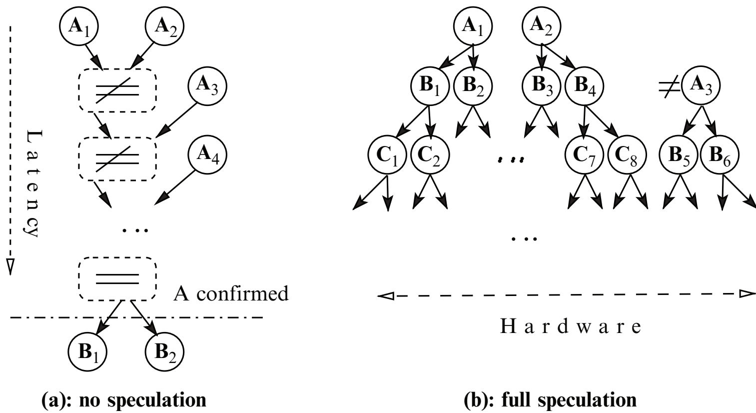


Fig. 4. An example for the cases of (**a**) no speculation and (**b**) full speculation

[3] We assume each speculative branch always allocates the minimum number, i.e., two, redundant computations for local confirmation purposes.

in Fig. 4a, the latency can be arbitrarily long; in the case of *full speculation* as shown in Fig. 4b, an exponential growth in the hardware requirements is encountered.

Basically, while speculation can speed up instruction execution in the presence of data dependencies, speculative branches can grow exponentially and exhaust rapidly even the abundant hardware available in a nanoelectronic environment. To avoid the severe latency problem in the *no speculation* approach and the exponential growth of hardware allocation in the *full speculation* approach, a hardware allocation framework needs to be developed to achieve both frugal hardware resource allocation and short overall latency. The essence of such a resource allocation mechanism is to control the generation of speculative branches, such that hardware resources are allocated on an as-necessary basis. Specifically, extra hardware is only allocated when an instruction is known definitely not confirmable.

4.4 Dynamic Hardware Allocation Algorithm

For a child instruction that depends on the unconfirmed input data from its parent instruction, the correctness of the result relies on:

1. The correctness of the input data, and
2. The confirmation of the computation process within the child instruction

The challenge of performing speculative execution essentially comes from these two points.

First of all, a parent instruction might have multiple unconfirmed results, thus forming multiple speculation branches for a child instruction. These speculation branches are dynamically changing. According to the time redundancy approach of the parent instruction, new unconfirmed results might emerge and form additional branches. Furthermore, the confirmation of the parent instruction will cause merging of correct branches and pruning of incorrect ones.

Secondly, within the child instruction itself, the hardware and time redundancy approach applied to confirm the computation for each speculation branch is dynamically changing: two C-units are allocated initially for each branch and additional ones are added in later cycles if necessary.

Overall, these two means of guaranteeing the correctness of a child instruction, when combined, result in an exponential hardware growth in the full speculation approach. The underlying reason for the exponential growth of hardware resources needed in the full speculation approach is that it does not differentiate the hardware allocation policy in a parent instruction and a child instruction. In a sequence of dependent instructions, the exponential growth in hardware resources occurs when a speculation tree is formed with the branch number doubling with depth. The most significant part of the hardware is therefore spent on extensive speculations, a large portion of which might turn out to be based on faulty input data.

From the above analysis, we can draw the conclusion that, hardware allocation should be concentrated on the root-level parent instruction, where the confirmation will accelerate the further confirmation of the descendent instructions. In other words, the C-unit allocation in the child instructions should be controlled so as to reduce the hardware resources spent in the speculative execution. This is the core idea that enables reductions in the exponential hardware growth to be otherwise expected.

An ideal C-unit allocation algorithm for the child instructions needs to achieve the goals of frugal hardware resource allocation, quick confirmation with low latency and fault tolerance for a computation. Specifically, the following aspects should be addressed:

- Parent instructions should be provided with higher priority for obtaining hardware resources in order to quickly prune out wrong speculative branches.
- Child instructions should be updated with the states of the parent instructions during the fault tolerant computation, so as to effectively control the speculation branches.
- The initial C-unit allocation for a child instruction should try to preserve the confirmation possibility of the instruction, yet with minimum hardware resources.
- For a child instruction based on unconfirmed input, hardware allocation should be frugal yet maintain the possibilities for confirmation. In other words, for fault tolerance of the speculative branches, hardware is only added when the instruction becomes impossible to confirm.

Essentially, when the fault rate is low, the initial speculative branches suffice for the child instruction to quickly confirm, without consuming large amounts of hardware resources. When the fault rate is high, the speculation branches should be controlled to a limited number to avoid an exponential growth of hardware requirement. Hardware resources need to be highly prioritized, in this case, on the root level of the speculation tree, thus guaranteeing the quick confirmation of the parent instructions.

We explore the hardware allocation algorithm in depth in the following three subsections. Basically, we discuss the initial C-unit allocation of the dependent instructions, how the parent instruction level information is utilized for the child instructions and how the hardware growth in the dependent instructions is managed. The description of the proposed technique is followed by an example and a discussion subsection.

Initial C-Unit Allocation of Dependent Instructions

The minimum hardware resources needed to confirm an instruction in the shortest time are two C-units. For a child instruction based on an unconfirmed input, the full speculation approach allocates likewise two initial C-units for each unconfirmed input. This eventually leads to an exponential growth in

hardware requirement for a sequence of dependent instructions. In fact, two of the results in the parent instruction will turn out to conform, yet the initial allocation of the child instruction in the full speculation approach presumes every unconfirmed result to be distinct, thus generating a number of redundant speculative branches.

When a child instruction is issued, information from the previous comparisons in the parent instruction should be utilized. Obviously, an input with the *full resolution* of information, i.e., known to be *correct* or *incorrect*, is easy to deal with. However, for most of the unconfirmed inputs, at a particular cycle, such full resolution is not achievable. However, even if an input is not under full resolution, information can be extracted by distinguishing between the following two states:

- *Distinct.* The unconfirmed result is known to be distinct among all the other unconfirmed results.
- *Conformable.* The unconfirmed result has a *comparison companion*, i.e., is to be compared with another unconfirmed result. In this case there is a possibility that the unconfirmed result might conform with the comparison companion's result.

In the example shown in Fig. 3, at the 3rd cycle and the 5th cycle, the stored results, represented by the shadowed circles, are all in the *distinct* state, since the newest comparison indicates they differ from each other. On the other hand, at the fourth and sixth cycle, the stored results are all in the *conformable* state, since they are all to be compared with the new result in execution.

Figure 5 shows the two specific situations as well as the corresponding initial C-unit allocation cases for a child instruction. In the case shown on the left side, all the results in the parent instruction are known to be *distinct* since a comparison has just completed and it turns out no conformity is achieved. For the child instruction, two C-units are initialized for each distinct result, forming multiple speculation branches that can be locally confirmed. There is

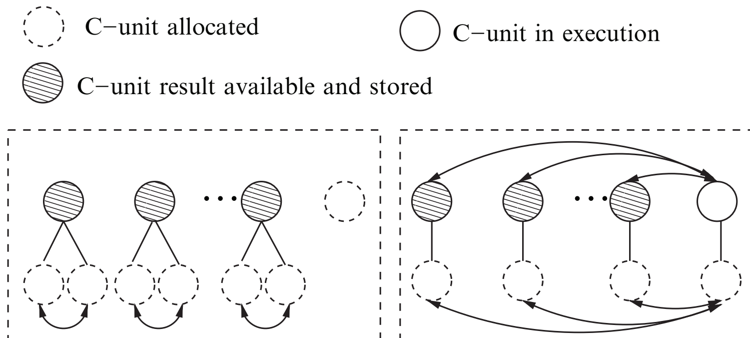


Fig. 5. Initial allocation of C-unit in a child instruction

no need to allocate any C-units in the child instruction to take the result of the newly issued C-unit in the parent instruction, since this result will not be available until two cycles later.

However, in the case shown on the right side of Fig. 5, all the available results in the parent instruction are to be compared with the last C-unit, the result of which will be available in the next cycle. Therefore, every result has its comparison counter part and is *conformable* with the new result in the parent instruction. In this situation, the child instruction only needs to allocate one C-unit for each unconfirmed input, with the same comparison companion relationship constructed as the parent instruction. A C-unit is also allocated in the child instruction for the C-unit currently being executed in the parent instruction, the result of which will be available in the next cycle. With this initial C-unit allocation, the child instruction can confirm in minimum latency when the fault rate is low.

To summarize, for every unconfirmed result in the parent instruction, the initial C-unit allocation for a child instruction is described below:

- For inputs known to be incorrect, no C-unit is allocated.
- If the input is known to be *distinct*, allocate two C-units initially.
- If the input is *conformable*, allocate only one C-unit initially and assign the comparison companion according to the parent instruction.

Information Propagation

In a child instruction, based on (1) the state of the input data, and (2) the computation within a specific branch, the result of a speculative branch at any cycle can fall into one of the following categories:

1. *Full resolution.*
   - *Correct.* Both the input data and the computation are confirmed to be correct; thus the instruction can be confirmed with this result.
   - *Wrong.* Either the input data or the computation within the branch itself is confirmed to be wrong; thus this result is known to be incorrect.
2. *Half resolution.*
   - *Global.* The input data is confirmed to be correct but the correctness of the computation within the branch is not confirmed yet.
   - *Locally confirmed.* The input data is not confirmed but the computation is locally confirmed within the branch.
3. *Zero resolution.*
   - *Unknown.* No information is available for the correctness of the input data, and the result is not confirmed within the branch either.

Table 1 illustrates how a specific result falls into one of the above categories according to the information from the input data as well as the computation within the child instruction branch.

It can be observed that in order for a result within a speculative branch to be *correct*, it has to satisfy two conditions (1) the input data has to be

Table 1. Speculation branch result categories

Computation within speculative branch	Input data from parent instruction		
	correct	wrong	unconfirmed
confirmed with conformation	*correct*	*wrong*	*locally confirmed*
confirmed to be incorrect	*wrong*	*wrong*	*wrong*
unconfirmed	*global*	*wrong*	*unknown*

confirmed as correct, and (2) the result conforms with another one locally within the branch. When any of the two conditions turns out to be false, the result is deemed *wrong*. When one of the conditions is true while the other not available yet, the result is either *locally confirmed* or *global*.

If a pair of results in a branch are marked as *global*[4] and they conform, then the instruction can be confirmed with this pair of results marked as *correct*. Otherwise, if the result pair within a branch agrees but is *unknown*, then the pair of results becomes *locally confirmed*.

During the process of the fault tolerance computation, the information obtained from a parent instruction comparison needs to be dynamically propagated to all the child instructions, so as to direct the branch pruning and hardware allocation in the child instruction. Basically, according to the comparison performed in the parent instruction, two types of information can be propagated:

- Information propagation on non-conforming comparison results
  If a pair of results in a parent instruction is compared, the non-conforming comparison result affects only the child instructions that had initially allocated one C-unit for each of the results, as is shown in the right hand part of Fig. 5. When the two C-units in the child instruction are set as comparison companions according to the parent instruction, this is based on the presumption that the two inputs are conforming. When the resulting C-unit pair in the parent instruction does not conform, the information should be propagated to cancel the comparison between the two child C-units, since they are deemed non-conforming due to their distinct data inputs. Through the propagation of the non-conforming comparison results, the *distinct* property is propagated from the parent instruction results to the corresponding results in the child instructions.
- Information propagation on conforming comparison results
  When a parent instruction is confirmed to be *correct*, information is passed to preserve the correct and prune the wrong speculation branches. The speculative branches that take the *correct* results as inputs are marked as *global*, indicating that the input inherited from the parent instruction is confirmed. If the results of the children are already *locally con-*

[4] If an instruction does not depend on the results of any unconfirmed instructions, i.e., it has no parent instruction, then all its unconfirmed results are marked as *global*.

firmed, then they become *correct* and the child instruction can be confirmed too. The speculative branches taking the *wrong* results as input will inherit the *wrong* attribute and propagate the information to all the descendants, thus pruning the speculation branches.

C-Unit Update

The speculative execution of a child instruction is based on the unconfirmed results of the parent instruction. These unconfirmed results of the parent instruction, however, may be dynamically generated after the initial C-unit allocation of the child instruction. A full speculation scheme allocates new C-units and forms a new speculation branch in the child instruction for every newly generated result in the parent instruction.

To control the growth of hardware resources in the speculation of child instructions, the allocation of new C-units for a child instruction should be strictly limited. A voter only allocates new C-units in two cases (1) the initial C-unit allocation, and (2) when the child instruction becomes *impossible to confirm*. The first case has been discussed in the previous section, and we focus on the second case in this section. There are two situations when a child instruction can be identified as *impossible to confirm*:

- All the results in the child instruction turn out to be *wrong*.
  This occurs when every speculation branch of the child instruction turns out to have a wrong input. In other words, the results which the child instruction inherited from the parent instruction all turn out to be *wrong*. Obviously the confirmation of the child instruction is impossible under this situation.
  The new C-units are allocated by taking the results which are not identified as *wrong* in the parent instruction. Similar to the initial C-unit allocation process, two C-units are allocated for each *distinct* result while one C-unit is allocated for each *conformable* result.
- There exist some unconfirmed results in the child instruction which are not *wrong*, but all known to be *distinct*.
  This is either due to the propagation of the *distinct* attribute, or because the computation within the branch fails to conform. Under this situation the child instruction becomes impossible to confirm, due to the lack of local confirmation capability within each speculative branch.
  To make the child instruction possible to confirm, new C-units are allocated by duplicating the computation of the *distinct* results.

4.5 An Example

We show an example of dynamic allocation of C-units in Fig. 6, where a sequence of instructions, A, B, C, D, E, are executed and confirmed, each depending on the result of the previous one. A C-unit of any instruction can be

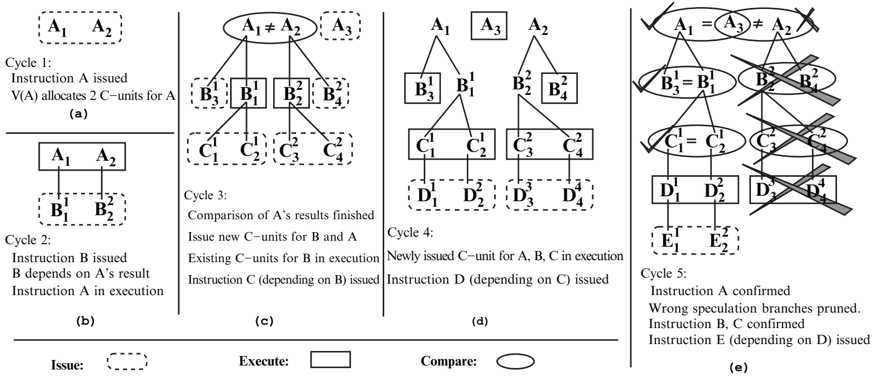


Fig. 6. An example of five cycles for a sequence of instructions using the proposed allocation algorithm

in one of the three main states: issue, execute, and compare. For representational convenience, we use the superscript of a C-unit to indicate the index of the parent-level C-unit from which the input is taken; we use the subscript of a C-unit for its own index.

Figure 6a shows the first cycle, during which instruction A is issued; the voter of A initially allocates two C-units, A_1 and A_2, in this cycle. In the second cycle shown in the (b) part, A_1 and A_2 are in the execution cycle, while at the same time instruction B is issued. B depends on A's result. At this stage A_1 and A_2 are set as comparison companions. According to the algorithm, the voter of B allocates two C-units B_1^1 and B_2^2, taking the results from A_1 and A_2 correspondingly.

In the third cycle, as is shown in Fig. 6c, the comparison of A_1 and A_2 is finished and the results do not conform. A_1 and A_2 are thus known to be *distinct*, and the information is passed further to split B_1^1 and B_2^2 to be *distinct* as well.

To continue the fault tolerance computation of A, the voter of A allocates and issues a new C-unit A_3 in this cycle. Now that B is known to be *impossible to confirm*, since all its C-units are *distinct*, the voter of B also needs to allocate two new C-units by duplicating B_1^1 with B_3^1 and B_2^2 with B_4^2. These newly allocated C-units are issued for instruction B in the third cycle.

Instruction C is also issued in the third cycle. Since B_1^1 and B_2^2 are in the execution stage and their results will be available in the next cycle, C-unit allocation for instruction C is only considered for them. Since both B_1^1 and B_2^2 are *distinct*, two C-units are allocated for each: C_1^1 and C_2^1 are set to take the result from B_1^1, while C_3^2 and C_4^2 are set to take the result from B_2^2. On the other hand, B_3^1 and B_4^2 are still in the issue stage, so no C-units are allocated to await their result at this cycle.

The (d) part of Fig. 6 shows the fourth cycle of the example. A_3, B_3^1 and B_4^2 are now in the execution stage while the results of B_1^1 and B_2^2 are available

and have been passed to $\{C_1^1, C_2^1\}, \{C_3^2, C_4^2\}$ correspondingly. No comparison is made between B_1^1 and B_2^2 because they are already identified to be distinct. All four C-units of instruction C are in the execution stage. Instruction D is issued with four C-units allocated, inheriting the comparison companion relationship of instruction C.

The (e) part in Fig. 6 shows the propagation of confirmation as well as the branch pruning process in the fifth cycle. In this cycle, instruction A is confirmed with $A_1 = A_3$, while the result of A_2 is identified to be *wrong* and the corresponding speculative branch is pruned. Also in this cycle, instructions B and C are locally confirmed with $B_3^1 = B_1^1$ and $C_1^1 = C_2^1$. Since the speculative branch of A_1 is confirmed to be correct, instruction B is confirmed, which further confirms instruction C.

The example shows that the proposed hardware frugal allocation algorithm can be used to achieve a fault tolerance scheme with high flexibility for performance. In this example, although A_2 is faulty and the confirmation of A is delayed for two cycles, with the proper control of speculative branches, the delay is not propagated to the subsequent dependent instructions and the whole sequence of instructions is confirmed in time.

4.6 Discussion

The fault tolerance computational model for the nanoelectronic processor architecture basically consists of a voter/C-unit architecture to perform a hybrid hardware and time redundancy based fault tolerance approach, and a novel controlled speculation mechanism for data dependent instructions.

With the new computational model, among data dependent instructions, the speculated execution of the instructions is performed out of order while the confirmed results are always generated in order. In general, such a computational model supports out-of-order execution among independent instructions so as to exploit the parallelism offered by the nanoelectronic environment. The related issues of out-of-order execution, including the imprecise exception problem, are essentially similar to the same issues existing in traditional CMOS based architectures, and can be approached accordingly with a number of available techniques being utilized currently.

The speculative execution in the computational model can be further extended beyond the arithmetic instructions to the branch instructions. Essentially, multiple speculations can be formed on the branch targets and the wrong speculation can be pruned according to the address calculation.

Overall, the fault tolerant computational model targets the main reliability challenge of the emerging nanoelectronic environment, and provides a novel approach that integrates fault tolerance, performance and hardware overhead considerations. These conflicting optimization criteria are effectively balanced under such a computational model at the processor architecture level.

4.7 Simulation Results

In the above described computational model, hybrid hardware and time redundancy is used to guarantee the correctness of instruction executions. For the instructions with dependencies, the hardware allocation algorithm is used to balance the performance and hardware utilization according to the occurrence of faults. A simulation framework is developed to evaluate the effectiveness of the computational model. Detailed simulation results are available in [49–51].

Basically, two sets of simulations are provided. First, the new fault tolerance computational model is compared with the *no speculation* and *full speculation* approaches, which represent the two extreme points in the performance/latency tradeoff. Both hardware requirement and latency are compared for the three strategies. Through this set of experiments, the tradeoff between latency and hardware is examined. Secondly, various replication quantities, N, for the initial NMR hardware redundancy approach are compared. The results show the multiple tradeoff points existing within the computational model and they have corresponding applicability under various fault rates.

It turns out that, as expected, the *no speculation* approach suffers from significant delays in comparison to the other two models. Since no speculation branch is ever formed, the data dependencies among the instruction sequences result in the delayed issuing of child instructions. The tremendous latency of the *no speculation* approach grows and makes the gap even larger as fault rates increase. The *full speculation* approach, as expected, achieves the minimum latency since it utilizes hardware resources without any constraints to provide the same redundancy for every speculation branch. The new computational model, in terms of latency, exhibits behavior similar to that of the *full speculation* approach. Overall, simulation results confirm that the new computational model can achieve a near-optimum performance that is comparable to the *full speculation*, which is the best case in terms of performance [49–51].

The hardware resource consumption in the fault tolerance computation models is considered from two aspects. First, the number of C-units occupied at each cycle shows the amount of computational unit hardware requirement in the system. Notice that a C-unit is released once its computation is finished and the result is returned to the voter. The number of C-units occupied therefore indicates the amount of parallelism existing when multiple speculation branches are executed. Second, since the unconfirmed results are stored by the voter, the number of unconfirmed results during the computation depicts the storage hardware requirement in a voter. From both aspects of hardware consumption, the new computational model displays efficient hardware allocation, that significantly outperforms the *full speculation* approach and is close to the *no speculation* approach in hardware requirement.

Essentially, through the simulation results it can be observed that the new computational model shows significant results both in performance and hardware, thus achieving the goal of fault tolerance with ideal balance of hardware and latency. In comparison to the other two models, the new computation

model can be seen to compete with the best aspects of both, i.e., the short delay of the *full speculation* and the frugal hardware allocation in the *no speculation* model.

As the emerging nanoelectronics are expected to offer a density boost in the order of 10^3–10^6 compared to today's CMOS technology, a crucial determinant factor for the amount of corresponding boost of parallel computation power at the architectural level is the amount of hardware that is demanded for fault tolerance purposes. According to the simulation results, the new computational model shows significant potential by using an order of magnitude less hardware for fault tolerance purposes while sacrificing neither reliability nor performance, thus supporting eventually the boost of computation power in the nanoprocessor architectures.

The minimum initial NMR of C-unit allocation for the proposed computational model is two; however, the manner in which various initial C-unit settings influence the behavior of the algorithm bears further scrutiny. In this experiment set, we compare the minimum case of two initial C-units with the cases of initially allocating three and four C-units. The simulation results are expected to provide knowledge of various tradeoff points within the proposed scheme.

With more initially allocated C-units, an instruction can be confirmed more quickly when the fault rate is high. On the other hand, when the fault rate is low, some initially allocated C-units are essentially redundant, thus consuming comparatively more hardware resources. The simulation results confirm this syllogism. Essentially, when the fault rate is low, the configuration of minimum initial C-unit allocation provides a highly attenuated loss of latency, while exhibiting a significantly reduced amount of hardware consumption. When the fault rate is high, allocating more C-units in the initial cycle helps reduce the overall latency by and large, while consuming almost the same hardware resources as the minimum initial C-unit allocation configuration. Therefore, the initial C-unit allocation number in the proposed computational model provides multiple optimal points under various fault rate ranges.

5 Topological Structure Design of Nanoprocessor Architectures

The proposed computational model aims at providing fault tolerance for the nanoelectronics based processor architecture, with the tradeoff consideration for performance and hardware issues. In addition to the unreliability challenge, a number of other characteristics, particularly, the localized communication constraint imposed by nanodevices, need to be investigated. Due to these new characteristics, multiple issues are raised when the proposed behavioral computational model is mapped to a number of structural components in a processor, consisting of both CMOS and nanodevices. In this section, we discuss a

coarse distribution of architecture level components in a CMOS/nano hybrid system, based on a number of existing research approaches for the interface design among the CMOS and nano systems. We investigate the message passing mechanisms that are crucial in the communication of the proposed fault tolerance computational model, with a further topological constraint imposed by the limitation of localized communication in nanoelectronic environment.

5.1 Research Approaches Supporting the Addressing Mechanism at the CMOS/Nano Interface

A number of research approaches have shown various interface designs that integrate a crossbar based nanoelectronic system with a CMOS based system [17–20, 43]. Based on these techniques, each nanowire/device can be addressed individually through a combination of CMOS devices.

In the CMOL approach [19, 20], a layer of nano crossbar is posed above a layer of CMOS cells. The interface between the nanowire and the CMOS cells is formed through a limited number of pins, such that each CMOS cell is connected to exactly one nanowire. A particular angle is formed between the CMOS cells and the nanowires; therefore, the addressing of each nanodevice located at the cross point in the nano crossbar structure can be approached through the combination of two CMOS cells.

An alternative approach has been proposed in [17, 18], which utilizes a decoder, possibly formed stochastically to perform the mapping from CMOS wires to nanowires. A number of microwires of CMOS scale are initially connected on a one-to-one base with the same number of nanowires, then expanded to an exponential number of nanowires through a decoder. A detailed description of the nanowire based architecture can be found in [17, 18, 43]

These techniques facilitate the communication between a CMOS system and a nano crossbar based system, thus exhibiting multiple possibilities of implementing a CMOS/nano hybrid architecture system. These approaches essentially provide the basis for the proposed computational model, which exploits the potentiality of both CMOS and nanoelectronic devices to overcome the challenges for the processor architectures in the nanoelectronic environment.

5.2 Processor Architecture Components Across CMOS/Nano Layers

We envision the construction of a nanoprocessor architecture in a hybrid approach, consisting of a reliable CMOS layer and an unreliable nano layer, each containing correspondingly the components of logic devices, buses/wires and storage elements. The main modules of the proposed nanoprocessor architecture consist of the main control system for the processor, the additional fault tolerance control of voters, the register file, memory and ALU

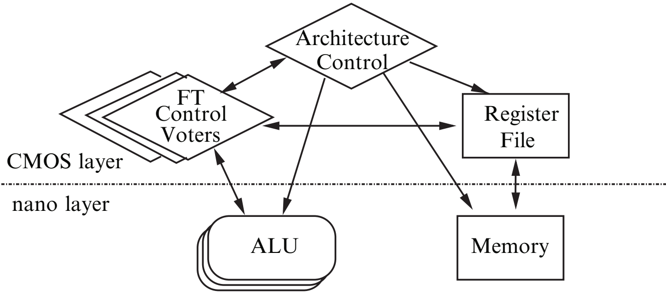


Fig. 7. Decomposition of the implementation layers across a CMOS/nano hybrid system for the components in the proposed nanoprocessor architecture

components. Figure 7 exhibits the information connection among the modules and our vision of their implementation in a CMOS–nano hybrid approach.

Basically, the main concern of implementing a component in the CMOS or the nano layer is related to the following differences:

1. *Performance.* Consisting of the following two aspects:
   - *Computation.* Nanodevices outperform CMOS devices significantly.
   - *Communication.* Nano signal transfer is expensive and suffers from significant wire delay, thus making it in general worse than the CMOS level communications.
2. *Hardware.* Nanodevices offer significantly more abundant hardware resources than CMOS.
3. *Reliability.* CMOS devices are significantly more reliable than nanodevices.
4. *Locality.* Communication among nanodevices is strictly constrained to within a nearby locality, in contrast to CMOS which has a comparatively much wider communication range.
5. *Power.* Nanoelectronic devices consume significantly lower power than the CMOS counterparts.

According to the above analysis, a component suffers from the reliability problem when implemented in the nano layer, while tending to be much more reliable when implemented in the CMOS layer. On the other hand, a component is more expensive in hardware if implemented in CMOS, in comparison to its implementation in nanoelectronics.

Based on the the consideration of the reliability/hardware cost tradeoff, ALU components and memory blocks, occupying the majority of hardware requirement in a processor architecture, should be implemented with nanoelectronic devices, so as to benefit from the abundant hardware resources. Furthermore, since ALU is heavily computational oriented, a nanodevice based ALU also has the advantage of computational performance boosting. However, from the aspect of locality, implementing these components in the nano layer strictly limits the communication range among the components within the nano layer. Global communication in the nano layer is prohibitively expensive,

while only localized communication among the ALU components (C-units) is supported. Consequently, the communication among the ALU components should be limited within a local area.

The control units, including the processor architecture controller and the voters for fault tolerance purposes, impose higher demands of reliability and massive communications with other modules. Therefore, the implementation of these control components in the CMOS layer is advantageous in terms of reliability, locality, and communication performance.

Due to the large loads of communication between the register file and the voters, it is more beneficial in terms of communication performance to implement the register file in the CMOS level.

5.3 Message Passing Mechanism

During the speculative computations, the information regarding an instruction being confirmed or speculation branches being confirmed/pruned is transferred among the voters of dependent instructions. Unconfirmed results are transferred directly among the C-units. Voters and C-units also communicate during the C-unit allocation process and the result transmission process.

Essentially, three types of messages are transferred in the proposed fault tolerant computational model:

- *Voter/voter.* Messages transferred among the voters contain information about the speculative computation: confirmation of instructions and confirmation/pruning/splitting of speculation branches. These messages are always transferred unidirectionally from the voters of the parent instructions to the voters of the dependent child instructions.
- *Voter/C-unit.* Messages transferred among the voters and the C-units contain the C-unit allocation and release information, parameters of the instructions sent from a voter to the C-units, and the transmission of unconfirmed results from a C-unit to its voter, thus consisting of bidirectional transfer of messages.
- *C-unit/C-unit.* Messages transferred among the C-units contain the direct passing of unconfirmed results among the C-units of dependent instructions. The messages are always transferred unidirectionally from the C-units of the parent instructions to the C-units of the dependent child instructions.

The three types of messages described above can be transferred by two genres of communication mechanisms: one with the sender in control, while the other with the receiver in control.

The first and the third types of messages, the voter/voter and the C-unit/C-unit, are transferred unidirectionally from the parent instructions to the dependent child instructions. Since the parent instructions are always dispatched ahead of the child instructions, the sender of the messages in this case cannot be aware of the receivers. The receivers, on the other hand, can

establish a connection with the sender easily at the initial stage, both in the cases of voter/voter messages and the C-unit/C-unit messages. Therefore, the messages need to be passed in a *receiver-in-control* mechanism, where a sender simply broadcasts the message while each receiver is responsible for identifying the messages relevant to it.

The second type of messages, which are transferred between a voter and multiple C-units, consist of information passing in both directions: from the voter to the C-units during C-unit allocation, and from the C-units back to the voter when returning the results. In this case, the voter and the C-units are of the same instruction. The C-units are allocated and initialized by the voter, during which process the communication bond can be established on both sides. Therefore, either the voter or the C-unit, when acting as the sender of a message can clearly identify the receivers, supporting a *sender-in-control* message passing mechanism to address the specific receivers without the overhead of broadcasting.

The receiver-in-control message passing mechanism can be easily implemented on a bus topology, while the sender-in-control mechanism fits best to a star topology with the voter connecting multiple C-units. Specifically, the messages within the voters are passed through a common bus, supporting the receiver-in-control, sender-broadcasting mechanism. A similar bus can be used for message passing among the C-units. Among a voter and its related C-units, a star topology is used to support the sender-in-control message passing.

In a CMOS–nano hybrid system, since the voters are in the CMOS layer while the C-units are implemented by the nanoelectronics, the bus among the voters is implemented by CMOS-level wires while the bus among the C-units is implemented by nanowires. The connections among the voters and the C-units can be implemented through the access mechanism developed for the CMOS/nano interface [17–20].

Error checking code (ECC) can be applied to the message transfer process to enhance reliability. It is worth noticing that, although the reliability of information transfer can be enhanced by applying well developed coding techniques, as in the memory case, the proposed fault tolerant computational model neither presumes, nor relies on the reliability in the message transfer process involving the nanodevices. Any fault or failure in the message transfer of voter/C-unit and C-unit/C-unit can be dealt with in the same manner as the failure of C-unit computations, thus covered by the proposed fault tolerance scheme.

5.4 Locality Constraint Aware Network Topology

The simple network topology model described above inherently bears a number of constraints. The receiver-in-control mechanism requires a broadcasting on the common bus; however, such a common bus is always subject to the contention problem. One solution to the contention problem is to introduce multiple buses. When broadcasting, a random subset of buses is selected so

as to minimize the collision probability. Furthermore, the star topology with the voter at the center is not flexible in the run-time environment where the number of C-units requested by a voter is dynamically changing. Allowing voters to share accessibility to all the C-units provides maximum flexibility, however, at the price of a highly complex mesh network topology.

More importantly, when the characteristics of a nanoelectronic environment are included in the consideration, a number of new challenges are raised. Neither the global common bus nor the complete mesh network fits the locality constraint severely imposed in a nanoelectronic environment.

Taking into consideration the localized interconnection constraints, the accessibility of any component in the nanoprocessor is limited to a localized neighborhood. Figure 8 exhibits a revised logic topology of the voters and the C-units adjusted to the localized communication constraint. As is shown in Fig. 8, the voters are connected by a number of buses that cover merely a local neighboring area. A similar localized bus structure is shown among the C-units. The communication among the voters, as well as among the C-units, is therefore limited to the ones that share at least one common local bus. The interconnection among the voters and the C-units is also organized to support the sharing of C-units among the voters in a localized manner. Essentially, neighboring voters have accessibility to a common set of C-units, thus being flexible when unbalanced C-unit allocation occurs dynamically among multiple neighboring voters.

Figure 8 shows the logic topology of the voters and C-units, where these components are placed on a number of one-dimensional lines. When the specific regular structure constraint is considered, these components need to be physically placed on a two-dimensional regular structure based nanofabric. It can be envisioned that the voter and the C-units need to be placed with an interleaved manner, as is shown in the example of Fig. 9, where each voter has a number of locally accessible C-units. The accessibility among the voters

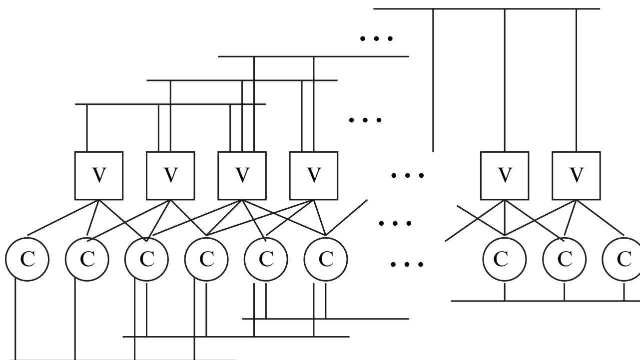


Fig. 8. The logic message passing channels for voters and C-units considering the locality constraint of nanoelectronic environment

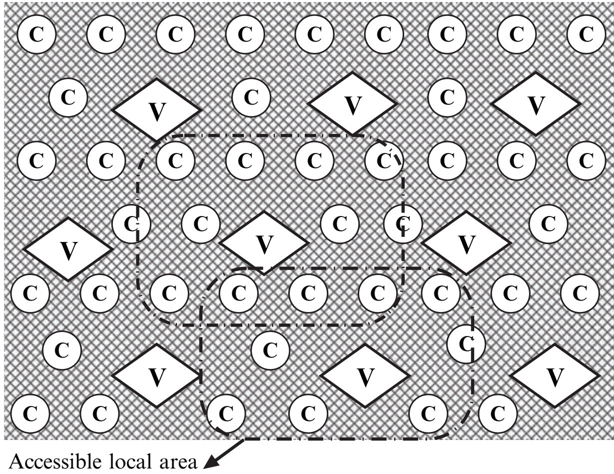


Fig. 9. The topology of voters and C-units on a regular structured nanofabric considering the locality constraint

and among the C-units is similarly limited to within a local range, with an underlying network implementing the logic topology of Fig. 8.

The localization constraint further influences the schedule and dispatch of instructions. Basically, according to the localized communication among the voters, dependent instructions need to be dispatched into a neighboring area where communication is feasible for the voter/voter messages. When allocating C-units for the fault tolerance computation, the same locality principle needs to be applied for the C-units so as to enable the fast transfer of unconfirmed results for dependent instructions among the neighboring C-units.

It is certainly true that the locality principle of most programs supports the dispatch of dependent instructions to the neighboring voters and C-units. However, how to optimally perform the mapping of a program with arbitrary dependency structure onto the regular fabric based nanoprocessor remains challenging.

6 Conclusion

Multiple challenges and opportunities are raised by the new characteristics of nanoelectronics. Particularly, for the construction of a nanoelectronics based processor architecture, unreliability, localized communication constraints, performance and hardware resource tradeoffs all need to be considered. A new architectural level computational model for the future nanoelectronic processors is developed, addressing the above issues.

The main techniques in this architecture level computational model include (1) a fault tolerance scheme exploiting hardware and time redundancy

dynamically to guarantee the correctness of each instruction, (2) the idea of computation through multiple speculative branches to improve system performance in data dependent instructions, (3) the algorithm of dynamically allocating computation units to avoid the exponential growth in hardware consumption, and (4) localized communication among the processor components in the computational model considering the interconnect overhead in the nanoelectronic environment.

Such a computational model exploits the abundant hardware resources provided by the nanoelectronic technology and makes tradeoffs between hardware and computation performance. Fault tolerance in instruction execution can be guaranteed and system performance is boosted as well; yet communications among the nanoelectronics based components are designed to be within a localized area. Through the simulation with multiple parameters, several tradeoff points are identified for the computational model, thus providing an insight in selecting the proper parameter for a certain fault rate range to achieve the best hardware/latency tradeoff. The overall simulation results confirm the effectiveness of the computational model from both a performance and a hardware overhead perspective, thus evincing that a strong solution is provided to the vital challenges in architecture level fault tolerant computation in nanoelectronic processors. Overall, the computational model sets up a starting point of designing nanoelectronic based processors, and based on the emerging nanoelectronic characteristics provides a framework in investigating multiple nanoelectronic characteristics related issues including reliability, performance, hardware and localized communication at the architecture level for the future processors based on nanoelectronics.

References

1. ITRS, *International Technology Roadmap for Semiconductors Emerging Research Devices*, 2006.
2. European Commission, *Technology Roadmap for Nanoelectronics*, 2001.
3. P. Avouris, J. Appenzeller, R. Martel and S. Wind, "Carbon Nanotube Electronics", *Proceedings of the IEEE*, vol. 91, n. 11, pp. 1772–1784, 2003.
4. P. Mazumder, S. Kulkarni, M. Bhattacharya, J. P. Sun and G. I. Haddad, "Digital Circuit Applications of Resonant Tunneling Devices", *Proceedings of the IEEE*, vol. 86, n. 4, pp. 664–686, April 1998.
5. C. S. Lent, P. D. Tougaw, W. Porod and G. H. Bernstein, "Quantum Cellular Automata", *Nanotechnology*, vol. 4, pp. 49–57, 1993.
6. M. A. Kastner, "The Single-Electron Transistor", *Review of Modern Physics*, vol. 64, pp. 849–858, 1992.
7. R. H. Chen, A. N. Korotkov and K. K. Likharev, "Single-electron Transistor Logic", *Applied Physics Letters*, vol. 68, n. 14, April 1996.
8. J. C. Ellenbogen and J. C. Love, "Architectures for Molecular Electronic Computers: 1. Logic Structures and an Adder Designed from Molecular Electronic Diodes", *Proceedings of the IEEE*, vol. 88, n. 3, pp. 386–425, 2000.

9. Y. G. Krieger, “Molecular Electronics: Current State and Future Trends”, *Journal of Structural Chemistry*, vol. 34, pp. 896–904, 1993.
10. M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach and M. M. Ziegler, “Molecular Electronics: From Devices and Interconnect to Circuits and Architecture”, *Proceedings of the IEEE*, vol. 91, n. 11, pp. 1940–1957, November 2003.
11. C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams and J. R. Heath, “Electronically Configurable Molecular-Based Logic Gates”, *Science*, vol. 285, pp. 391–394, July 1999.
12. S. A. Wolf, D. D. Awschalom, R. A. Buhrman, J. M. Daughton, S. von Molnar, M. L. Roukes, A. Y. Chtchelkanova and D. M. Treger, “Spintronics: A Spin Based Electronics Vision for the Future”, *Science*, vol. 294, pp. 1488–1495, November 2001.
13. Y. Huang, X. Duan, Y. Cui, L. J. Jauhon, K. Kim and C. M. Lieber, “Logic Gates and Computation from Assembled Nanowire Building Blocks”, *Science*, vol. 294, pp. 1313–1317, November 2001.
14. P. J. Kuekes, D. R. Stewart and R. S. Williams, “The Crossbar Latch: Logic Value Storage, Restoration, and Inversion in Crossbar Circuits”, *Journal of Applied Physics*, vol. 97, n. 3, pp. 034301, July 2005.
15. G. Snider, P. J. Kuekes and R. S. Williams, “CMOS-like Logic in Defective, Nanoscale Crossbars”, *Nanotechnology*, vol. 15, pp. 881–891, August 2004.
16. G. Snider and W. Robinett, “Crossbar Demultiplexers for Nanoelectronics Based on n-Hot Codes”, *IEEE Transactions on Nanotechnology*, vol. 4, pp. 249–254, 2005.
17. A. DeHon and M. J. Wilson, “Nanowire-based Sublithographic Programmable Logic Arrays”, in *FPGA*, pp. 123–132, 2004.
18. A. DeHon, “Array-Based Architecture for FET-Based, Nanoscale Electronics”, *IEEE Transactions on Nanotechnology*, vol. 2, n. 1, pp. 23–32, 2003.
19. D. B. Strukov and K. K. Likharev, “CMOL FPGA: A Reconfigurable Architecture for Hybrid Digital Circuits with Two-terminal Nanodevices”, *Nanotechnology*, vol. 16, pp. 888–900, April 2005.
20. D. B. Strukov and K. K. Likharev, “A Reconfigurable Architecture for Hybrid CMOS/Nanodevice Circuits”, in *ACM FPGA*, pp. 131–140, 2006.
21. P. Beckett and A. Jennings, “Towards Nanocomputer Architecture”, in *Asia-Pacific Computer System Architecture Conference*, pp. 141–150, 2002.
22. K. Nikolic, A. Sadek and M. Forshaw, “Architectures for Reliable Computing with Unreliable Nanodevices”, in *Proceedings of the 1st IEEE Conference on Nanotechnology*, pp. 254–259, 2001.
23. J. R. Heath, P. J. Kuekes, G. S. Snider and S. Williams, “A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology”, *Science*, vol. 280, pp. 1716–1721, June 1998.
24. S. C. Goldstein and M. Budiu, “NanoFabrics: Spatial Computing Using Molecular Electronics”, in *ISCA*, pp. 178–191, 2001.
25. S. C. Goldstein, M. Budiu, M. Mishra and G. Venkataramani, “Reconfigurable Computing and Electronic Nanotechnology”, in *ASAP*, pp. 132–143, 2003.
26. M. S. Montemerlo, J. C. Love, G. J. Opitech, D. G. Gordon and J. C. Ellenbogen, *Technologies and Designs for Electronic Nanocomputers*, MITRE, July 1996.
27. T. Juhnke and H. Klar, “Calculation of the Soft Error Rate of Submicron CMOS Logic Circuits”, *IEEE Journal of Solid-State Circuits*, vol. 30, n. 7, pp. 830–834, July 1995.

28. T. Karnik, P. Hazucha and J. Patel, “Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes”, *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 128–143, April–June 2004.
29. P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger and L. Alvisi, “Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic”, in *DSN*, pp. 1112–1119, 2002.
30. M. Forshaw, R. Stadler, D. Crawley and K. Nikolic, “A Short Review of Nanoelectronic Architectures”, *Nanotechnology*, vol. 15, pp. 220–223, 2004.
31. K. Nikolic, A. Sadek and M. Forshaw, “Fault-tolerant Techniques for Nanocomputers”, *Nanotechnology*, vol. 13, pp. 357–362, 2002.
32. J. von Neumann, “Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components”, in C. Shannon and J. McCarthy, editors, *Automata Studies*, Princeton University Press, Princeton, 1956.
33. J. Han and P. Jonker, “A System Architecture Solution for Unreliable Nanoelectronic Devices”, *IEEE Transactions on Nanotechnology*, vol. 1, n. 4, pp. 201–208, December 2002.
34. J. Han, J. Gao, Y. Qi, P. Jonker and J. A. B. Fortes, “Toward Hardware-Redundant, Fault-Tolerant Logic for Nanoelectronics”, *IEEE Design and Test of Computers*, vol. 22, n. 4, pp. 328–339, July–August 2005.
35. T. M. Austin, “DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design”, in *ACM/IEEE Annual Symposium on Microarchitecture*, pp. 196–207, 1999.
36. P. Agrawal, “Fault Tolerance in Multiprocessor Systems without Dedicated Redundancy”, *IEEE Transactions on Computers*, vol. 37, pp. 385–362, March 1988.
37. D. K. Pradhan and N. H. Vaidya, “Roll-Forward Checkpointing Scheme: A Novel Fault-Tolerant Architecture”, *IEEE Transactions on Computers*, vol. 43, pp. 1163–1174, October 1994.
38. A. Dahbura, K. Sabnani and W. Henry, “Spare Capacity as a Means of Fault Detection and Diagnosis in Multiprocessor Systems”, *IEEE Transactions on Computers*, vol. 38, n. 6, pp. 881–891, June 1989.
39. S. Tridandapani, A. K. Somani and U. R. Sandadi, “Low Overhead Multiprocessor Allocation Strategies Exploiting System Spare Capacity for Fault Detection and Location”, *IEEE Transactions on Computers*, vol. 44, pp. 865–877, July 1995.
40. M. A. Gomaa, C. Scarbrough, T. N. Vijaykumar and I. Pomeranz, “Transient-Fault Recovery for Chip Multiprocessors”, *IEEE Micro*, vol. 23, n. 6, pp. 76–83, November/December 2003.
41. G. Manimaran and C. S. R. Murthy, “A Fault-Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems and Its Analysis”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, pp. 1137–1152, November 1998.
42. B. Izadi and F. Ozguner, “Enhanced Cluster k-Ary n-Cube, A Fault-Tolerant Multiprocessor”, *IEEE Transactions on Computers*, vol. 52, n. 11, pp. 1443–1453, November 2003.
43. A. DeHon, “Nanowire-Based Programmable Architectures”, *ACM JETC*, vol. 1, n. 2, pp. 109–162, 2005.
44. R. B. Blahut, *Algebraic Codes for Data Transmission*, Cambridge University Press, Cambridge, 2002.

45. P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*, Morgan Kaufmann, San Francisco, 2000.
46. J. H. Patel and L. Y. Fung, "Concurrent Error Detection in ALUs by Recomputing with Shifted Operands", *IEEE Transactions on Computers*, vol. 31, pp. 589–592, December 1982.
47. K. Wu and R. Karri, "Algorithm Level RE-computing with Shifted Operands - A Register Transfer Level Concurrent Error Detection Technique", in *ITC*, pp. 971–978, 2000.
48. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Third Edition, Morgan Kaufmann, San Francisco, 2002.
49. W. Rao, A. Orailoglu and R. Karri, "Fault Tolerant Nanoelectronic Processor Architectures", in *ASPDAC*, pp. 311–316, 2005.
50. W. Rao, A. Orailoglu and R. Karri, "Architectural-Level Fault Tolerant Computation in Nanoelectronic Processors", in *ICCD*, pp. 533–542, 2005.
51. W. Rao, A. Orailoglu and R. Karri, "Towards Nanoelectronics Processor Architectures", *JETTA Special Issue on Test, Defect Tolerance, and Reliability of Nanoscale Devices*, vol. 23, pp. 235–254, 2007.

Chapter 14: Design and Analysis of Fault-Tolerant Molecular Computing Systems

D. Bhaduri, S.K. Shukla, H. Quinn, P. Graham, and M. Gokhale

1 Introduction

As electronics enters the nanoscopic realm, patterning and fabrication costs to manufacture nanoscale CMOS chips are expected to increase exponentially. Hence, researchers are looking at technologies that can be more viable for the miniaturization of electronics than conventional technologies that require almost perfect control over lithography, etching and other processes. One of the technologies that the microelectronics community has been investigating is chemical self-assembly of devices from elementary and identical molecular units by controlled deposition of molecular monolayers on the substrates, a technology that has given birth to molecular electronics [28].

Molecular electronic devices usually consist of self-assembled organic molecules sandwiched between conducting electrodes. Early work on these devices showed that they either exhibited electron tunneling [20] or rectification [1], but reaching agreement between theory and experiments has always been a major hurdle [11]. Recently, molecular switch tunnel junctions have been constructed experimentally [2] and have been seen to exhibit prominent switching behavior. However, such switching behavior can also arise from non-molecular mechanisms leading to inherent transient faults [19]. Manufacturing defects have also been observed to be a major problem since controlled fabrication of these devices within specified tolerances is a non-trivial process, specifically, controlling the properties of molecule-electrode interfaces [11] is the main challenge. Therefore, there is a need to develop methodologies and tools to design molecular systems that are tolerant to both manufacturing defects and transient faults.

Chemically self-assembled molecular nanofabrics (Fig. 1) are by nature very regular and homogeneous, hence, well-suited for reconfigurability [9, 10, 12]. Due to the reconfigurable nature of molecular nanofabrics, defect-mapping techniques [15, 21] followed by defect-avoidance can be used to circumvent manufacturing defects in these nanofabrics so that systems can be built which

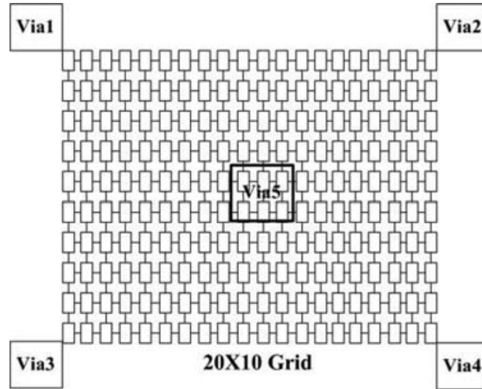


Fig. 1. Molecular nanofabric

are close to 100% defect-tolerant. But such systems are still highly susceptible to transient faults. It has been shown in [13, 14, 22], that, the addition of structural redundancy a priori may enhance the reliability of such systems in the presence of transient faults.

There are two schools of thought for designing fault-tolerant molecular nanofabrics and the application of any of these design philosophies depends on the failure tolerance threshold of the system being designed. One school of thought [6, 14] considers mapping systems directly onto unreliable reconfigurable nanofabrics with adequate structural redundancy, hence saving computational time and cost associated with defect-mapping and avoidance. This design philosophy is only applicable to systems that are not mission-critical and admit low but nonzero failure probabilities. The other school of thought is applicable to mission-critical systems that need 100% reliability guarantee, hence, this design philosophy uses defect-mapping and defect avoidance techniques on reconfigurable molecular nanofabrics to circumvent manufacturing defects and then applies structural redundancy-based techniques to tackle transient faults.

Both of the above design philosophies have a number of problems that hinder their usefulness in designing fault-tolerant systems on molecular nanofabrics. These problems are as follows:

- Most defect map generation techniques consider fail-stop deterministic defect models [24] that assume nodes to either function perfectly or fail completely. Others use a large number of test-circuit configurations to determine the exact functional correctness of each node [21], hence, consuming more computational resources and time. Hence, there is a need for generating defect maps that can reflect the inherent non-deterministic nature of manufacturing defects in the nodes, interconnects and interfaces (vias) without using a large number of test-circuits.

- Although [21,24] propose scalable defect-mapping techniques, the problem of scaling these techniques to ultra-dense nanofabrics remains far from solved.
- There is a lack of scalable methodologies that can map hardware designs onto dense molecular nanofabrics.
- If structural redundancy is inserted a priori, the *redundancy factor* (R) that can guarantee tolerance to transient faults needs to be determined. We define R as the ratio of the circuit sizes of the redundant and non-redundant designs. Figuring out the correct architectural level (gate, configurable logic block, functional block, etc.) where redundancy may be most effective and analyzing the performance, area and cost penalties that may be incurred due to redundancy insertion are additional challenges.

A lot of effort is being employed to provide solutions to these challenges [16,21, 24]. This chapter attempts at providing methodologies and tools for addressing these issues. Our proposed solutions to these problems are as follows:

- We address the first problem by developing a non-deterministic defect-mapping scheme which is a hybrid of the methodologies proposed in [21] and [24].
- The second problem of scalability is addressed by modeling the molecular nanofabrics hierarchically, and computing the defect distributions of the nanofabric nodes by applying state space partitioning techniques to the Markovian analysis [25].
- The third issue has been reasonably dealt with in [16]. The authors in that work have proposed a three level design hierarchy to map hardware designs onto ultra-dense nanofabrics with bounded complexity. We have extended this methodology so that behavioral and/or structurally redundant systems can be mapped effectively to nanofabrics.
- Since there is no formal methodology to pin-point R and the granularity level for structural redundancy insertion, we have developed an automation framework that provides an experimental environment to determine these parameters and to measure the different penalties associated with such redundancy insertion.

To demonstrate the effectiveness of our methodologies and automation framework, mission-critical systems are designed and analyzed on 2D programmable crossbar-based molecular nanofabrics. We have chosen crossbars since they are reconfigurable architectures well-suited for implementing defect- and fault-tolerance. Probabilistic defect maps are generated for specific nanofabrics and designs are mapped onto these defect-mapped nanofabrics with adequate redundancy to mask permanent and transient faults. Our defect-mapping and hierarchical redundancy insertion methodologies are analyzed independently in terms of different performance measures with special emphasis on reliability, to clearly show the benefits of each. In the next subsection, we discuss the main contributions of our work.

1.1 Main Contributions

Probabilistic Defect-Mapping Mechanism

Mishra and Goldstein [21] define *recovery* as a quality metric that shows the percentage of defect-free components that are identified by a defect-mapping algorithm, and points out that recovery depends on the type of test-circuits, the number of test-circuits run and the post-testing analysis. This means that for achieving high recovery values, the computational complexity of on-line defect map generation techniques maybe unacceptable. This is why we have developed a methodology that uses simple test-circuits from [21] to determine approximate defect rates of the structural primitives of the nanofabrics, and uses these approximate values to compute the failure distributions of the nodes and interconnects that are composed of these primitives. These failure distributions and our non-deterministic *broadcast-based* algorithm are then applied to probabilistic models of these nanofabrics to locate the defective nodes and generate the defect maps. Since this broadcast-based simulation is done by an external system (off-line), we can avoid the time-consuming on-line testing-based defect location phase of the defect-mapping technique in [21].

The broadcast-based defect-mapping algorithm we use is a modification of the algorithm proposed in [24]. In [24], the on-line defect map generation algorithm employs a variation of the reverse path forwarding (RPF) algorithm often used for broadcast routing [8] in computer networks. But the defect model used in [24] is deterministic, hence, is non-ideal for self-assembled molecular nanofabrics. We have modified this methodology such that probabilistic defect maps can be generated that can truly reflect the non-deterministic defects and faults in molecular nanofabrics.

Since a predominant part of our methodology could be externally run on coprocessors suitable for high-speed floating point computations, we claim that the computational time associated with our limited on-line testing and off-line broadcast-based defect-mapping methodology will be less than generating fully on-line defect maps [21,24]. Note that Sect. 3 points out the novelty of our approach and highlights the differences with the approaches in [21, 24].

We have evaluated the performance of our non-deterministic defect-mapping mechanism in terms of recovery for different (1) nanofabric configurations, (2) failure probabilities of the crossbar switches, (3) R at the crossbar level, and (4) via configurations. We have also computed the expected latency of our methodology to show the effect of varying nanofabric sizes and defect distributions on the broadcast latency.

Hierarchical Redundancy Insertion Methodology

The basis of the hierarchical logic mapping approach in [16] is the translation of data flow graphs (DFGs) of systems to graphs composed of fine-grained

primitive data flows (limited behavior). Jacome et al. (2004) [16] also assumes that the basic structural units, or processing elements (PEs), are homogeneous and have limited functionalities, hence, the fine-grained computational behaviors or data flows can be directly mapped to the PEs.

Redundancy can be added to the behavioral primitives, i.e., at the application level and/or to any of the structural units of the molecular nanofabrics. Our hierarchical redundancy insertion methodology extends the design methodology in [16] by generating *pseudo-random* maps (Sect. 3) that translate the non-redundant or redundant DFGs of systems onto the structurally redundant nanofabrics. These maps introduce redundant structural units at the different architectural levels of the nanofabrics and at the same time attempt to reduce routing latencies.

We have analyzed the performance of our redundancy insertion methodology by determining the trade-offs associated with redundant designs mapped onto defect-mapped nanofabric models. Some of the trade-offs computed for these designs are: (1) Reliability-Redundancy, (2) Reliability -Redundancy-Delay at different architectural levels, and (3) Reliability-Redundancy-Cost. These trade-off computations help in analyzing our redundancy insertion methodology.

Toolset

We have developed a toolset to design fault-tolerant molecular systems and analyze the quality of the defect maps and the fault-tolerant designs. Our toolset

1. Integrates both our probabilistic defect-mapping and hierarchical redundancy insertion methodology,
2. Models any arbitrary molecular nanofabric and any number of vias,
3. Analyzes the quality of the defect maps,
4. Analyzes performance measures of behaviorally and/or structurally redundant systems, and
5. Is extensible and user-guided.

1.2 Organization

This chapter is organized as follows: Sect. 2 introduces the crossbar architecture, our target nanofabric and fault models, the software package SMART and the probabilistic design paradigm proposed in [16]. We discuss our fault-tolerant design methodology and automation framework in Sect. 3. Section 4 analyzes our defect-mapping mechanism and reports interesting performance measures of redundant designs mapped to defect-mapped molecular nanofabrics. Finally, Sect. 5 provides some concluding remarks.

2 Background

2.1 Crossbar Architecture

The two-dimensional (2D) crossbar architecture is a general approach to integrate molecular devices. Some of the reasons for the crossbar being one of the more dominant nanoarchitectures are: (1) simple and homogenous layout – it involves only two sets of aligned and perpendicular wires with molecular switches formed at the junction point of the wires, (2) easy integration with microscale addressing circuitry, and (3) defect-tolerance due to structural redundancy. One of the ways to enhance the inherent defect-tolerance of this architecture is by increasing the number of rows and columns of the crossbar, hence, increasing the number of molecular devices or junction points [27].

Molecular diode-based crossbars are not sufficient for building complete systems since the output signals suffer from degradation, making the cascading of many crossbar stages challenging. In this chapter, we use a crossbar architecture that uses molecular latches [7] and a variant of the conventional diode logic. Such an architecture (Fig. 2) has been used in [7] to build complete systems. Logic values held in such latches are encoded using impedance – an unusual characteristic of hysteretic resistor-based latches [7].

The probability that a column wire can be used to form a k-input gate (k rows) is $(1-p)^k$, if each junction has an independent probability of failure p. The probability that at least one column out of N columns will be able to implement the k-input gate is $P_{gate}(k,N) = 1-(1-(1-p)^k)^N$. In this

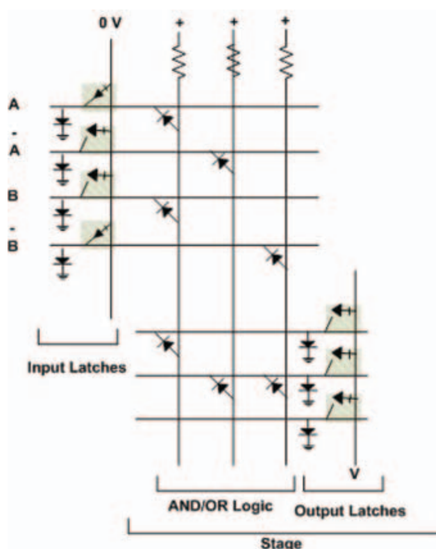


Fig. 2. A crossbar with molecular latches built from hysteretic resistors

chapter, the crossbar is configured to implement a logic function as a combination of 2-input gates. Thus, the probability of a crossbar composed of R 2-input gates functioning correctly is $P_{crossbar} = (P_{gate}(2, N))^R$. Test-circuits are configured on such crossbars to deduce approximate failure probability of the crossbar, and the individual junction failure probability (p) is computed from this equation.

2.2 Nanofabric and Fault Models

Nanofabric model. Our target nanofabric model is composed of PEs built from crossbars such as the ones shown in Fig. 2 and is similar to the model in [24]. Figure 1 shows a nanofabric composed of a grid of PEs connected together. We have modeled the PEs to function as simple single-bit Arithmetic Logic Units (ALUs), 8-bit adders or 8-bit combinational multipliers depending on the application being targeted. In our model, each PE has four transceivers that are connected to the four asynchronous bi-directional interconnects. Figure 1 also shows vias that are used to interface the nanofabric with external circuitry. Our nanofabric model has three levels of hierarchy: regions, mapping units (MUs) and components, whereas, the model used in [24] is monolithic. We will discuss this hierarchy in details in Sect. 2.4.

Fault model. Our fault model considers the effects of both manufacturing defects and transient faults. Since this work focuses on crossbar-based nanofabrics, we discuss the faults relevant to such nanofabrics. The manufacturing defects in the crossbar cause stuck-open and stuck-closed faults at the junctions and wires. It is possible to bias the chemical self-assembly process to decrease the probability of stuck-closed faults – faults which significantly reduce the crossbar yield [12]. Hence, we consider only stuck-open faults at the junctions and wires. Our methodology models these faults as the probability of failure to program the molecular diodes or latches to the appropriate logic value. Also, the faults at the interconnects are modeled as Gaussian failure distributions to quantify the effects of signal noise – flipping the correct signal value. It is also assumed that the faults at the junctions and interconnects are distributed independently, identically and are unclustered (henceforth called i.i.u.). This fault model can be easily changed in our toolset and the current model has been used to represent the fault classes that have the highest probability of occurrence in chemical self-assembly fabrication processes.

2.3 SMART Overview

The Stochastic Model checking Analyzer for Reliability and Timing (SMART) [26] is a tool that can be used to model and analyze complex probabilistic systems. As in traditional model checking, the logical behavior of the system can be analyzed by modeling the system as a probabilistic state machine, generating the state space and asking temporal logic queries about its

dynamic behavior. The tool implements both numerical solution algorithms and discrete-event simulation techniques and can integrate different high-level logical and stochastic modeling formalisms such as Petri Nets, discrete time Markov chains (DTMC), etc.

We use SMART to (1) build molecular nanofabric models that are prone to defects, (2) generate probabilistic defect maps by analyzing these models, and (3) analyze redundant designs that are mapped onto these nanofabric models in the presence of transient faults. We have used SMART's state space partitioning capabilities in our models to alleviate the computational complexity associated with analyzing these models. The DTMC-based modeling formalism is used, since DTMCs are suitable for the analysis of digital circuits [25].

2.4 Background on Hierarchical Mapping Methodology

Jacome et al. [16] proposes a hierarchical approach to map logic onto reconfigurable nanofabrics. The authors show that this approach enhances the scalability of mapping large designs onto dense nanofabrics. This methodology is based on decomposing a nanofabric into a structural hierarchy shown in Fig. 3, decomposing designs into smaller logic functions and hierarchically mapping these designs. We extend the methodology in [16] to allow hierarchical insertion of redundancy in nanofabrics with delay and cost constraints.

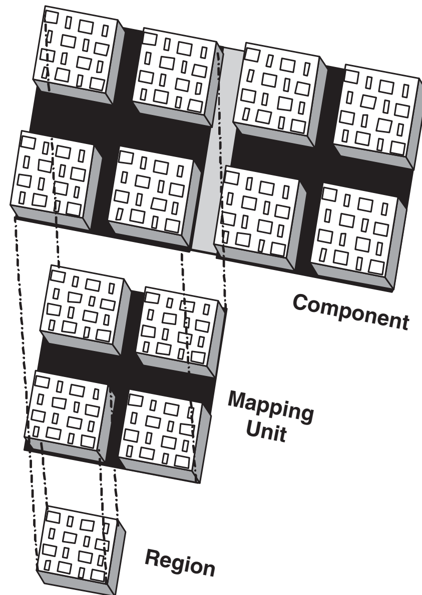


Fig. 3. Three level design hierarchy from [16]

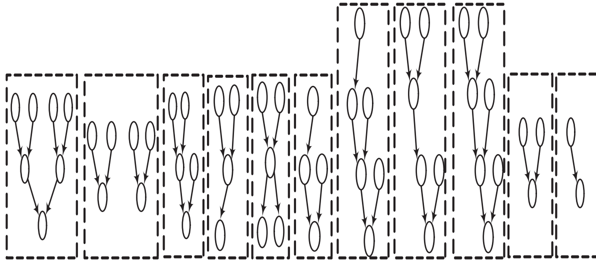


Fig. 4. The set of primitive data flows from [16]

Let us discuss the methodology in [16] in brief. The lowest tier of the design hierarchy is composed of *regions*, that are comprised of eight PEs and the same number of switching elements (SEs). The PEs and SEs can either be crossbars (Fig. 2) or more complicated nanoBlocks [12]. Logic is configured on the PEs, while the SEs are used for interconnections. These regions are the basic configurable blocks (structural primitives) and can be used collectively to form *mapping units* (MUs), and these MUs can be grouped together to form *components*. Jacome et al. [16] also identifies 11 primitive functional data flows (behavioral primitives), shown in Fig. 4, that can be directly mapped to regions. DFGs of different designs are translated to DFGs composed of such data flows, and these DFGs are called *covers*. This is a sort of behavioral decomposition of the designs such that each primitive data flow is configured on a single, limited functionality region. These flows are instantiated on regions if the probability of successful configuration is high. The probability of such successful configuration is estimated by applying Monte Carlo (MC) simulations. In this chapter, we consider our nanofabric models to have the same structural hierarchy and represent designs that need to be mapped to these nanofabrics as covers – DFGs composed of the primitive data flows.

3 Our Probabilistic Design and Analysis Methodology

There are two distinct methods [5] that can be used to analyze the reliability of circuits: generalized or instance-based. The *generalized* approach entails the combinatorial modeling of circuits without considering specific failure distributions at the inputs, gates and interconnects. The output probability distribution of a circuit is computed through combinatorics under the assumption that each gate can fail independently. Thus, the reliability is evaluated in stages using conditional probabilities. Generalized techniques to compute the reliability of large circuits require complex combinatorial reasoning, and re-using the analysis of sub-circuits in the analysis of a larger circuit is difficult. Since specific input probability distributions are not considered during analysis, the generalized approach determines either the circuit's lower or upper bound on the reliability.

Several *instance-based* methodologies have been proposed recently [3, 18, 23]. Instance-based reliability circuit analysis uses probability distributions on the primary inputs as well as gate and interconnect failure probabilities to develop an instance of the circuit. Each instance is then transformed into probabilistic circuit models. This method computes the exact reliability of the circuit for a specific primary input distribution. The main drawback of these methodologies is that several instances of the circuit need to be analyzed to predict performance trends, which can be computationally expensive.

We have developed an instance-based methodology to design and analyze molecular nanosystems. This has been done by: (1) developing a script to translate molecular nanofabric specifications in terms of the number of PEs, each PE's crossbar, etc. into probabilistic transition models; (2) determining the exact failure probability of each junction by running test-circuits physically on the nanofabrics and using these values and a broadcast-based mechanism on the nanofabric models to generate probabilistic defect maps; (3) developing a script to insert behavioral redundancy in the designs and/or structural redundancy at the different structural hierarchies of the nanofabrics; and (4) developing Markovian and state space traversal techniques to analyze redundant instances of different designs that are mapped onto defective or defect-mapped nanofabrics. Our methodology computes the exact reliability of specific instances of the nanofabrics and the designs mapped onto them, hence, is plagued by the drawback of computational complexity that also affects others instance-based approaches. We have addressed this issue by hierarchical modeling and state space partitioning techniques discussed later in this section.

Our methodology can be applied for the design and analysis of mission-critical systems and systems that can tolerate certain degrees of failure as shown in Fig. 5. If the system is mission-critical and needs a reliability guarantee that is arbitrarily close to 100%, we can use our probabilistic defect-mapping mechanism with high reachability threshold values – implying that each PE needs to be accessed from one of the vias with very high probability. PEs that are not reachable are marked defective and are not considered while mapping the system onto the nanofabric. The non-defective nodes are

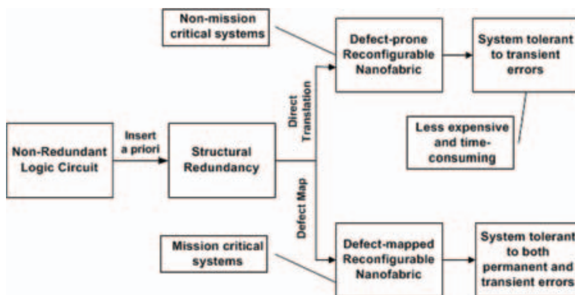


Fig. 5. Application of our methodology

partitioned into regions, MUs and components, and the DFG of the system is mapped to the defect mapped nanofabric model with suitable redundancy. If the system can tolerate low but non-zero failure probabilities, it may not be required to generate defect maps for the nanofabric. Instead, such a system can be mapped onto the defective nanofabric with suitable redundancy. This will help in avoiding the defect-mapping phase, but still allow the design of a fault-tolerant system with less computational overhead. In both of the system types, the value of R and architectural level for redundancy insertion is dependent on the system and its operational environment, hence, determining these parameters is experimental. We use our toolset to determine whether these parameters provide the necessary reliability without causing unacceptable delay, area and cost penalties. The next subsections elaborate on our instance-based methodology.

3.1 Test-Circuits

Our defect-mapping technique is based on limited on-line testing of nanofabrics. We configure simple test-circuits on the nanofabrics to get a notion of the probability of the crossbar-based PE being defective. One of the easiest test-circuits proposed for such purposes is a counter circuit [21]. These counters indicate the number of defective PEs up to some threshold t. Since the lowest element of our structural hierarchy is a region (composed of eight PEs), t for such a counter circuit may be set to 8.

Another set of simple circuits that can be used for this purpose are linear feedback shift registers (LFSRs). LFSRs are used widely in built-in self test (BIST) of RAMs. These can be configured on MUs and run autonomously with sets of inputs. Parallel Signal Analyzers (PSAs) are also configured on the MUs. These PSAs are used as parallel-to-serial compression circuits to avoid testing a large number of pseudo-random vectors generated by the LFSRs. The compression of a number of patterns using a PSA is called a signature. If there are signature mismatches, the LFSRs may be split into smaller units and configured on each region of the specific MU instance. Note that even if there are no signature mismatches, it does not mean that the constituent nodes are defect free. This is due to a non-zero probability of aliasing. A way to minimize this aliasing problem is to use maximal-length PSAs and to frequently compare the signatures with the expected values [17].

The test-circuits discussed here are just a small subset of the different test-circuits that can be used to compute the probability of PEs being defective. The crossbar junction failure probability can be computed using these probabilities and the equations in Sect. 2.1. Note that determining the junction failure probability is required for both our defect-mapping and structural redundancy insertion methodologies, since broadcasting and other logic functions need to be mapped to the crossbar-based PEs, respectively.

3.2 Defect-Mapping Technique

We have developed a non-deterministic defect-mapping technique based on a variant of the RPF broadcast scheme [24] and on the defect-mapping methodology proposed in [21]. We configure simple test-circuits from [21], also discussed in Sect. 3.1, physically on the nanofabrics to determine the probability of the molecular junctions being defective. These defect rates are used in determining the probability of successfully configuring all PEs with the same broadcasting functionality. Once this configuration probability is computed, probabilistic models representing such PE-based nanofabrics are constructed and our broadcast-based algorithm is used to determine the reachability graph of each PE from the vias.

The broadcast-based algorithm we use is a modification of the deterministic algorithm proposed in [24]. We have modified this algorithm such that defect maps represented by probabilistic broadcast trees can be generated, hence, reflecting the non-deterministic defects in molecular nanofabrics. In our algorithm, the probability of a PE being reached depends on the probability of the packet reaching the previous PE that forwarded the packet and the probability of the interconnect on which the packet was transmitted being defect free. Our methodology also stores the via number from which a PE is reachable with highest probability, since this facilitates the configuration of the fabric at a later stage if there are more than one available via. Since our methodology applies this modified broadcasting algorithm on a nanofabric model, we mimic the dynamics of the physical broadcast in our model. Note that the broadcast-based part of our defect-mapping technique is run off-line, i.e., on an external system.

Our methodology also partitions the nanofabric models to decrease the computational complexity of our off-line probabilistic broadcasting algorithm. A molecular nanofabric is partitioned into equal-sized smaller units such that the algorithm can be run simultaneously on all the units. This partitioning implies that in certain cases some of the PEs may be a part of more than one unit and hence can be reached from more than one via, an algorithmic feature that increases the reachability of these PEs. Our defect-mapping mechanism pin-points the via from which a specific PE can be reached with the highest probability. If the highest probability value of reaching a PE from any of the vias is lower than a user-specified threshold, that PE is marked defective.

Our defect mapping technique forms a generalized probabilistic model for a specific molecular nanofabric configuration and generates a defect map for that particular configuration. Whereas, on-line methodologies [21, 24] have to be applied physically to each fabricated molecular nanofabric, even if their configurations are the same. Hence, we claim that the computational time associated with our limited on-line testing and generalized off-line broadcast-based methodology will be less than generating fully on-line defect maps for each molecular nanofabric.

At this point, it is pertinent to provide a detailed discussion on the differences between our methodology and the methodologies in [21, 24]. These are as follows:

- Mishra and Goldstein [21] uses extensive on-line testing to improve the recovery metric, whereas, our methodology only uses limited on-line testing.
- Patwardhan et al. [24] proposes a deterministic broadcast algorithm that has been extensively modified to a non-determinsitc broadcast algorithm.
- Patwardhan et al. [24] generates on-line defect maps by physically broadcasting test packets in the nanofabrics, whereas, we mimic such broadcasts in our nanofabric models.
- Our methodology uses state space partitioning techniques on the nanofabric models to alleviate off-line computational complexity. This was not considered in either [21] or [24] since the problem of physically partitioning nanofabrics is intractable.

3.3 Hierarchical Redundancy Insertion Methodology

The problem of mapping logic onto ultra-dense nanofabrics has been handled well in [16]. We have used the same procedure to map covers (2.4) of different designs onto the structural hierarchy of our nanofabric models. Since there can be more than one way of decomposing a generic DFG of a system into a cover, there may be more than one cover for a system. Figure 6 shows one of the possible covers for an Auto Regression (AR) filter hardware design.

We have extended the methodology in [16] so that behaviorally redundant systems can be mapped onto structurally redundant molecular nanofabrics.

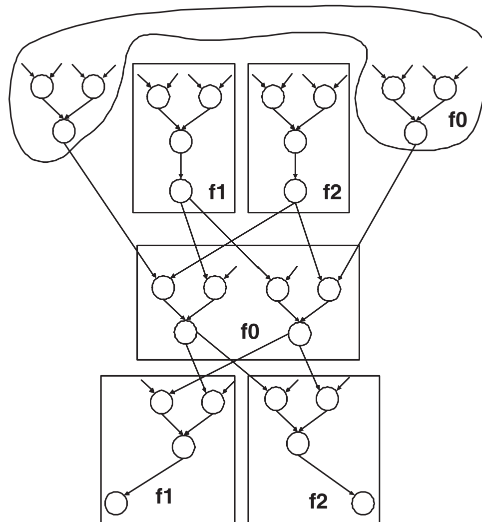
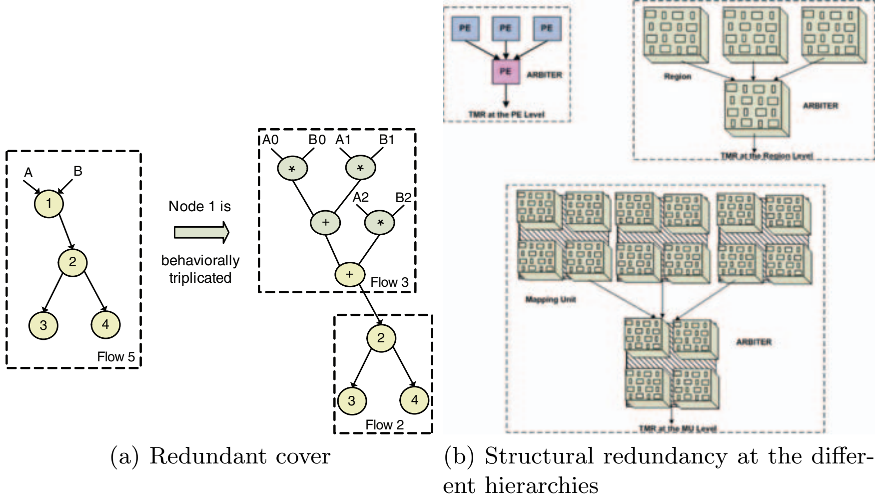


Fig. 6. Cover for AR filter



(a) Redundant cover

(b) Structural redundancy at the different hierarchies

Fig. 7. Behavioral and structural redundancy

Redundancy can be added to the covers by replacing original computational nodes by primitive data flows that represent some form of behavioral redundancy. For instance, Fig. 7a shows how a non-redundant cover with a single data flow changes to a cover with two data flows when node 1 is behaviorally triplicated. The logic function of a Triple Modular Redundancy ($A0 \times B0 + A1 \times B1 + A2 \times B2$) replaces the original $\times$ function at node 1. Similarly, structural redundancy can be inserted either at the crossbar level by adding columns (2.1) or at any of the three tiers of the nanofabric hierarchy. Figure 7b shows Triple Modular redundancy (TMR) configurations [25] for the different structural hierarchies. The TMR configurations are representative examples of behaviorally and structurally redundant fault-tolerant configurations. Our hierarchical redundancy insertion methodology maps the non-redundant or redundant covers of the systems onto the structurally redundant nanofabrics, by generating *pseudo-random* maps.

These pseudo-random maps (1) avoid defective components if and only if a defect map is available for the nanofabric, since a designer can opt not to generate a defect map; (2) introduce redundant structural units at the different hierarchies of the nanofabrics; and (3) attempt to reduce routing latencies while inserting structural redundancy. We have developed scripts that use these maps to hierarchically translate each primitive data flow of the covers onto the structurally redundant nanofabric models. We emphasize hierarchy since the computational time required for steady state probability analysis of hierarchically mapped designs is less when compared to flat models [4]. Such a specific structural mapping of a cover is called a *cover-map* for a hardware design.

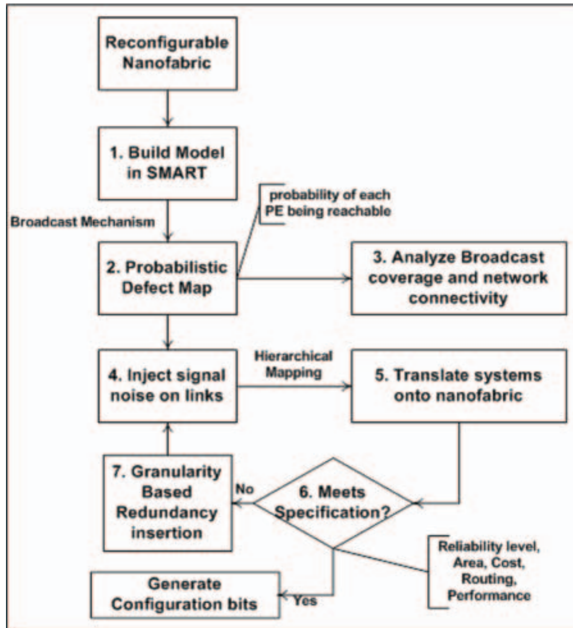


Fig. 8. Design flow

3.4 Design Flow

Figure 8 shows the design flow of our design and analysis methodology. The different steps are outlined in details below:

1. Test-circuits would be run physically on the nanofabric to compute the approximate junction defect rate. A model of the specific molecular nanofabric is built in SMART by creating an array of PEs and interconnects. Since each PE needs to have some functionality to support our broadcast-based algorithm, our nanofabric model requires that each PE must have enough logic functionality to support four transceivers, a simple single-bit ALU and control circuitry for routing. The junction defect rate is used to compute the probability of the PEs being successfully configured to achieve such minimal functional capabilities.
2. Our non-deterministic broadcast-based algorithm (Sect. 3.2) is used to generate a defect map, if the designer opts to generate one.
3. If a defect map is generated, the designer can analyze the performance of our defect-mapping scheme in terms of the broadcast coverage and recovery.
4. We have developed a transient fault injection library to model signal noise at the interconnects. This library can be used to model noise as Gaussian failure distributions with different means and variances.

5. We use the hierarchical logic-mapping scheme proposed in [16] and our pseudo-random map to translate the non-redundant or redundant cover (Fig. 7a) onto the nanofabric model composed of regions, MUs and components.
6. The mapped design is analyzed to check conformance to required specifications in terms of reliability, cost, latency and area. If the specifications are met, the system is physically configured onto the nanofabric.
7. If the specifications are not met, structural redundancy is added at the different hierarchies of the nanofabric by using our redundancy insertion methodology discussed in Sect. 3.3.

4 Experimental Results

In this section, we experimentally analyze our design methodology by translating behaviorally redundant or non-redundant mission-critical systems onto defect-mapped molecular nanofabrics. We have also inserted structural redundancy at the different nanofabric hierarchies to demonstrate the effect of varying R and redundancy insertion level on the area, cost and delay parameters.

All experiments have been run using SMART 1.1 [26] on a Dell workstation running Linux with dual 3.4 GHz microprocessors and 4 GB of RAM. As discussed in Sect. 3.1, our methodology entails determining the junction and latch defect rates by running test-circuits. But for the sake of analyzing different scenarios, we have varied the junction defect rate in all the experiments and fixed the defect rate of molecular latches to 0.1. Also, we have modeled four vias at the corners of the nanofabrics and assumed that if any PE is reachable with a probability of less than 0.9 from these vias, it is marked defective. Since our framework is parameterized, such reachability thresholds can be varied depending on the level of reliability that has to be guaranteed for the system. We have used benchmark image and digital processing systems in our experiments. The computational nodes of the covers for these systems are either 8 bit adders or 8 bit multipliers, hence, the crossbar-based PEs need to be configured with such logic functionalities.

We show the execution time in Table 1 required to generate a defect map for a target nanofabric model, translating each non-redundant system onto the nanofabric model, and determining different performance measures of each design. All execution times are averaged over 1,000 runs. Specifically, each run includes the time to (1) build a model of a 100×100 grid of PEs and interconnects; (2) determine the probability to reach each PE from the different vias; (3) partition the non-defective PE-based model into sub-models based on regions, MUs and components, and (4) construct a pseudo-random map to translate a specific system onto a nanofabric model and compute the different design trade-offs. As can be seen from Table 1, these execution times for all of the hardware designs is reasonable. The analyses of median sort and Sobel

Table 1. Execution time for the design and analysis of systems

Systems	Cover	Map	Time (s)
Auto regression	1	2	2054.83
Discrete cosine	1	2	2047.90
FIR	2	3	2053.50
Avenhous filter	2	1	2049.25
Avenhous filter mod	1	2	2048.88
Median sort	1	2	3400.01
Sobel edge detection	1	2	3690.33

edge detection systems need relatively higher execution time since the state space generated by these models are relatively large.

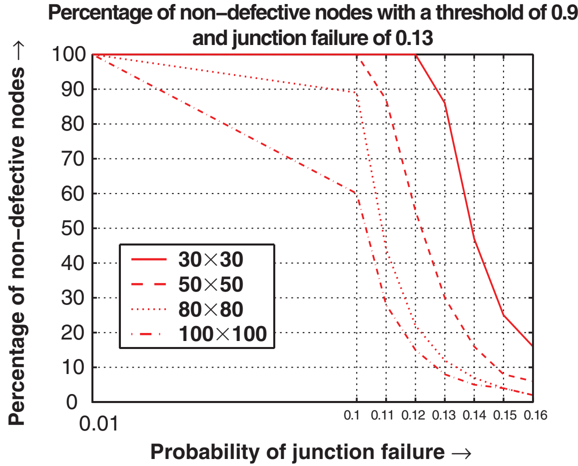
It has also been observed from experimental results that as the nanofabric size and the number of vias increases, the execution time of our design and analysis methodology increases. This is intuitive since the probabilistic computation becomes more involved as the nanofabric size increases. Since our methodology is parallelizable, high performance parallel computing solutions may exist that can address this execution time problem. Currently, we are looking at efficient ways of parallelizing our methodology and using high performance computing solutions. We present the analysis of our defect-mapping and hierarchical redundancy insertion methodologies separately in the next subsections.

4.1 Analyzing Defect-Mapping Technique

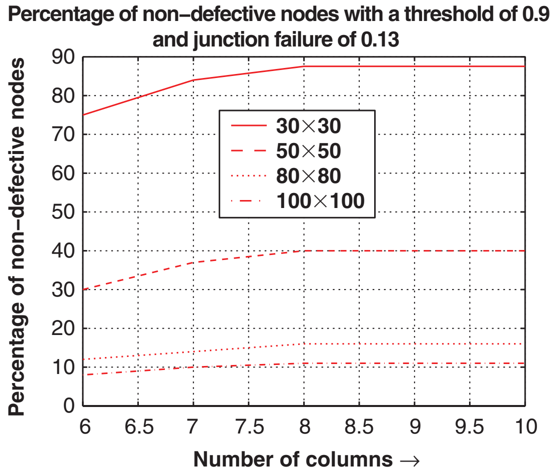
In this subsection, we analyze the performance of our probabilistic defect-mapping technique. We evaluate the defect-mapping scheme by varying (1) the number of PEs in a nanofabric model, (2) the junction defect rate, (3) the number of columns of the crossbar (R), and (4) the number of vias. We also compute the expected latency of the probabilistic broadcast to quantify the delay that is associated with our technique.

Varying junction defect rate. Figure 9a indicates that *smaller nanofabrics have higher tolerance to junction failure, given that the number of vias is a constant.* As the nanofabric size increases, the broadcast distances from the vias increase. Since each PE is assumed to fail independently with the same failure probability, the likelihood that a broadcast packet reaches a PE located far away from the via decreases. For this experiment, we increased the number of vias from 4 to 5 and observed that the percentage of reachable (non-defective) PEs increased slightly due to the addition of the fifth, center via (Fig. 1), since some of the PEs became more accessible from the center via. Hence, the number of vias need to be increased as the nanofabric size increases.

Further, for large nanofabrics, the percentages of non-defective nodes at higher junction defect rates are almost the same. This can be seen from the



(a) Varying junction defect rate



(b) Varying crossbar R

Fig. 9. Percentage of non-defective PEs for different nanofabric sizes

plots for the 80×80 and 100×100 nanofabrics and for junction failure probabilities above 0.15. This can be inferred as follows: for high junction defect rates, the probability of reaching PEs in a large nanofabric decreases steadily until almost all the PEs become unreachable, i.e., the reachability threshold of 0.9 in this experiment is not met. Similar results are also observed when junction defect rates are varied between 0.01 and 0.16 for five vias and for nanofabric sizes up to $10^4 \times 10^4$.

Varying number of columns. In this experiment, R for the molecular crossbars is varied (crossbar size). The junction defect rate is fixed at a probability of 0.13. The major observation from Fig. 9 (b) is as follows: *for a specific*

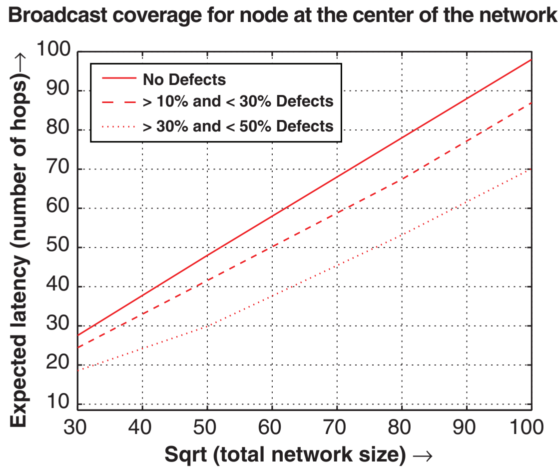


Fig. 10. Broadcast latency

nanofabric configuration, increasing the number of columns of the crossbar results in a higher percentage of non-defective nodes. This improvement reaches a steady state at a particular point (number of columns) beyond which increasing R does not increase the percentage of non-defective nodes. This observation can be interpreted as follows: for a junction failure probability of 0.13, the probability of successfully configuring broadcasting functionality on each PE saturates at a certain R of the crossbar. Thus, the number of PEs that can be reached reliably stabilizes and cannot be improved by adding more number of columns. A similar observation has been made for nanofabrics with five vias and sizes up to $10^4 \times 10^4$, for different values of R.

Broadcast latency. We have computed the expected latency of our scheme by modeling the dynamics of broadcasting test packets physically. For a nanofabric with four vias, the maximum broadcast latency is for the center PE. Figure 10 shows the expected latency for the center PE in terms of the number of hops and plots it as a function of the square root of the number of PEs. The plots show that, *for a nanofabric with no defects, $> 10\%$ and $< 30\%$ defects, or, $> 30\%$ and $< 50\%$ defects, the expected latency to reach the center PE from one of the vias is almost a linear monotonically increasing function of the square root of the nanofabric size.* It is also observed that as the number of defective PEs increases, the expected broadcast latency decreases, since fewer PEs are reachable with probability values within the reachability threshold and the algorithm does not need to be process PEs beyond the reachable PEs in the system.

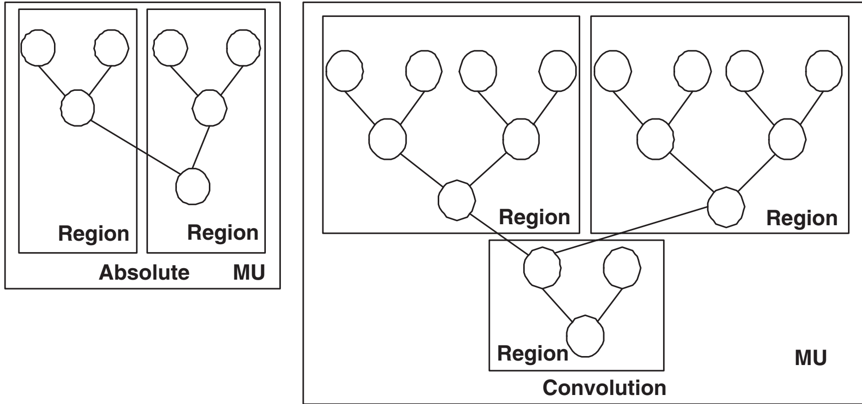
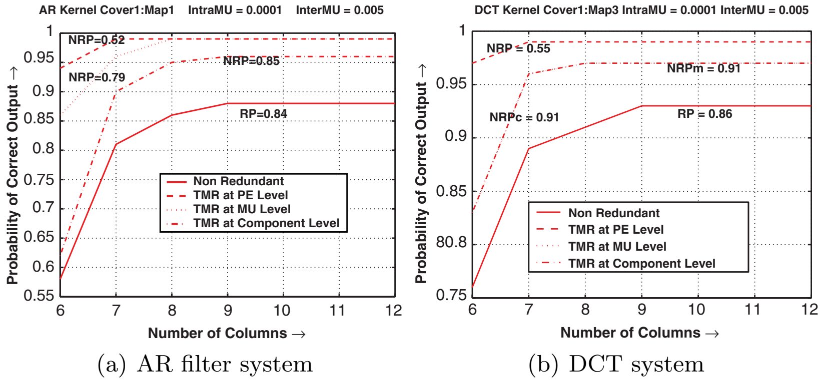


Fig. 11. Sobel edge detection cover-map combination

4.2 Analyzing Hierarchical Redundancy Insertion Methodology

In this subsection, we analyze our hierarchical redundancy insertion methodology by experimenting with the systems outlined in Table 1. Different cover-map combinations and structural redundancy-based techniques are considered for these experiments. The pseudo-random map generation technique (3.3) is used to translate these systems onto defect-mapped nanofabric models and our toolset is used to compute different design trade-offs for these mapped systems. We present results for the AR filter and 2D-Discrete Cosine Transform (DCT) systems to illustrate the influence of such analysis on fault-tolerant system design.

To compute the delay associated with redundancy insertion, we use the metrics of relative performance (RP) and normalized relative performance (NRP) for each cover-map combination of the two systems. We will discuss how these metrics are computed for each system. Each PE is assumed to take 2 cycles to complete an operation and route it to an adjacent PE. Thus, if CP is the critical path length of a system, $CP_{delay} = 2 \times CP$ is a lower bound on the delay for any cover-map combination representing a system [16]. For instance, it can be seen from Fig. 11 that the CP has eight PEs in the Absolute and Convolution covers, hence $CP_{delay} = 16$. RP is computed by taking the ratio of the CP_{delay} and actual delay that is dependent on the cover-map combination, i.e., the structural mapping of the cover. The actual delay includes intra-MU and inter-MU routing delays that are considered to take 1 and 2 cycles, respectively. Thus, a high RP implies a lower delay overhead. NRP is computed by taking the ratio of the RPs of any cover-map and the cover-map combination that has all the flows mapped to a single MU (least routing delay). A higher NRP similarly indicates better performance for a cover-map combination.

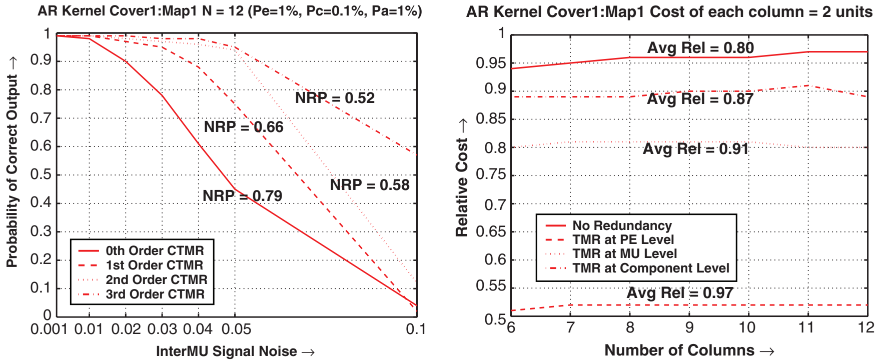


(a) AR filter system (b) DCT system

Fig. 12. Reliability-redundancy-delay trade-offs

Also, to determine the cost of inserting redundancy in systems, we develop a cost metric, relative cost (RC) for each cover-map combination. RC consists of the cost associated with fabricating crossbar-based PE, intra-MU and inter-MU routing lines. Routing within regions is assumed to be free. It is assumed that the cost of each molecular crossbar column is 2 units. Similarly, intra-MU and inter-MU routing tracks cost 2 and 4 units, respectively. The RC for any non-redundant system is computed as the ratio of the cost associated with the CP and the actual cost of the cover-map combination. The RC for any redundant system is computed as the ratio of the least cost non-redundant cover-map and the cost of the redundant cover-map. A higher RC value implies a more cost-effective design. We have also computed the average reliability of both the non-redundant and redundant cover-maps for both the AR filter and DCT systems. This metric is the mean reliability level, determined by varying the number of crossbar columns (redundancy at the PE level).

Non-redundant and TMR-based designs. Figure 12a shows the probability of correct output for the AR filter for different numbers of crossbar columns. Signal noise at the intra-MU and inter-MU routing tracks are assumed to be Gaussian failure distributions with means 0.0001 and 0.005 and variances 0.001 and 0.01, respectively. The plots in Fig. 12a indicate the reliability-redundancy-delay trade-offs for a specific non-redundant AR filter cover-map and its TMR configurations at the PE, MU and component levels. One of the major observations from these plots is: *TMR at the PE level (maximum redundancy) yields high reliability when the number of columns is small. But once the number of columns reaches* 8, *TMR at the MU level provides similar reliability with less redundancy. Also, the NRP values indicate that the performance penalty for TMR at the PE level is very high as compared to TMR at the MU level.* Hence, for this specific fault distribution and cover-map, if the number of crossbar columns for implementing a 2-input logic function is greater than or equal to 8, TMR at the MU level is the better choice.



(a) Reliability-redundancy-delay trade-offs for different MU-based CTMR configurations

(b) Reliability-redundancy-cost trade-offs for non-redundant and TMR configurations

Fig. 13. Trade-offs for different redundant configurations for the AR filter

Such non-intuitive inferences can also be drawn for the DCT system from Fig. 12 (b). In this case, the pseudo-random map generated for the system places all of the behavioral flows in one MU (higher probability of failure but lower delay). TMR at the component and MU levels provides equivalent reliability and performance (as indicated by NRP_c and NRP_m), since the TMR configurations are equivalent.

Cascaded triple modular redundancy (CTMR). We have computed the trade-offs for CTMR configurations for the AR filter cover-map combination used previously by varying the failure probability of the inter-MU routing lines and keeping the number of crossbar columns, N, fixed at 12. The probability of failure of the PEs, SEs and majority voters or arbiters are computed to be $P_e = 1\%, P_c = 0.1\%, P_a = 1\%$ when these are configured on the crossbars with 12 columns and junction defect rate of 0.1. The plots in Fig. 13 (a) indicate that as the MU-based CTMR orders increase, the reliability of the architecture improves and the delay penalties increase (intuitive). But the major observation is: *when the inter-MU routing line is affected by a failure distribution due to signal noise that is centered around* 0.1 *with a variance of* 2*, the 0th order CTMR (TMR) seems to do better than 1st order CTMR. Although, it is observed that the 2nd and 3rd order CTMRs for this cover-map provides marginal improvements in reliability as compared to the 0th and 1st order CTMRs, they also entail substantial degradation. Hence, TMR is the best choice for this cover-map and fault distribution.*

Reliability-redundancy-cost trade-offs. The other basis for analyzing redundant systems is analyzing the implementation cost. Although the cost units in this chapter are artificial, they can be modified easily in our automation framework. In Fig. 13b, the RC is plotted against different redundancy levels,

i.e., number of columns at the crossbar level. One of the design intuitions indicated by the plots is: *for the TMR configuration at the PE level, average reliability is highest but the implementation cost is very high.*

5 Conclusion

Nanoscale molecular systems will be susceptible to both defects and transient faults. It has been shown that defect-mapping techniques can be used to mitigate defects and structural redundancy may be used to mitigate transient faults. Since defect-mapping is computationally intensive, it would be beneficial to avoid this process for systems that are more tolerant to failures. In this chapter, we propose a unified probabilistic methodology to design and analyze mission-critical systems and systems that allow low but non-zero failure probabilities. Our automation framework can be directed to either generate defect maps or avoid defect map generation and be used to automatically insert structural redundancy. Since redundancy-based nanodesigns may entail penalties in terms of area, cost and delay, our framework can be used to analyze these parameters to quantitatively guide the selection of an acceptable fault-tolerant design.

In this chapter, we have also pointed out the need for developing defect-mapping techniques that truly reflect the non-determinism in nanodevices, yet are computationally viable. We have developed a limited testing and probabilistic broadcast-based defect-mapping scheme that is predominantly run off-line on external systems. Although this methodology is motivated from the rich literature in this area, we have pointed out the differences of our non-deterministic defect-mapping scheme with existing techniques. We have also enhanced a hierarchical logic mapping methodology to incorporate behavioral and structural redundancy insertion.

We have also applied our design methodology in designing image and signal processing systems on target molecular nanofabrics. The computational times for designing and analyzing these systems have been presented and seem to be reasonable even for large designs. Different performance trade-offs such as reliability-redundancy-delay trade-offs have been determined that indicate the effectiveness of our methodology in expediting and guiding fault-tolerant design of such systems.

References

1. A. Aviram and M. Ratner. Molecular rectifiers. *Chemical Physics Letters*, 29(2):277–283, November 1974
2. A. Bandyopadhyay and A. Pal. Large conductance switching and memory effects in organic molecules for data storage applications. *Applied Physics Letters*, 82(8):1215–1217, 2003

3. D. Bhaduri and S. Shukla. NANOLAB—a tool for evaluating reliability of defect-tolerant nanoarchitectures. *IEEE Transactions on Nanotechnology*, 4(4):381–394, 2005
4. D. Bhaduri, S. Shukla, P. Graham, and M. Gokhale. Comparing reliability-redundancy trade-offs for two von neumann multiplexing architectures. *IEEE Transactions on Nanotechnology*, 2006. To appear. Available at http://fermat.ece.vt.edu/Publications/online-papers/Nano/MUX_TNANO.pdf
5. D. Bhaduri and S. K. Shukla. Comparing the reliability-redundancy trade-offs for two von neumann multiplexing architectures. Technical report, Fermat Lab, Virginia Tech, 2005. Available at http://fermat.ece.vt.edu/Publications/pubs/techrep/techrep05_01.pdf
6. D. Bhaduri, S. K. Shukla, P. Graham, and M. Gokhale. Reliability analysis of fault-tolerant reconfigurable architectures. In *NANOARCH*, May 2005. Available at http://fermat.ece.vt.edu/Publications/pubs/techrep/techrep0415.pdf
7. Yong Chen et al. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14:462–468, 2003
8. Y. K. Dalal and R. M. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, 1978
9. A. DeHon. Array-based architecture for fet-based nanoscale electronics. *IEEE Transactions on Nanotechnology*, 2:223–232, 2003
10. A. DeHon and M. J. Wilson. Nanowire-based sublithographic programmable logic arrays. In *Int'l Symp. on FPGAs*, pages 123–132, 2004
11. A. Flood, J. Stoddart, D. Steuerman, and J. R. Heath. Whence molecular electronics? *Science*, 306(5704):2055–2056, Dec 2004
12. S. C. Goldstein and M. Budiu. Nanofabrics: Spatial computing using molecular electronics. In *Annual International Symposium on Computer Architecture (ISCA)*, pages 178–189, July 2001
13. J. Han and P. Jonker. Fault tolerance in nanocomputers: random interwoven redundancy. *IEEE Trans. VLSI*. To Appear.
14. J. Han and P. Jonker. A system architecture solution for unreliable nanoelectronic devices. *IEEE Transactions on Nanotechnology*, 1:201–208, 2002
15. J. Heath, P. Kuekes, G. Snider, and R. Williams. A defect tolerant computer architecture: Opportunities for nanotechnology. *Science*, 80:1716–1721, 1998
16. M. Jacome, C. He, G. Veciana, and S. Bijansky. Defect tolerant probabilistic design paradigm for nanotechnologies. In *DAC*, pages 596–601, June 2004
17. J. Koeter. What's an lfsr? (rev. a). Tech. report, Texas Instruments, 1996. Available at http://www.ti.com/sc/docs/psheets/abstract/apps/scta036a.htm
18. S Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes. Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In *Design, Automation and Test in Europe (DATE'05)*, volume 1, pages 282–287, New York, NY, USA, 2005. ACM Press
19. C.N. Lau, D. R. Stewart, R. S. Williams, and M. Bockrath. Direct observation of nanoscale switching centers in metal/molecule/metal structures. *Nano Letters*, 4(4):569–572, 2004
20. H. M. McConnell. Intramolecular charge transfer in aromatic free radicals. *The Journal of Chemical Physics*, 35(5704):508–515, August 1961
21. Mahim Mishra and Seth Copen Goldstein. Defect tolerance at the end of the roadmap. In *International Test Conference (ITC)*, Charlotte, NC, Sep 30-Oct 2 2003

22. K. Nikolic, A. Sadek, and M. Forshaw. Architectures for reliable computing with unreliable nanodevices. In *Proc. IEEE-NANO'01*, pages 254–259. IEEE, 2001
23. G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla. Evaluating the reliability of nand multiplexing with prism. *IEEE Transactions on CAD*, 24(9). To appear September 2005
24. J. Patwardhan, C. Dwyer, A. Lebeck, and D. Sorin. Evaluating the connectivity of self-assembled networks of nano-scale processing elements. In *IEEE International Workshop on Design and Test of Defect-Tolerant Nanoscale Architectures*, May 2005
25. Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems: Design and Evaluation*. Digital Press, Burlington, MA, 2nd edition, 1992
26. Web Page: http://www.cs.ucr.edu/ ciardo/SMART/, 1994
27. M. B. Tahoori. Defects, yield and design in sublithographic nano-electronics. In *IEEE Symposium in Defect and Fault Tolerance*, 2005
28. Victor V. Zhirnov and Daniel J. C. Herr. New frontiers: Self-assembly and nanoelectronics. *Computer*, 34(1):34–43, January 2001

Index

A

Adaptive Recovery, 35, 44, 54–57
Adder circuit, 1, 10, 13–29
Allocation algorithm, 15–21, 28
AND gate, 9–13
Application-dependent, 127, 128, 148, 149
Application-independent, 122, 128, 129, 148, 149
Arbitrary functional fault (AFF), 186
Arbitrary reversible fault (ARF), 187
Asynchronous logic, 203–204
Auto Regression (AR), 385, 392–394
Auto-model, 320
Average recovery, 108–109, 114–118

B

Bayesian Network, 230–232, 234, 236–239, 243, 248
Behavioral model, 96, 102, 118
Behaviorally redundant, 385, 388
Belief propagation, 319, 322, 329
Benefits of delay-insensitive circuit, 203–205, 207, 209, 210, 221, 223–225
Benign fault, 216
Biclique, 125, 126, 130, 132, 134–140, 145, 147, 148
Biochip, 287–298, 303–307
Biomedical applications, 267, 269
Biomedical assay, 268, 269, 271, 280, 281
BioMEMS, 283
Biosensing, 283
Bipartite graph, 125, 126, 130, 134–136, 138
BIST, 383
Bistable storage element, 322, 327
Blocks under test (BUT), 34, 40–46, 48–50, 52–54, 57, 60, 111
Bottom-up fabrication, 95
Branch merging, 353
Branch pruning, 357, 360
Bridging fault, 46, 47, 49–51
Broad-cast based defect-mapping, 376, 395
Built-in redundancy, 96
Built-in Self-test (BIST), 96, 119, 126

C

CAEN-BIST, 96–99, 101, 103, 105, 107, 109–119
Capacitance matrix, 253
Carbon nanotubes, 120, 121, 123, 124
Catastrophic fault, 271
Cell, cellular array, 203–212, 223–225
Cellular automata, 262
Chemical synthesis process, 33
Chemically-assembled electronic nanotechnology (CAEN), 33–35, 43, 60, 96
Circuit area, 1, 12, 28, 30
Circuit compiler, 1, 6, 9
Circuit layout, 10, 18
Circuit reliability, 5

Circuit shape, 25
Circuits, 227–229, 231, 232, 234, 235, 237–240, 243–246, 248, 249
Clinical diagnostics, 283
Clique energy function, 316, 320–323, 328, 331–333
Clique energy, 316, 317, 320–323, 327, 328, 331– 333
Clique, 316–318, 320–323, 327, 328, 331–333
Clocked QCA, 251
Clocking zone, 162, 164, 167–170, 174–176, 178
Clocking, 255–257, 262
Clustered, 132, 134, 141–144, 146
Clustering parameter, 133
CMOS/nano hybrid system, 363, 364
CMOS-CAEN interface, 103
Complement graph, 138–140, 146
Complete bipartite graph, 125, 134
Computational complexity, 272–275
Computational components, time varying fault rate, 342, 345, 346
Conditional probability, 322
Configuration shift, 110, 111
Configuration transformation, 216
Configuration, 97, 99–101, 104–105, 107–108, 110–111
Conservative, 157, 160
Continuous-flow, 290, 293
Control units, 345–348, 365
Corner nanoBlock, 112–114
Cost, 209
Co-tunneling, 252
Coulomb blockade, 253
Coulomb coupling, 251
Critical path (CP), 392, 393
Crossbar architecture, 1, 6–9, 28, 99, 101
Crossbar, 122–136, 138, 141–149
2D Crossbar, 378
Crosspoint, 35, 38, 46–50, 123–126, 129, 133, 134
C-testability controllability, 159, 160, 178, 179, 183, 185–187, 189, 191, 194, 196, 198, 199
CTMR, 394
C-unit, 347–362, 365–368
Cut edge, 297

D

Decentralized control, 347, 348
Defect correlations, 9, 29
Defect density, 96–98, 101–103, 107–110, 113, 115, 118, 122, 124, 129, 130, 132–135, 138, 141–147
Defect Distribution Model, 132–136, 141, 143, 144
Defect map, 38, 41, 43, 44, 54, 57, 58, 101–102, 107–108, 113–114, 117–118, 122, 126–132, 136, 147, 149, 374–377, 380, 382–388, 395
Defect tolerance, 1, 2, 5, 9, 96–99, 101, 118–119, 124, 126–129, 131, 141, 145, 148, 149, 253, 255, 257, 259, 261, 289, 298, 304, 340, 378
Defect, 121–136, 138, 141–149
Defect-avoidance, 373, 374
Defect-aware place-and-route, 101
Defect-aware, 127–130, 148, 149
Defect-models, 374, 376
Defect-unaware design flow, 122, 126–132, 149
Defect-unaware, 122, 126–132, 141, 148, 149
Delay-insensitive AND gate layout, 210
Delay-insensitive AND gate, 210, 221
Delay-insensitive circuit, 203–205, 207, 209–216, 221, 223–225
Delay-insensitive register layout, 212
Delay-insensitive register, 212
Delay-insensitive XOR gate layout, 211
Delay-insensitive XOR gate, 210, 221
Dependence graph, 317, 318, 321, 331
Design view, 128, 129, 141, 148
Diagnosis, 96–97, 101, 110–112, 118–120
Diagonal algorithm, 108, 109, 112
Differential cascade voltage switch (DCVS) logic, 326, 327, 336
Digital circuits, 315, 335
Digital microfluidics, 288, 307
Diode logic circuit, 6, 9
Diode-resistor logic (DRL), 36, 123
Directed self-assembly, 95
Discrete cosine transform (DCT), 392–394
Double stuck-at-1 fault in a TRIA, 221–222

Droplet, 267–271, 273, 277–284
Drug discovery, 283
Dual-rail 2-to-1 multiplexer, 212, 213
Dual-rail encoding, 204, 205, 209
Dynamic hardware allocation, 353

E

EasyPath, 97, 119
Edge-dependent, 298
Electrode, 288–296, 298, 300, 304–308
Electrode-short fault, 292, 304, 305
Electrowetting, 268
Electrowetting-on-dielectric (EWOD), 292
Error probability, 261
Error rates, 259
Euler circuit, 299–302, 307
Euler path, 299, 300

F

Fault clustering, 117
Fault detection configuration (FDC), 41–50, 52, 53, 55
Fault injection, 106
Fault masking, 182, 183, 190
Fault model, 291
Fault modeling, 34
Fault pattern, 135, 136, 141
Fault rate, 343, 344, 346, 349, 351, 354, 356, 361, 362, 369
Fault tolerance, 322, 340, 342–349, 351, 354, 357, 359–366, 368, 369
Fault tolerant computational model, 347, 360, 365, 366
Fault, 204, 208, 209, 216–225
Fault-equivalent module, 182
Fault-models, 377, 379
Faulty-signal, 217–219
Feedback, 322, 323, 327–329, 335
Field effect transistor (FET), 124
Field Programmable Gate Arrays (FPGAs), 34, 35, 121, 123, 124
Final mapping, 122, 128, 129, 132, 136, 149
Fleury's algorithm, 301, 302
Fork, 206, 207, 216–219
FPGA-BIST, 97–98
Fredkin gate, 160, 163–166, 169, 174, 176, 177, 183, 189–192
Free energy, 253–255, 258

G

Gain, 253–254, 259–260, 262
Gate-level fault, 219–223
Gibbs distribution, 316–318
Global view, 131, 147, 148
Glucose assay, 306, 308
Graph monomorphism, 18, 19
Graph theory, 290, 298
Greedy mapping algorithm, 138–140

H

Hamiltonian path, 271–276, 278, 282
Hammersley-Clifford theorem, 316
Hard repair, 96
Hardware redundancy, 342, 343, 346, 349, 361
Heuristic algorithm, 278–284
Hierarchical mapping, 380–381
Hybrid redundancy, 348, 351

I

Implied dependence, 331, 333
Independent set, 138–140
Indium tin oxide (ITO), 295,
Information propagation, 356, 357
Instruction confirmation, 350
Instruction dependency, 347
Instruction execution, 345, 347, 348, 350, 353, 361, 369
Instruction issue, 346, 348
Integer linear programming, 275
Interaction coefficients, 321
Irreversible fault, 185, 186
Irreversible logic, 157, 159
Iterations, 114–117
Iterative Logic Array (ILA), 160, 178, 186

J

Joint probability, 316, 318, 322
Junction defect rates, 389, 390

K

Kullback-Leibler distance (KLD), 335, 336

L

Lab-on-a-chip, 287
Lactate assay, 306
LFSR, 383
Linear feedback shift register, 98
Local view, 131, 147, 148
Locality constraint, 366–368
Localized communication, 339, 362, 363, 365, 367–369
Localized interconnection, 340, 342, 367
Locally confirmed instruction, 355–357, 360
Locked cell, 256
Logic circuits, 6, 30
Logic compatibility function, 320–323
Logical formula rewrites, 6
Low power, 315

M

Majority voting, 228
Manufacturing defect, 340
Manufacturing yield, 122, 124, 149
Mapping units (MU), 379, 381, 383, 388, 392–394
Markov neighborhood, 319
Markov random field, 315–318, 320
Markov random network, 319, 320
Markovian, 375, 382
Master equation, 254, 258
Matching, 125, 126, 132, 134, 148, 149
Memory, 256
MEMS, 270, 287, 289,
Merge, 206, 207, 216, 218, 219
Message passing, 363, 365–367
Metal-dot QCA, 253, 254, 258, 262
Microfluidic array, 268–272, 277, 278, 280–282, 288–290, 293–296, 298–309
Microfluidic systems, 267, 268, 270, 271
Microfluidics, 287–289, 307
Microsystems, 267
Mixed-energy domain, 289
Mixed-technology, 287
Molecular circuits, 6, 29
Molecular CMOS (CMOL), 124
Molecular electronics, 1, 5, 7–9, 28, 33–60
Molecular latch, 378, 388
Molecular logic array (MLA), 36, 37, 99
Molecular nano-fabrics, 373–377, 380, 382, 384, 385, 387, 388, 395
Molecular switch, 35–38, 40, 46, 373, 378
Molecule cascades, 208–209
Mono-layered interconnected circuits, 154
Monte-Carlo, 278–280, 282, 283
Multiple fault, 159, 186, 187, 189
Multiple faulty module, 160, 183, 185, 190, 193, 199
Multiplexed bioassay, 306, 307

N

Nano-architecture, 340, 342–350, 360, 362–369
NanoBlock controllability, 113
NanoBlock tester, 105, 111–114
NanoBlock, 33–45, 47–50, 53–57, 60, 99–101, 103–105, 107–114, 117, 123
Nanoelectromechanical switch, 124
Nanoelectronic device, 339–342, 345, 363, 364
Nanoelectronic system, 340–344, 363
Nanoelectronics, 340, 341, 343, 344, 362, 364, 366, 368
NanoFabric, 33–43, 45, 49, 54, 55, 57, 60, 95–109, 114–118, 123
Nanoprocessor architecture, 362–368
Nanoscale computational model, 342–344, 347–363, 365, 366, 368, 369
Nanoscale devices, 315, 324, 336, 337
Nanoscale wire, 35, 36
Nanotechnology, 1, 2
Nanowire bridging fault, 135
Nanowire open fault, 135
Nanowire, 99
N-Modular Redundancy (NMR), 343, 346, 349, 352, 361, 362

Noise immunity, 316, 324–326, 329, 335, 336
Noise, 315, 316, 324–326, 329, 330, 332, 334–337
Noise-aware logic design, 337
None-some-many algorithm, 116
Non-volatile, 123, 148
Normalized Relative Performance (NRP), 392–394
NP-hard, 272–275
Null cell, 251, 255–257
Null configuration, 216, 217

O

Observability, 158–160, 178, 186, 187, 189–191, 193–196, 199
Off-line test, 298–303, 306–308
One-to-one onto mapping, 157–159, 163, 182, 183, 185, 186
On-line test, 288, 303, 304, 307
Optimal partitioning, 277, 278
Output response analyzer (ORA), 34, 35, 41–47, 49, 54, 55, 60, 97

P

Perfect matching, 126
Performance overhead, 347
Performance thresholds, 24, 25, 27
Permutation crossbar, 148, 149
Personality matrix, 104, 107
Phase diagram, 255, 260
Physical view, 128, 129, 141
Power gain, 254, 259, 260
Primary observable line, 187–189
Primitive, 203, 205, 207, 208, 216, 223, 224
Primitive-level fault, 216–219
Probabilistic computing, 315, 322
Probabilistic defect-maps, 375, 376, 380, 382
Processing elements (PE), 377, 379, 382–384, 387–389, 391–395
Processor architecture, 340, 342–345, 347, 349, 350, 360, 362–365, 368
Programmable Logic Arrays (PLAs), 103, 124
Programmable logic block (PLB), 34, 96
Programmable switch, 123–125, 148
PSA, 383
Pseudo-random maps, 377, 386, 388, 392, 394

Q

QCA (Quantum-dot Cellular Automata), 157–176, 178, 179, 186, 187, 199, 227–235, 237, 239, 243, 246, 248, 251–262
QCA cell, 251–253, 256, 258
QCA devices, 251, 253
QCA shift register, 253, 255, 256, 258, 261
QCA1 gate, 166, 193, 194
QCA2 gate, 195–197
Quantum dot, 251
Quantum modeling, 232
Quantum tunneling, 253–255

R

RAM, 96
Reagent, 306–308
Realistic defect, 291, 292
Reconfigurability, 34, 38, 41, 97, 101, 118
Reconfiguration, 289, 298, 304, 306
Recovery, 34, 35, 38–41, 44, 54–57, 59, 60, 102, 107–109, 114–117
Recursive Biclique Algorithm, 136–138
Reduced noise margins, 315, 324, 325
Redundancy factor, 375, 376, 379, 383, 388–391
Redundancy insertion, 375–377, 383, 385, 386, 388, 389, 392, 395
Redundancy, 122, 141, 144
Reed-Muller cell, 203, 210–223, 225
Reed-Muller form, 203, 209, 210, 213, 215, 225
Region based mapping, 388
Regions, 379–381, 383, 388, 392, 393
Relative cost (RC), 393, 394
Relative Performance (RP) Self-assembled, 392
Reliability, 267, 340, 341, 344, 345, 349, 360, 362, 364–366, 369
Resettable Modulo 2 Counter, 207

Resistor logic circuit, 8, 36
Resonant tunneling diode (RTD), 35
Return-to-spacer protocol, 205, 207
Reversibility, 154
Reversible computing, 157–160, 199
Reversible fault, 187–189
Reversible logic, 157–160, 163, 167, 170, 174, 183
Robust Coplanar Crossing, 227–249
Robustness, 253, 258
Rotaxane, 99
Routing, 376, 377, 386, 387, 392–394
Row-column algorithm, 108, 109, 116

S

Self-alignment, 33, 95, 101
Self-assembly, 33, 35, 121, 123, 145, 149
Shift register, 251–253, 255–258
Signal noise, 324, 336
Signal polarization, 154
Signal restoration, 254
Signal, block, 204, 217
Signature generator, 98, 107–109
Silicon nanowires, 121
Single electron transistor, 252
Single electronics, 253–254
Single fault, 160, 163, 185, 199
Single faulty module, 160, 185, 199
Single missing/additional cell defect, 176, 177, 179, 243, 244
Single Pin Inversion (SPI), 187
Single stuck-at-1 fault in a TRIA, 219–221
SMART, 377, 379–380, 387, 388
Soft repair, 96
Speculative branch, 347, 351–358, 360, 369
Speculative computation, 351–353, 365
Stochastic Model Checking, 379
Structurally redundant, 375, 377, 385, 386
Stuck-at fault, 49, 50, 60, 160, 186
Stuck-at-0 fault, 217–219
Stuck-at-1 fault in AND gate, 221, 222, 224
Sub-threshold voltage, 324, 330, 335
Switch stuck-closed fault, 134–135, 145
Switch stuck-open fault, 134, 135
SwitchBlock, 35–41, 44, 47–49, 52–57, 60, 99, 103, 105–107, 110
System-on-chip, 263

T

Teramac, 34, 98
Test group (TG), 41, 43–49, 52–55, 57, 58, 60
Test pattern generator (TPG), 34, 35, 41–44, 48, 49, 52, 54, 57, 60, 97
Test planning, 268, 270, 278–280, 284
Test resource optimization, 268
Testability, 158–160, 163, 178, 179, 183, 186, 187, 189, 191, 194, 196, 198, 199
Thermal effects, 227, 228, 234
Thermal noise, 154, 324
Threshold voltage variation, 325
Time complexity, 125, 132, 136, 138, 140
Time redundancy, 343–347, 349, 352, 360, 361, 368
TMR, 386, 393–395
Toffoli gate, 160, 165–168, 176, 179, 183–185, 190
Topological structure, 362–368
Transient faults, 346, 373–375, 379, 380, 387, 395
Transition rules, 205–207, 224
Transition, 205–207, 224,
TRIA, 207, 208, 219–223
Trinder's reaction, 306
Triple stuck-at-1 fault in a TRIA, 221, 223
Triple-Modular Redundancy (TMR), 334, 346, 349
Tunnel junction, 252, 253
Tunneling, 252–255
Two dimensional (2D) crossbar, 123, 125, 149
Two-dimensional cellular array, 206

U

Ultimate CMOS, 325
Ultra-low voltage, 337
Unclustered, 132–134, 141–146
Universal defect-free subset, 122, 128, 129, 136
User crossbar, 148, 149

V

Von Neumann neighborhood, 205
Voter, 344, 347–351, 358–361, 363–368

W

Walking sequence of ones, 48, 51
Walking test patterns, 110
Wave-like test, 112
Wire mesh, 99, 101

Y

Yield estimation, 123, 134–136
Yield metric, 123, 132–136, 142

Frontiers in Electronic Testing

(Continued from page ii)

Power-Constrained Testing of VLSI Circuits
Nicola Nicolici and Bashir M. Al-Hashimi
Volume 22B, ISBN 978-1-4020-7235-2, 2003

High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test
R. Dean Adams
Volume 22A, ISBN 978-1-4020-7255-0, 2002

SOC (System-on-a-Chip) Testing for Plug and Play Test Automation
Krishnendu Chakrabarty (Ed.)
Volume 21, ISBN 978-1-4020-7205-5, 2002

Test Resource Partitioning for System-on-a-Chip
Krishnendu Chakrabarty, Vikram Iyengar and Anshuman Chandra
Volume 20, ISBN 978-1-4020-7119-5, 2002

A Designer's Guide to Built-In Self-Test
Charles E. Stroud
Volume 19, ISBN 978-1-4020-7050-1, 2002

Boundary-Scan Interconnect Diagnosis
José T. de Sousa and Peter Y.K. Cheung
Volume 18, ISBN 978-0-7923-7314-8, 2001

Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits, Second Edition
M. Bushnell and Vishwani Agrawal
Volume 17, ISBN 978-0-7923-7991-1, 2000

Analog and Mixed-Signal Boundary-Scan: A Guide to the IEEE 1149.4 Test Standard
Adam Osseiran (Ed.)
Volume 16, ISBN 978-0-7923-8686-5, 1999

Design for AT-Speed Test, Diagnosis, and Measurement
Benoit Nadeau-Dostie (Ed.)
Volume 15, ISBN 978-0-7923-8669-8, 1999

Delay Fault Testing for VLSI Circuits
Angela Krstic and Kwang-Ting (Tim) Cheng
Volume 14, ISBN 978-0-7923-8295-9, 1998

Research Perspectives and Case Studies in Systems Test and Diagnosis
John W. Sheppard and William R. Simpson (Eds.)
Volume 13, ISBN 978-0-7923-8263-8, 1998

Formal Equivalence Checking and Design Debugging
Shi-Yu Huang and Kwang-Ting (Tim) Cheng
Volume 12, ISBN 978-0-7923-8184-6, 1998

On Line-Testing for VLSI
Michael Nicolaidis, Yervant Zorian, and Dhiraj Pradhan (Eds.)
Volume 11, ISBN 978-0-7923-8132-7, 1998

Reasoning in Boolean Networks: Logic Synthesis and Verification Using Testing Techniques
Wolfgang Kunz and Dominik Stoffel
Volume 9, ISBN 978-0-7923-9921-6, 1997

Introduction to IDDQ Testing
S. Chakravarty and Paul J. Thadikaran
Volume 8, ISBN 978-0-7923-9945-2, 1997

Multi-Chip Module Test Strategies
Yervant Zorian
Volume 7, ISBN 978-0-7923-9920-9, 1997

Testing and Testable Design of High-Density Random-Access Memories
Pinaki Mazumder and Kanad Chakraborty
Volume 6, ISBN 978-0-7923-9782-3, 1996

From Contamination to Defects, Faults and Yield Loss: Simulation and Applications
Jitendra B. Khare and Wojciech Maly
Volume 5, ISBN 978-0-7923-9714-4, 1996

Efficient Branch and Bound Search with Application to Computer-Aided Design
Xinghao Chen and Michael L. Bushnell
Volume 4, ISBN 978-0-7923-9673-4, 1996

Testability Concepts for Digital ICs: The Macro Test Approach
F.P.M. Beenker, R.G. Bennets, and A.P. Thijssen
Volume 3, ISBN 978-0-7923-9658-1, 1995