Emergence, Complexity and Computation ECC

Ioannis Vourkas
Georgios Ch. Sirakoulis

Memristor-Based Nanoelectronic Computing Circuits and Architectures

Foreword by Leon Chua

Springer

Emergence, Complexity and Computation

Volume 19

About this Series

The Emergence, Complexity and Computation (ECC) series publishes new developments, advancements and selected topics in the fields of complexity, computation and emergence. The series focuses on all aspects of reality-based computation approaches from an interdisciplinary point of view especially from applied sciences, biology, physics, or chemistry. It presents new ideas and interdisciplinary insight on the mutual intersection of subareas of computation, complexity and emergence and its impact and limits to any computing based on physical limits (thermodynamic and quantum limits, Bremermann's limit, Seth Lloyd limits…) as well as algorithmic limits (Gödel's proof and its impact on calculation, algorithmic complexity, the Chaitin's Omega number and Kolmogorov complexity, non-traditional calculations like Turing machine process and its consequences,…) and limitations arising in artificial intelligence field. The topics are (but not limited to) membrane computing, DNA computing, immune computing, quantum computing, swarm computing, analogic computing, chaos computing and computing on the edge of chaos, computational aspects of dynamics of complex systems (systems with self-organization, multiagent systems, cellular automata, artificial life,…), emergence of complex systems and its computational aspects, and agent based computation. The main aim of this series it to discuss the above mentioned topics from an interdisciplinary point of view and present new ideas coming from mutual intersection of classical as well as modern methods of computation. Within the scope of the series are monographs, lecture notes, selected contributions from specialized conferences and workshops, special contribution from international experts.

More information about this series at http://www.springer.com/series/10624

Ioannis Vourkas · Georgios Ch. Sirakoulis

Memristor-Based Nanoelectronic Computing Circuits and Architectures

Springer

Ioannis Vourkas
Department of Electrical and Computer Engineering
Democritus University of Thrace
Xanthi
Greece

Georgios Ch. Sirakoulis
Department of Electrical and Computer Engineering
Democritus University of Thrace
Xanthi
Greece

ISSN 2194-7287 ISSN 2194-7295 (electronic)
Emergence, Complexity and Computation
ISBN 978-3-319-22646-0 ISBN 978-3-319-22647-7 (eBook)
DOI 10.1007/978-3-319-22647-7

Library of Congress Control Number: 2015946759

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

to my wife Evelyn with love
I.V.
to my family: Stella, Marina, and Christos for their love and support
G.S.

Foreword

The memory-resistor (memristor) is a two-terminal electronic device, defined by a state-dependent Ohm's law; its resistance depends on a set of internal state-variables. The favorable circuit properties of memristors justify the recent explosive growth of related research efforts which led to several advancements in theory and potential unique applications of memristors including, among others, computing.

Currently there are only a few available book titles devoted to memristors. Vourkas and Sirakoulis in *Memristor-Based Nanoelectronic Computing Circuits and Architectures* bring together a series of memristor-related topics which are studied and presented for the first time in a single volume, i.e. device modeling, complex device interconnections, logic and memory circuits, as well as computing circuits and systems where the memristors are used either as two-state switches or as analog devices. More specifically, the book consists of eight main chapters. Chapter 1 deals with the foundations of memristor theory and the fundamental properties of memristors. Chapter 2 is devoted to modeling of voltage-controlled bipolar memristors and describes a threshold-type SPICE-compatible device model, on which the authors based the simulations and research findings shown in the rest of the book. Chapter 3 focuses on complex memristor interconnections and studies the composite emerging behavior with application in memristive multi-state switches. Chapter 4 addresses design strategies for digital logic circuits with memristors, passing from sequential stateful logic to new circuit design schemes which allow for parallel processing of the applied inputs. Chapter 5 is dedicated to crossbar-based information storage systems, studying alternative memory cells and architectural aspects which could lead to more reliable memristor memories. In the same context, Chapter 6 integrates the memristive multi-state switches of Chap. 3 with the crossbar circuit geometry in a multi-level memristor-based crossbar memory, which is then used in an early approach to memristor-based high-radix arithmetic logic units (ALU). Chapter 7 studies the emerging parallel computing capabilities of complex two-dimensional memristor networks and presents a novel methodology to efficiently map oriented graphs onto memristive networks, using circuit models which cover a variety of connection types between graph vertices. Finally, Chapter 8 presents a circuit-level Cellular Automata (CA)-inspired methodology for computational schemes which are applied to solve several NP-hard problems of various areas of artificial intelligence (AI).

All the parts of the book are written in a simple language accessible by scientists, researchers, engineers, as well as young undergraduates. This book title is unique and timely, providing a comprehensive study which spans from memristor fundamental theory, device modeling and device interconnections, to circuit-level and system-level digital/analog applications. It includes several new results originating from the research endeavor of the authors in this very promising and highly multidisciplinary scientific field. At the moment, there is not any competitive title which deals with the range of the provided here memristor-related research in a truly compact form, which is why *Memristor-Based Nanoelectronic Computing Circuits and Architectures* can be a valuable textbook for undergraduate and postgraduate students.

Berkeley, USA Leon Chua

Preface

Motivation

Continued dimensional and functional scaling of Complementary Metal-Oxide-Semiconductor (CMOS) technology is driving information processing into a broadening spectrum of new applications. Many of these applications are enabled by performance gains and/or increased complexity realized by scaling. The performance of the components and the final application can be measured in many different ways; higher speed, higher density, lower power, more functionality, etc. Traditionally, though, dimensional scaling had been adequate to bring about these performance merits but it is no longer so. Since dimensional scaling of CMOS will eventually approach fundamental limits, several new alternative information processing devices and architectures for existing or new functions are being explored to sustain the historical integrated circuit scaling cadence and the reduction of cost/function in the next decades [1].

CMOS logic and memory together form the predominant majority of semiconductor device production. Today the semiconductor industry is facing two classes of difficult challenges related to extending integrated circuit technology to new applications and to beyond the end of CMOS dimensional scaling. One class relates to pushing CMOS beyond its ultimate density and functionality by integrating a new high-speed, highly-dense, and low-power memory technology onto the CMOS platform. The other class is to extend information processing substantially beyond that attainable by CMOS, using an innovative combination of new devices, interconnect and architectural approaches for extending CMOS and eventually inventing a new information processing platform technology. Difficult challenges gating the development of emerging research devices are therefore divided into two parts: (i) those related to memory technologies, and (ii) those related to information processing or logic devices.

The semiconductor industry is definitely in need of a new memory technology that combines the best features of current memories in a fabrication technology compatible with the CMOS process flow, scaled beyond the present limits of SRAM and Flash. This would provide a memory device fabrication technology required for both stand-alone and embedded memory applications. For DRAM, currently the main goal is to continue to scale the foot-print of the 1T-1C cell to the practical limit of $4F^2$, where F is the minimum feature size. Some issues concern vertical transistors, new dielectrics which improve the capacitance density, and keeping the leakage currents low. The requirement of low leakage currents, however, causes problems in obtaining the desired access transistor performance. A revolutionary solution of having a capacitor-less cell would be highly beneficial. Regarding nonvolatile memory (NVM), the current mainstream is Flash memory. Dense, fast, and low-power NVM is becoming highly desirable in computer architecture. However, there are serious issues with scaling of Flash memories. 2D Nand-type Flash should stay dominant for as far as it can scale because it is a well-established technology and has a very simple structure, requiring only one transistor. Ultimate density scaling may require 3-D architecture, such as vertically stackable cell arrays with acceptable yield and performance. 3-D Nand Flash is currently being developed but cost-effective implementation of this new technology, along with multi-level cell and acceptable reliability, remains a difficult challenge. Consequently, since the ultimate scaling limitation for charge-based storage devices is too few electrons, devices that provide memory states without electric charges are promising to scale further.

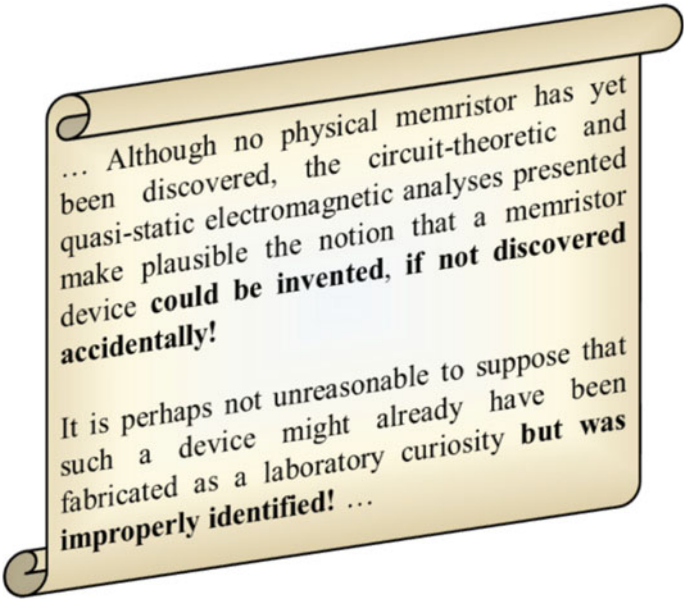
Moreover, as mentioned before, a major portion of semiconductor device production is devoted to CMOS digital logic, both high-performance and low-power, which is typically for mobile applications. A longer-term challenge is therefore the invention of a producible information processing technology addressing "beyond CMOS" applications. For example, emerging research devices might be used to realize special purpose processing units that could be integrated with multiple CMOS components to obtain performance advantages. These new special purpose units may provide a particular system function much more efficiently than a digital CMOS block, or they may offer a uniquely new function not available in a CMOS-based approach. A new information processing technology must also be compatible with a system architecture that can fully utilize the new device. Possibly, a non-binary data representation and/or non-Boolean logic may be required to employ a new primitive device for information processing.

All aforementioned requirements are currently driving the industry towards a number of major technological innovations, including material and process changes, as well as totally new circuit structures. There is a growing interest in new devices for information processing and memory, new technologies and new paradigms for system architecture. Solutions to all these challenges could also lead to new opportunities for an emerging research device technology to eventually replace CMOS as a mainstream information processing technology, provided that it possesses most of (if not all) the mentioned desirable performance merits. To this end, resistive-switching devices known as "memristors" or "memristive devices" have become the focus of many research efforts by academia and industry lately. Their advantageous performance characteristics render them a candidate technology able to bring the next technological revolution in electronics, while serving as a bridge between CMOS and the realm of nanoelectronics beyond the end of CMOS dimensional and equivalent functional scaling.

Memristor: A Promising Emerging Nanoelectronic Device

As a result of his preliminary exceptional work in nonlinear circuit theory during the 1960s, in 1971 Prof. L.O. Chua made an interesting observation that led to his discovery of the memristor as a mathematical entity [2]. For completely linear circuits there are only three independent two-terminal passive circuit elements: the resistor R, the capacitor C, and the inductor L, which are defined axiomatically via a constitutive relation between a pair of variables chosen from $\{v$ (voltage), i (current), q (charge), φ (flux linkage)$\}$. There are six different pairs than can be formed from these four variables, namely $\{(v, \varphi), (i, q), (v, i), (v, q), (i, \varphi), (\varphi, q)\}$, and five of them were already related mathematically. However, when Chua generalized the mathematical equations to be nonlinear, there was another independent differential relationship that in principle coupled the charge q that flowed through the circuit and the *flux linkage* (time-integral of the applied voltage) φ as in $d\varphi = Mdq$, different from the resistance which coupled the voltage v to the current i, $dv = Rdi$.

He mathematically explored the properties of this new nonlinear circuit element and found that it was essentially a "resistor with memory", so he called it a memristor M; it was a two-terminal device that changed its resistance according to the amount of change that flowed through it. This prediction of the properties of a new "missing" (by that time) circuit element from symmetry principles was absolutely revolutionary; more importantly, it did not depend on any experimental observation but it was rather a result of curiosity. As Chua himself declared in his 1971 paper, it was not obvious at that time that a physical analog of such circuit element existed; the attached text below is a summary of what is stressed in the original paper (last paragraph on page 519 of [2]).



… Although no physical memristor has yet been discovered, the circuit-theoretic and quasi-static electromagnetic analyses presented make plausible the notion that a memristor device **could be invented**, **if not discovered accidentally!**

It is perhaps not unreasonable to suppose that such a device might already have been fabricated as a laboratory curiosity **but was improperly identified!** …

The reason why memristors are substantially different from the other fundamental circuit elements is that, when you turn off the voltage to the circuit they still remember how much voltage was applied before and for how long, thus presenting a memory of their past. That's an effect that can't be duplicated by any circuit combination of resistors, capacitors, and inductors, which qualifies the memristor as a fundamental circuit element.

Today we know that memristors are ubiquitous and many devices, including the "electric arc" which dates back to 1801, have been identified as memristors. Indeed, there had been experimental clues to the memristor's existence all along the last two centuries. Scientists have been publishing in the literature experimental results with "strange" voltage characteristics, where one sees clearly memristance, though such a material property had always been shadowed by other effects that were of primary interest [3]. In the absence of an application, there was no particular need to seek memristive behavior anyway. After the publication of Chua's seminal paper, the connection between many strangely behaving components and his original theoretical definition was not made at least for three decades by then. The memristor had been relegated as an abstract device with no practical significance until 2008 when Chua's theory of memristor was successfully linked to its first "modern" practical nanoscale implementation by a group at Hewlett Packard (HP) Laboratories [4]. Their seminal Nature paper originated intense research activity in this novel scientific field and generated unprecedented worldwide interest for the potential applications, with publications increasing at an exponential rate ever since.

Memristor exhibits its unique properties primarily at the nanoscale. Therefore, much of the recent research work has focused on the technological side concerning the physical realization of such devices for a better understanding of the physical principles and their tuning. Currently, there is a growing variety of systems that exhibit memristive behavior, as academia and industry keep on with their research and prototyping [5, 6]. Among them, molecular and ionic thin film memristive systems primarily rely on different material properties of thin film atomic lattices that exhibit hysteresis under the application of charge. In experimentally realizable systems, memristive devices with threshold voltages seem to be the norm rather than the exception, and electronic conduction is in most cases dominated by an effective tunneling barrier—width that varies with the applied voltage.

The memristor creates a new opportunity for realization of innovative circuits that in some cases are not possible or have inefficient realization in the present and established design domain. It provides many advantages such as scalability down to sub-10 nm, nonvolatility, fast switching speed, energy efficiency, and CMOS compatibility, just to name a few; thus it is believed to bring a new wave of innovation in electronics, supplanting or supplementing transistors in several applications, while it might bring analog information processing back into the world of computing. Memristor-based circuits open new pathways for the exploration of advanced computing architectures as promising alternatives to conventional integrated circuit technologies, which are facing serious challenges related to continuous scaling [1]. Most importantly, memristors provide an unconventional computation framework, different from familiar paradigms, which combines information processing and storage in the memory itself; i.e. the major distinction from the present day's computing technology [7]. Such framework is determined more by the device properties than any previously conceived logic paradigm.

Amongst several emergent applications of the memristance switching phenomenon, implementation of logic circuits is gaining considerable attention. In binary digital circuits, memristors would operate as two-state switches, toggling between max and min resistance. Using memristors for digital processing has the advantage of combining storage and logic functionality with the same technology in one single device. However, the widest field of proposals on how to use memristors for processing concerns analog computing. If several intermediate resistive states could be distinguished reliably, then the information density could be raised to more than one bit per device, but the end point of this evolution is to be able to fully exploit the analog nature of memristors. For example, using the possibility to store a ternary value in one physical storage cell allows building up a better arithmetic unit as is fundamentally possible and actually done with conventional binary logic. Anyway, active components such as transistors would still be needed even if most information processing were done by memristors. One reason is that signals are reduced in amplitude by every passive circuit element and, at some point, they must be restored. Another reason might concern accessing memristors for reading/programming their state. Hybrid circuits that combine memristors and active elements are a lively area of investigation, whereas the distinct properties of memristive devices might even lead to *neuromorphic* computer systems in the future [8].

Up to now, the fabrication of digital memories is the driving force of memristor technology, since very dense memory architectures can potentially be manufactured. Rapid progress in the advancements of memristive technology is reflected in the early commercialization of memristive memory (resistive RAM—ReRAM) products [9]. Such activity together with the groundbreaking announcement of “the Machine” by HP on June 2014 [10], prove the ever-increasing interest and active involvement of industry leading companies in the future production of memristor-related products and pioneering memristive computing architectures. The continuous improvement of the memristance switching behavior, thanks to the incessant accumulation of knowledge on resistive switching materials and the underlying phenomena, is encouraging for the future implementation and establishment of unconventional computing paradigms and sophisticated memristive circuits and systems. But whether the memristor will finally fulfill all these hopes remains to be seen; in order to evaluate long-term prospects of such technologies one would have to go beyond the basic principles and to questions of reliability, variability, manufacturing cost, etc.

The content of this book spans from fundamental device modeling to emerging storage system architectures and novel circuit design methodologies, targeting advanced non-conventional analog/digital massively parallel computational structures. Effective modeling is the first step towards a deeper understanding of the memristive dynamics and the better exploitation of their unique properties for potential utilization in a variety of emerging applications. Well defined and effective SPICE-compatible memristor models, as those presented in Chap. 2, would certainly accelerate research in memristive circuits and systems. Also, while most of the research has so far focused on the properties of single memristors, very little is known about their response when they are organized into networks. Composite memristive systems built out of networks of individual memristors, demonstrate different electrical characteristics from their structural elements due to their threshold-dependent nonlinear resistance switching behavior. Therefore, their rich and dynamic overall behavior could be exploited for the creation of novel sophisticated memristive circuits and systems with multi-bit storage per device capabilities. Collective memristive dynamics is the focus of Chaps. 3, 7, and 8, whereas the same property is the basis for the design of memristor-based logic circuits in Chap. 4. Furthermore, nonvolatile resistive RAM (ReRAM) is nowadays considered as one of the promising alternatives to current baseline memory technologies. At the architectural level, crossbar memory cell array structure offers several benefits and is considered one of the best ways to implement ReRAM of highest possible device density. However, a typical passive crossbar memory suffers from the existence of parasitic conducting (current sneak paths) reducing both the size and the reliability of the memory. Innovative approaches to memory cell structure and memory architecture, which will efficiently address the current sneak-path problem, constitute nowadays a key factor towards the practical realization of passive crossbar-based ReRAM and reflect the content of Chap. 5.

Moreover, a great effort was placed towards the creation of relevant design automation/simulation tools and proper methodologies which address important current technological drawbacks and thus enable/facilitate the development of efficient design flows for reliable circuits and architectures comprising memristors. Such tools are presented in Chaps. 4 and 5. Furthermore, it has been well-known for a long time that faster arithmetic operations could be achieved via high-radix numeric systems [11]. However, in the absence of appropriate storage devices, such practice was not given much attention because it would require doubling the memory capacity to represent high-radix numbers in binary mode. In Chap. 6 we present a novel method for implementing crossbar-based multi-level memories, where each cross-point cell stores multiple bits. Furthermore, we propose a conceptual solution for novel CMOS-compatible, memristive, high-radix arithmetic logic units (ALUs) for future computing systems.

The extensive study of memristive nanoelectronic circuits and architectures presented within this book is indicative of the fast pace of this novel and intriguing field. High-density memristive data storage combined with memristive circuit-design paradigms and computational tools applied to solve NP-hard artificial intelligence (AI) problems, as well as memristive arithmetic-logic units, certainly pave the way for a much promising memristive era in electronic computing systems. The graph-based NP-hard problems are depicted to memristive networks and coupled with Cellular Automata (CA)-inspired computational schemes that enable computation within memory. The following chapters may constitute an informative cornerstone for researchers and scientists, as well as a comprehensive reference to the more experienced readers, hoping to stimulate further research on memristive devices, circuits, and systems.

Book Outline

Below there is a short summary of the following chapters which highlights the original contributions of this book to the *state-of-the-art*.

Basic theoretical definitions and general properties of memristors and memristive systems are shortly presented in Chap. 1. All necessary information for the purposes and the complete comprehension of the content of this book is provided.

In Chap. 2, the device characteristics of thin-film memristors are considered and a novel, SPICE-compatible, generic, threshold-type switching model of a two-terminal voltage-controlled bipolar memristor is presented, explaining the memristive behavior of the device by investigating the occurrence of quantum tunneling.

A rigorous study of the switching response of composite memristive systems, consisting of multiple memristors connected in complex circuit configurations, is presented in Chap. 3. A methodology for the construction of composite memristive devices, which comply with certain design specifications and facilitate the design of nanoelectronic circuits with multi-state switches, is presented. Particular application examples of the methodology in novel analog computational structures conclude this chapter.

Chapter 4 addresses memristive logic design and computational methodologies. It includes a comprehensive summary of the most recognized memristive Boolean logic design concepts, which are based on collective memristive dynamics, and presents two novel circuit design methodologies based on memristors. Particularly, a new CMOS-*like* memristor-based circuit design approach and methodology that enables the creation of complementary logic, mapped onto a hybrid memristor/CMOS crossbar-based platform, is described. The proposed methodology is applied to the design and simulation of large combinational logic circuits, i.e. encoders and decoders. A proper, high-level design and simulation software tool for CMOS-*like* memristive logic circuits, which incorporates the developed memristor device model of Chap. 2, is also presented. The focus is then on the evolution of the memristor-based logic circuit design strategies from the proposed sequential stateful logic up to novel design schemes which support parallel processing of input signals.

In Chap. 5 alternative crossbar architectures are introduced in an attempt to minimize the impact of the current sneak paths, while enabling larger array size and better read voltage margins towards more reliable memristor-based crossbar memories, compared to other approaches found in the literature. Moreover, novel memristive memory cell structures, comprising parallel/serial memristors, are investigated to possibly address the parasitic conducting problem. XbarSim, a high-level, educational GUI-based simulation environment which incorporates the proposed device model for memristors and enables the study and experimentation with standard/alternative memristive crossbar architectures, targeting memory or logic applications, is also presented.

Chapter 6 presents an early approach to the design of a reconfigurable, memristor-based, arithmetic-logic unit (ALU) for future computing systems. The proposed ALU system combines CMOS peripheral circuitry with a high-density memristive multi-level crossbar, which allows the compact high-radix storage of numbers. The high-radix stored information is selectively converted to binary representation with the use of a network of comparators before it is supplied to a computational layer of fast adders. The memory module of the system allows for parallel read/write operations and achieves inherently the parallel creation of partial products, to be used for faster multiplication.

Chapter 7 explores memristive networks (grids) where emergent computation arises through collective device interactions. Computing efficiency of the grids is studied in several scenarios and new composite memristive structures are utilized in shortest path and maze-solving computations, addressing known problems of relevant published works in the recent literature. Some already published approaches are substantially extended by introducing modifications in the computing platform, thus leading to better results. Additionally, a methodology for the appropriate mapping of oriented graphs onto memristive networks, based on circuit models which correspond to several types of connections between graph vertices, is presented for the first time. This methodology simplifies the precise network projection of any mesh-based oriented graph via a one-to-one correspondence.

Chapter 8 concludes this Book providing a novel circuit-level Cellular Automata (CA)-inspired methodology for computational schemes capable of executing computations within memory. The novel computing structures are based on the threshold-based resistance switching behavior of multi-state composite memristive components located in array-like circuit structures. The unique composite circuit properties of memristors are exploited within CA-inspired circuit implementations, which are applied to solve several NP-hard problems of various areas of artificial intelligence.

Xanthi, Greece
March 2015

Ioannis Vourkas
Georgios Ch. Sirakoulis

References

1. International Technology Roadmap for Semiconductors (ITRS) (2013). Available: http://www.itrs.net/. Accessed June 2014
2. L.O. Chua, Memristor—the missing circuit element. IEEE Trans. Circuit Theory **18**(5), 507–519 (1971)
3. T. Prodromakis, C. Toumazou and L.O. Chua, Two centuries of memristors. Nature Materials **11**(6), 477–557 (2012)
4. D.B. Strukov, G.S. Snider, D.R. Stewart, R.S. Williams, The missing memristor found. Nature **453**(May), 80–83 (2008)
5. Y.V. Pershin, M. Di Ventra, Memory effects in complex materials and nanoscale systems. Adv. Phys. **60**(2), 145–227 (2011)
6. L.O. Chua, If it's pinched it's a memristor. Semicond. Sci. Technol. **29**(10), 104001 (2014)
7. E. Linn, R. Rosezin, S. Tappertzhofen, U. Bottger, R. Waser, Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations. Nanotechnology **23**(305205) (2012)
8. Y.V. Pershin, M. Di Ventra, Neuromorphic, digital and quantum computation with memory circuit elements. Proc. IEEE **100**(6), 2071–2080 (2012)
9. Panasonic, The new microcontrollers with on-chip non-volatile memory ReRAM (2012). Available: http://panasonic.co.jp/corp/news/official.data/data.dir/jn120515-1/jn120515-1.html. Accessed 20 September 2014
10. HP Cloud Source Blog, The Machine, a view of the future of computing (2014). Available: http://h30507.www3.hp.com/t5/Cloud-Source-Blog/The-Machine-a-view-of-the-future-of-computing/ba-p/164568#.VB2Bw5qKDGg. Accessed 20 September 2014.
11. R.P. Brent, P. Zimmermann, *Modern Computer Arithmetic* (Cambridge University Press, Cambridge, 2010)

Contents

Abstract

In the last decades, exponential reduction of integrated circuits feature size and increase in operating frequency was achieved in Very Large Scale Integration (VLSI) fabrication industry using conventional Complementary Metal-Oxide-Semiconductor (CMOS) technology. However, dimensional scaling of CMOS is expected to soon reach fundamental physical limits and this has driven great research efforts in emerging nanoelectronic devices over the last decade. Several new alternative information processing devices and architectures for existing or new functions, are being explored in an attempt to sustain the historical integrated circuit performance increase. To this end, the semiconductor industry today is facing two main challenges related to extending integrated circuit technology to beyond the limits of CMOS scaling: (i) to propel CMOS beyond its ultimate functionality by integrating a new high-performance memory technology onto the CMOS platform; (ii) to extend information processing far beyond that achievable by CMOS, using new devices which will either complement CMOS components or will eventually replace them completely. Among many nanotechnologies currently under intense investigation, resistance-switching devices generally referred to as “memristors” show great potential and the focused R&D efforts in many industrial laboratories make this technology widely considered as a potential successor of CMOS-based storage and processing cells in future electronic systems.

Memristor (concatenation of “memory resistor”), is the 4th fundamental circuit element, predicted by Chua in 1971 (joining the resistor, the capacitor, and the inductor), which represents one of today’s latest technological achievements. Memristor (here used to refer both to an “ideal” memristor as well as to a generalized memristive system) is a passive two-terminal electronic device whose behavior is described by a nonlinear constitutive relation between the voltage drop at its terminals and the current flowing through the device. The reason why memristors are substantially different from the other fundamental circuit elements is that, when the applied voltage is turned off, they still remember how much voltage was applied before and for how long; thus presenting memory of their past. However, this innovative device attracted most of attention worldwide only after 2008 when the first practical implementation was announced by Hewlett-Packard

(HP) Laboratories, originating intense research activity ever since. The increasing interest and the active involvement of industry-leading companies in future production of memristor-related products and pioneering memristive architectures, as well as the continuous improvement of the memristance-switching behavior thanks to the incessant accumulation of knowledge about the underlying device materials, are encouraging for the future implementation and establishment of memristive circuits and systems.

This book considers the design and development of nanoelectronic circuits and architectures focusing particularly on memristors. The ultimate goal is to study, explore, and address the related challenges and propose solutions for the smooth transition from conventional circuit technologies to emerging nanotechnologies. To this end, several new results on memristor modeling, memristive interconnections, logic circuit design, memory circuit architectures, computer arithmetic systems, development of design and simulation software tools, and applications of memristors in computing, are presented. Memristor device modeling constitutes a necessary first step towards further investigation and experimentation. After a brief introduction to the fundamentals of memristors in Chap. 1, memristor modeling is the focus of Chap. 2 where a threshold-type model of a voltage-controlled bipolar memristor is presented. Threshold-type switching is closer to the actual behavior of most experimentally realizable devices and the developed model attributes the resistance-switching behavior to a tunneling-distance modulation. Throughout the rest of the book, which spans a wide range of memristor-related topics and gives a nice overview of the current research trends, all analyses and simulations are based on this model. Specifically, complex memristive interconnections are studied in Chap. 3 in an attempt to explain and harness the threshold-dependent sophisticated composite behavior of multiple interconnected devices. The exploitation of the threshold-type switching of memristors and memristive compositions enabled the design of digital logic circuits as presented in Chap. 4. A CMOS-*like* memristor-based logic family that enables the creation of complementary logic in the crossbar geometry, is introduced. Memristors, which here are used as two-state switches rather than analog devices, serve both for information encoding and computation. This chapter also presents a software tool which allows the design and simulation of memristive CMOS-*like* circuits via a user-friendly graphical user interface (GUI). The chapter closes with the presentation of a logic design strategy which enables parallel processing of input signals, delivering high-performance resistive logic circuits. Crossbar-based resistive random access memory (ReRAM), a powerful promising alternative to existing baseline memory technologies, is the focus of Chap. 5. Mathematical analyses and simulation results of innovative approaches to memory cell structure and memory architecture, which alleviate the impact of the unwanted sneak currents by improving the read-out sense voltage-margins, are presented. The chapter closes with the demonstration of XbarSim, an educational simulation environment which was developed for the study of crossbar-based memristive circuits and which was used in all relevant conducted simulations whose results appear in this chapter. In Chap. 6 we exploit the multi-bit storage capability and the small footprint of memristors to propose a

novel CMOS-compatible high-radix arithmetic-logic unit (ALU) for future computing systems. The proposed ALU combines CMOS peripheral circuitry with a high-density memristive crossbar which comprises multi-state composite memristive switches and allows the compact high-radix storage of numbers. Chapter 7 presents system-level applications of memristors and composite memristive structures. Memristors create a new opportunity for realization of innovative circuits that are not possible or have inefficient realization in the present circuit design domain. So, this chapter explores memristive networks where emergent computation arises through collective device interactions, something promising to revolutionize hardware computing architectures. Computing efficiency of the networks is studied in several scenarios and composite memristive components are utilized for the solution of known, inherently complex in terms of computation time, problems in a massively parallel way. Finally, Chapter 8 presents a novel circuit-level Cellular Automata (CA)-inspired methodology for computational schemes capable of executing computations within memory. CA constitute a well-studied, inherently parallel, computing paradigm able to capture globally emerging behavior from the collective interaction of simple and local components. The proposed computing structures exploit the threshold-based resistance switching behavior of memristors and of their multi-state composite components in array-like circuit structures where the sparse nature of computations resembles certain operational features of CA. This way, a powerful computational tool is combined with the unique circuit properties of memristors within CA-inspired implementations which are applied to efficiently solve NP-hard artificial intelligence (AI) problems.

Chapter 1
Memristor Fundamentals

The *memristor* is considered one of the most promising nano-devices among those currently being studied for possible use in electronic systems of the future. The best performance features which have been demonstrated in published experimental results regarding research device prototypes so far include fast switching speed, high endurance and data retention, low power consumption, high integration density, and (perhaps most importantly) CMOS compatibility. Undoubtedly, the combination of such advantageous characteristics in a single device justifies the phenomenal research interest that resistance-switching devices have generally attracted over the last few years and verify the existing rumors about their potential application in both storage and processing units of future electronic systems. Memristive nano-devices are the focus of this book and this chapter aims to introduce the reader to their fundamental properties on which the presented study is based.

1.1 Introduction

The concept of the “ideal” *memristor* (concatenation of “memory resistor”) was first introduced in 1971 [1] as a two-terminal circuit element that linked the remaining missing pair of the four basic circuit variables, namely, *flux* and *charge*, as shown in Fig. 1.1. It was thus formally defined as the fourth fundamental circuit component (joining the resistor, the capacitor, and the inductor). A few years later, Chua and Kang [2] introduced to the scientific community the generic properties of a broad generalization of the memristor to an interesting class of nonlinear dynamical devices, called memristive devices. This chapter presents a summary of the memristor from a circuit-theoretic perspective, independent of the material the device is made of, and focuses on the information necessary to capture the memristor fundamentals and move on with the more-technical content of the chapters that follow.

I. Vourkas and G.Ch. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Emergence, Complexity and Computation 19,
DOI 10.1007/978-3-319-22647-7_1

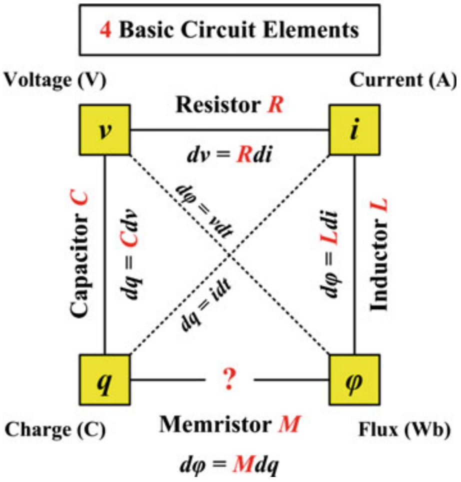


Fig. 1.1 The four fundamental circuit elements: resistor, capacitor, inductor, and memristor, defined using the four fundamental circuit variables: voltage *v*, current *i*, charge *q*, and flux linkage *φ*

1.2 Memristor Defined by a State-Dependent Ohm's Law

Normally there are two mathematical representations of time-invariant memristors depending on whether the input signal is a current source (current-controlled memristor) or a voltage source (voltage-controlled memristor). In a broader sense, any two-terminal electrical device is called a memristor if its behavior is described by a nonlinear constitutive relation between the voltage drop at its terminals *v* and the current flowing through the device *i*, as shown below:

Current-controlled memristor:

$$v = R(x)i \tag{1.1}$$

with the state equations:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, i) \tag{1.2}$$

Voltage-controlled memristor:

$$i = G(x)v \tag{1.3}$$

with the state equations:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = g(x, v) \tag{1.4}$$

The scalars $R(x)$ and $G(x)$ are called memristance and memductance (acronyms for memory resistance/conductance), respectively, and have units Ohm (Ω) and Siemens (S). The state-vector $x = (x_1, x_2, \ldots, x_n)$ has $n \geq 1$ components $x_1, x_2, \ldots x_n$

called state-variables, which represent internal physical parameters and do not depend on any external variables, such as voltages or currents.

1.3 Fingerprints of Memristors

Memristors have a unique set of "fingerprints", i.e. two important common properties which distinguish them among other resistance-switching electronic devices. The first is the "pinched" current–voltage (i–v) hysteresis loop which must hold for all amplitudes, for all frequencies, and for all initial conditions of any periodic waveform which assumes both positive and negative values over each period. In other words, there is no time delay (i.e. no phase-shift) between the voltage and the current waveforms since $v(t) = 0$ whenever $i(t) = 0$ for current-controlled memristors, or $i(t) = 0$ whenever $v(t) = 0$ for voltage-controlled memristors. The nonvolatile memory property of memristors is a direct consequence of the state-dependent Ohm's Law in Eqs. 1.1 and 1.3. It is important to note that, if one opens or short-circuits a memristor having a resistance R_0 at $t = t_0$ so that $v = 0$ and $i = 0$, the memristor does not lose its state information but it instead holds its state unchanged (ideally) forever!

This property is seen in Fig. 1.2 which shows qualitatively the response of a voltage-controlled, threshold-type switching bipolar memristor to a sinusoidal AC

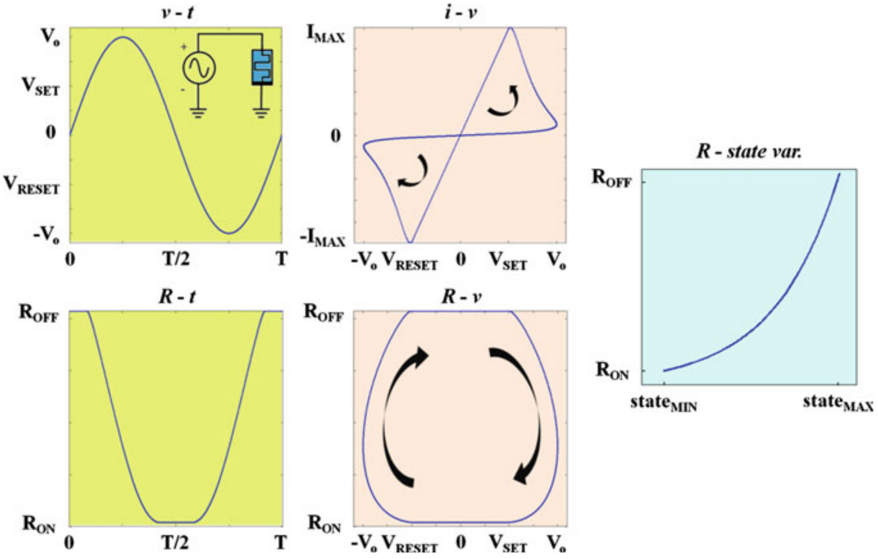


Fig. 1.2 Qualitative representation of the response of a voltage-controlled, threshold-type switching bipolar memristor to a sinusoidal AC applied voltage according to the model presented in Chap. 2. The simulation graphs include the applied voltage (v–t), the hysteretic current–voltage (i–v) characteristic, the change of the memristance with time (R–t) and with the applied voltage (R–v), respectively, as well as the memristance-state map

applied voltage. Threshold-type switching is closer to the actual behavior of most experimentally realizable memristive devices [3]; the resistance switching rate is small below (fast above) a voltage threshold (namely V_{SET} or V_{RESET}) which is viewed as the minimum voltage required to induce a change to the memristance of the device. The graphs shown in Fig. 1.2 include the applied voltage signal (*v*–*t*), the hysteretic current–voltage (*i*–*v*) characteristic, the corresponding change of the memristance with time (*R*–*t*) and with the applied voltage (*R*–*v*), respectively, as well as the memristance plotted as a function of the state-variable (memristance-state map).

The memristance-state map is a very useful graph because it shows how to navigate from one memristance R_0 at state s_0 to another memristance R_1 at state s_1 by applying a voltage pulse of properly selected amplitude and duration; it therefore allows one to tune the memristor's resistance continuously (in the chapters that follow we will refer to this analog operation of memristors which is still difficult to be achieved experimentally in a reliable manner). On the contrary, the rest of the plots shown in Fig. 1.2 cannot be used to predict the response given any other excitation waveform different from the depicted one. A "pinched" *i*–*v* is not unique but varies with the input waveforms, as well as the amplitude and the frequency. While a pinched *i*–*v* loop, measured from an experimental device, implies that a device is a memristor, it is completely useless by itself as a model as it cannot predict the response to an arbitrary input signal. The only way to do this is via the memristance-state map. The latter obeys the Ohm's Law, except that the memristance is not constant, as illustrated in Fig. 1.2, but it depends on a dynamical state-variable which evolves according to a prescribed state-equation as Eqs. 1.2 and 1.4.

The other unique property shared by all memristors is that, as the frequency of the applied periodic signal increases, the area enclosed within each part of the *i*-*v* sub-loop in the first and third quadrants deforms and shrinks continuously. The graph tends to collapse to a straight line (a single-valued function) which passes through the origin. In other words, high-frequency input signals do not give the memristor the time required for it to change its state. This property is confirmed in Fig. 1.3 which shows three different {*i*–*v*, *R*–*v*} pairs of a voltage-controlled memristor under sinusoidal excitations of the same amplitude but of different frequencies. The memristor is initially found in the high resistive state whereas the minimum achieved resistance differs each time, thus causing a different *i*–*v* plot. The above criteria of *pinched hysteresis loop* and the *single-valued function limiting phenomenon* as $\omega \rightarrow \infty$ must hold for all memristors.

1.4 Memristor Defined by a "Pinched" Hysteresis Loop

As Chua himself stressed in one of his most recent papers [4], for a device to be called a memristor, its hysteresis loop must be pinched and must pass through the origin in the *i*–*v* plane. A hysteresis loop is said to be pinched at the origin if it always passes through the origin at all time instants when the input signal waveform is zero. However, it is important to understand that pinched hysteresis loops are not

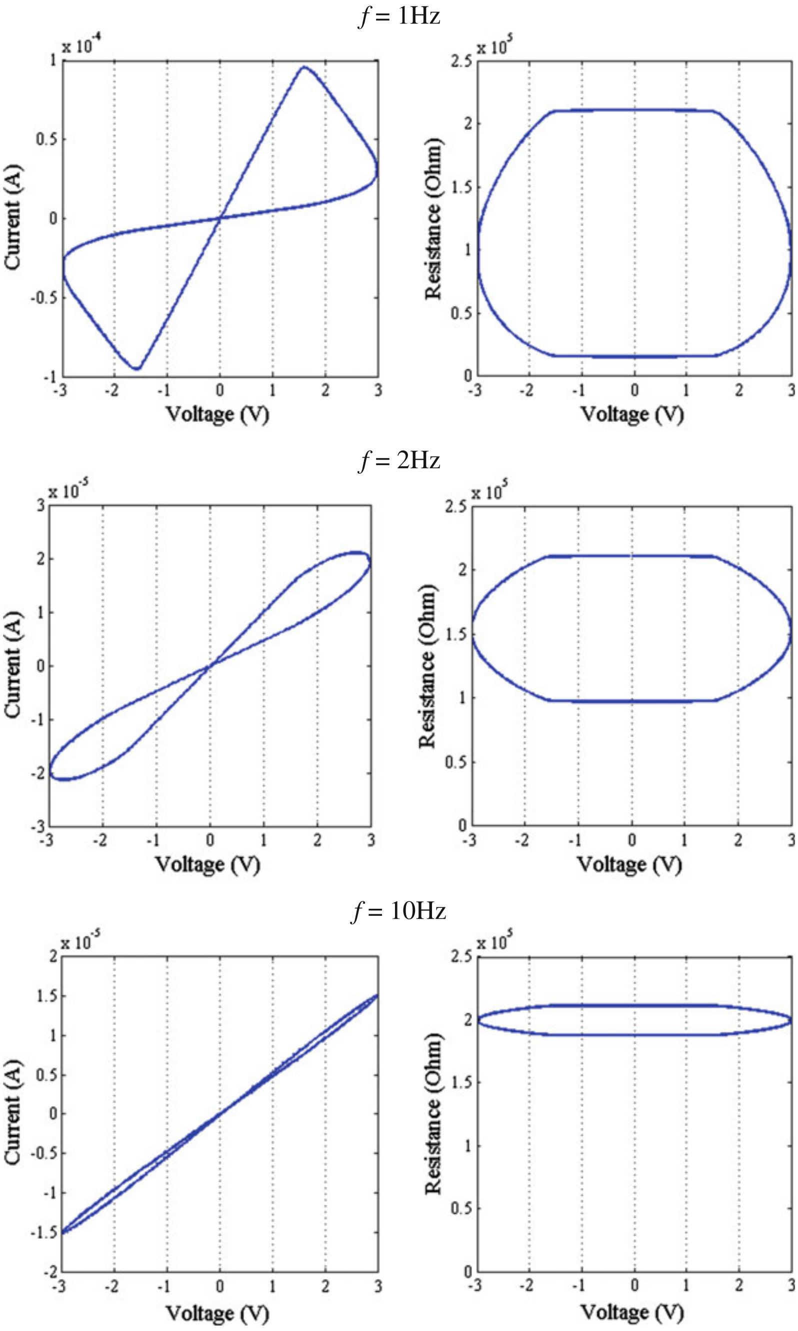


Fig. 1.3 Pinched hysteretic *i*–*v* loops along with the corresponding *R*–*v* plots for a memristor under a sinusoidal applied voltage of the same amplitude but of different frequency *f* according to the model presented in Chap. 2

models because "models must predict" but pinched hysteresis loops cannot predict what happens if another waveform is applied across the device. Any two distinct periodic input signals would give distinct pinched hysteresis loops associated with a particular memristor, thus they constitute an "identity card" of that particular device.

Indeed, from an experimental perspective a memristor is best defined as any two-terminal device that exhibits a pinched hysteresis loop in the *v*–*i* plane when driven by any periodic voltage or current signal. This definition greatly broadens the scope of memristive devices to encompass even non-semiconductor devices, both organic and inorganic [5]. It is also in line with the original definition of the ideal memristor in [1], thus pinched hysteresis loops are in fact the hallmarks of all memristors, ideal or generic. Nevertheless, pinched hysteresis loops of ideal memristors must be odd symmetric, thus any non-volatile resistive memory device that exhibits a pinched hysteresis loop that is not odd symmetric, such as those shown in the following chapters, must be modeled as a generic memristor.

1.5 The "Ideal" Memristor

Let us consider the "ideal" case where the state equations Eqs. 1.2 and 1.4 are $f(x, i) = i$ and $g(x, v) = v$, respectively. Therefore, integrating both sides of these equations respectively gives:

$$x(t) = \int_{-\infty}^{t} i(\tau)\mathrm{d}\tau = q(t) \tag{1.5}$$

$$x(t) = \int_{-\infty}^{t} v(\tau)\mathrm{d}\tau = \varphi(t) \tag{1.6}$$

Now substituting Eqs. 1.5 and 1.6 for x in Eqs. 1.1 and 1.3 respectively, and integrating both sides, gives:

$$\varphi(t) = \int_{-\infty}^{t} v(\tau)\mathrm{d}\tau = \int_{-\infty}^{t} R(q(\tau))\frac{\mathrm{d}q(\tau)}{\mathrm{d}\tau}\mathrm{d}\tau = \int_{q(-\infty)}^{q(t)} R(q)\mathrm{d}q = \hat{\varphi}(q(t)) \tag{1.7}$$

$$q(t) = \int_{-\infty}^{t} i(\tau)\mathrm{d}\tau = \int_{-\infty}^{t} G(\varphi(\tau))\frac{\mathrm{d}\varphi(\tau)}{\mathrm{d}\tau}\mathrm{d}\tau = \int_{\varphi(-\infty)}^{\varphi(t)} G(\varphi)\mathrm{d}\varphi = \hat{q}(\varphi(t)) \tag{1.8}$$

The above equations indicate that in this degenerate special scalar case, the two equations Eqs. 1.1 and 1.2 (resp. Eqs. 1.3 and 1.4) defining a current-controlled (resp. a voltage-controlled) memristor are equivalent to a single equation:

$$\varphi = \hat{\varphi}(q) \tag{1.9}$$

for a *charge*-controlled memristor, or

$$q = \hat{q}(\varphi) \tag{1.10}$$

for a *flux*-controlled memristor.

The latter are precisely the fourth constitutive relationship shown in Fig. 1.1, defining the memristor via an axiomatic approach where the variables q and φ do not need to have precise physical significance. Differentiating Eqs. 1.9 and 1.10 with respect to time t, we obtain:

$$v = \frac{\mathrm{d}\varphi}{\mathrm{d}t} = \frac{\mathrm{d}\hat{\varphi}(q)}{\mathrm{d}q}\frac{\mathrm{d}q}{\mathrm{d}t} = R(q)i \tag{1.11}$$

and

$$i = \frac{\mathrm{d}q}{\mathrm{d}t} = \frac{\mathrm{d}\hat{q}(\varphi)}{\mathrm{d}\varphi}\frac{\mathrm{d}\varphi}{\mathrm{d}t} = G(\varphi)v \tag{1.12}$$

It follows from Eq. 1.11 that the charge-controlled memristor defined in Eq. 1.9 is equivalent to a charge-dependent Ohm’s Law where $R(q)$ is just the slope of the curve $\varphi = \varphi(q)$ at q. Of course, Eqs. 1.9 and 1.11 are equivalent and one can recover Eq. 1.9 by integrating both sides of Eq. 1.11 with respect to t.

Since experimental devices obeying the ideal constitutive relation of Eq. 1.9 or Eq. 1.10 are rather rare, most memristor prototypes will be rather modeled as generic memristive devices according to Eqs. 1.1–1.2 or Eqs. 1.3–1.4. Such model of a generic memristive device is presented in Chap. 2 and is later used in the rest of this Book. However, for terminology reasons, we will henceforth refer to all such devices as memristors and call only the fourth circuit element of Fig. 1.1 as an “ideal” memristor whenever a distinction is required.

The chapters that follow span a wide range of memristor-related topics and give a good overview of the ongoing research and the current trends in this exciting scientific field.

References

1. L.O. Chua, Memristor—the missing circuit element. IEEE Trans. Circuit Theory **18**(5), 507–519 (1971)
2. L.O. Chua, S.M. Kang, Memristive devices and systems. Proc. IEEE **64**(2), 209–223 (1976)

3. Y.V. Pershin, M. Di Ventra, Memory effects in complex materials and nanoscale systems. Adv. Phys. **60**(2), 145–227 (2011)
4. L.O. Chua, Resistance switching memories are memristors. Appl. Phys. A Mater. Sci. Process. **102**(4), 765–783 (2011)
5. V. Erokhin, Organic memristors: basic principles, in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Paris, France (2010)

Chapter 2
Memristor Modeling

2.1 Introduction

Since the exciting discovery of nonvolatile memristive behavior in Titanium dioxide (TiO_2)-based nano-films at Hewlett Packard (HP) Labs in 2008 [1], both academia and industry have been engaged in the search for novel memristive materials and manufacturing technologies. HP's version of the TiO_2 substrate memristor remains up to now the most generally recognized memristor type. It is based on two thin-layer TiO_2 films. The bottom layer acts as an insulator whereas the top film layer acts as a conductor via oxygen vacancies in the TiO_2; TiO_2 changes its resistance in the presence of oxygen. Voltage increment moves the oxygen vacancies from the top layer towards the bottom layer, thus changing its resistance. A great deal of ongoing work has been devoted to the development of mathematical models capable to capture the complex dynamics exhibited by these nanostructures. An appropriate descriptive model will not only lead to a better understanding of its behavior, but will also result to a better exploitation of its unique properties in novel systems and architectures combining data storage and data processing in the same physical location.

Currently there are several available device models which attempt to characterize both current–voltage (*i*–*v*) behavior as well as the device dynamics [2–9]. The HP group, in their first memristor implementation announcement, suggested a coupled variable-resistor model for memristors [1]. This model was later improved by Joklegar and Wolf [10], whereas several papers by HP [11, 12] reported on further developments of resistance switching theory for TiO_2-based devices. Nevertheless, until nowadays there has been no direct connection between a model and the memristor physical properties. Only a few models were derived on the basis of material characterization and experimental electronic measurements, thus giving some hint on the physical mechanisms at the origin of the unique behavior of memristors [9, 13]. However, given the complexity of the physical processes that occur in the devices, the corresponding detailed mathematical descriptions are

I. Vourkas and G.Ch. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Emergence, Complexity and Computation 19,
DOI 10.1007/978-3-319-22647-7_2

usually far too complex to solve analytically and numerical solutions are too time consuming to include in a simulation. Moreover, since simulation with Simulation Program with Integrated Circuit Emphasis (SPICE) is common practice in circuit development, several models of memristors were also implemented in SPICE [2, 5–8, 14–20].

The study of some of the most noteworthy published memristor models has shown that simple models are able to reproduce most of the dynamics observed with more accurate models, whose far greater computational complexity may lead to convergence problems and instability issues in complex circuits [21]. For example, the original linear oxygen vacancy drift model proposed by HP is valid only for certain choices of input signals and initial state of the memristance. Furthermore, a common problem in most models is that there is no threshold consideration. Threshold-type switching, though, is an extremely important common feature of the majority of experimental memristive devices. Physical memristor devices demonstrate a threshold voltage where hysteresis is not seen unless the voltage across the memristor exceeds the threshold [22]. Another important feature concerns the switching speed of memristors during the "set" and "reset" operations which generally are not similar [9, 22]. According to characterization data from HP Labs, the motion of the memristor state variable depends both on its current state and on the polarity of the applied voltage [9], something which could be attributed to the interaction of the external applied field, the internal field of the concentrated defects (e.g. charge traps, mobile ions, oxygen vacancies, etc.), and the diffusion, all acting in the same or in the opposite directions according to the applied voltage.

In the rest of this chapter we present a SPICE-compatible device model [23] of a voltage-controlled memristor which explains memristive behavior while primarily attributing the switching effect to an effective tunneling distance modulation [24]. This model aims to address most of the aforementioned shortcomings; it satisfies the desired memristive fingerprints [25] and involves significantly low-complexity operation under an unlimited set of frequencies over a wide range of applied voltages. The SPICE simulation results are found in good qualitative and quantitative agreement with the theoretical formulation of the model [26]. Also, the model represents well the complex switching behavior of memristor when fitted to other widely used published models. Therefore, it can be used to provide accurate enough circuit simulations for a wide range of memristor devices and voltage inputs, while it can be incorporated as a circuit element in any current computer-aided memristor-based circuit design work.

2.2 A Novel Threshold-Type Memristor Circuit Model

Inspired from the original circuit model proposed by HP for TiO_2-based devices, the equivalent circuit of the proposed memristor model is depicted in Fig. 2.1.

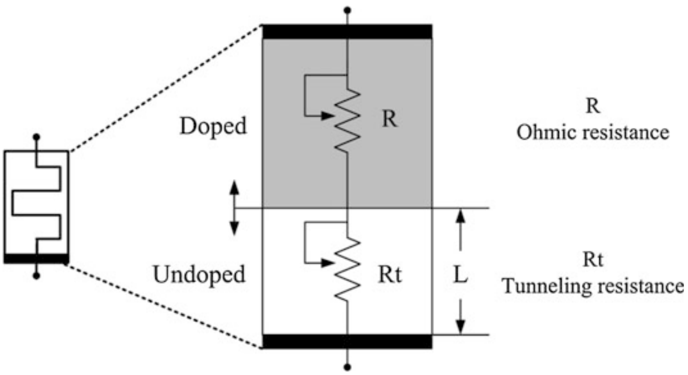


Fig. 2.1 Equivalent circuit of the coupled ohmic-tunneling variable-resistor circuit model

It concerns a threshold-type switching model of a two-terminal voltage-controlled electrical device that exhibits memristive behavior, whose general definition is given by the following equations:

$$I(t) = G(L,t)V_M(t) \tag{2.1}$$

$$\dot{L} = f(V_M, t). \tag{2.2}$$

Parameter L denotes the single state variable of the system (indicating the internal memristor state), which in our model is the tunnel barrier-width (e.g. the thickness of the free of oxygen vacancies dioxide layer), with the electrical current transport process being limited primarily by tunneling through it. G is the conductance (memductance) of the device, whereas I and V_{M} represent the flowing current and the applied voltage, respectively. In the coupled ohmic-tunneling variable-resistor equivalent circuit of Fig. 2.1, we consider an ohmic variable-resistor R and a tunneling variable-resistor Rt connected in series. R represents the resistance of the doped dioxide layer and Rt represents the tunneling resistance of the undoped layer of the device. The doped layer acts as a conductor whereas the undoped layer is a pure insulator. Therefore, there is a significant difference between the actual values of their resistances, with $Rt \gg R$, which is the reason why the model concentrates mainly on Rt.

The tunneling resistance Rt is expected to be proportional to the tunnel barrier width L, given the fact, that the larger the barrier width the higher the resulting resistance should be. Also, its value is anticipated to change according to the "movement" of the boundary between the two layers because of the transport of oxygen deficiencies under positive or negative applied voltage. Thus, any mathematical formulation for Rt could include at least a fitting parameter which would bound the effect of the varying geometry of the device on the actual concentration of the oxygen vacancies in either the doped or the undoped side of the film.

Furthermore, according to Schiff [24], *Rt* is inversely proportional to the product of the voltage-dependent tunneling transmission coefficient (T_0) and the electron effective density of states (N_{eff}), whereas it is exponentially proportional to the tunnel barrier-width (L). Therefore, its particular mathematical formulation is:

$$Rt(V_M) = \frac{1}{N_{eff}} \cdot \frac{e^{2k_{V_M}L}}{T_{0,V_M}}. \tag{2.3}$$

The voltage dependence of Eq. 2.3, due to the presence of the voltage-dependent parameters T_0 and k, can be translated into a corresponding variation of L; it can be passed to a new voltage-dependent parameter $L_{V,t}$ with no significant error implication. In this model we define *Rt* to be described by the following equation:

$$Rt(L_{V_M,t}) = f_0 \cdot \frac{e^{2L_{V_M,t}}}{L_{V_M,t}}. \tag{2.4}$$

Equation 2.4 gives the resistance (memristance) of the device for a certain restricted range of the state variable L. All unknown material-specific and geometrical issues are contained into the model-fitting constant parameter f_0. The qualitative agreement of Eqs. 2.3 and 2.4 verifies our assumption for the exponential dependence of *Rt* on L. Moreover, Pickett et al. in [9] reported on experimental results from the application of a dynamical testing protocol applied to a set of TiO_2-based memristive devices. Through analysis of the switching dynamics that arise from ionic motion in the devices, it was concluded that electronic conduction in these devices is dominated by an effective tunneling barrier width that varies with time under the applied voltage. Thus, the switching effect is primarily attributed to an effective tunneling distance modulation, which is in line with the present assumptions for the *Rt*-*L* dependence.

A heuristic equation $L(V_M, t)$ that qualitatively gives the expected response of L as a function of the time t and the applied voltage V_M is given below:

$$L(V_M, t) = L_0 \cdot \left(1 - \frac{m}{r(V_M, t)}\right). \tag{2.5}$$

L_0 is the maximum value that L can attain. The term in parenthesis of Eq. 2.5, which contains a voltage-dependent parameter $r(V_M, t)$ and a fitting constant parameter m, determines the boundaries of the barrier width. By considering tunneling as the dominant physical mechanism, Eq. 2.5 introduces the initial as well as the current position of L which is limited within two boundary values. Parameter $r(V_{\mathrm{M}}, t)$ defines both the device dynamics and the current state of the device. Its value is monitored and maintained within a valid range; i.e. when $r < r_{\mathrm{MIN}}$ or $r > r_{\mathrm{MAX}}$, it is set equal to r_{MIN} or r_{MAX}, corresponding to L_{MIN} and $L_{\mathrm{MAX}} \approx L_0$, respectively. As a consequence, the memristance is correspondingly set to the most (R_{ON}) or the least conductive state (R_{OFF}) via Eq. 2.4. Values for parameters m and r_{MIN} should be selected so that the fraction (m/r_{MIN}) < 1 (so, the tunnel barrier-width will never be zero).

Furthermore, since "set" and "reset" switching times can differ in many experimental memristive devices, this model is based on the assumption that the switching rate of L is small (fast) below (above) a threshold voltage (V_{SET} or V_{RESET}), which is viewed as the minimum voltage required to impose a change on the physical structure, and thus the memristance, of the device. This assumption is encapsulated in the use of the voltage-dependent parameter $r(V_M, t)$, whose time derivative is slow or fast depending on the applied voltage, as shown below:

$$\dot{r}(V_M,t) = \begin{cases} a_{RESET}\cdot\frac{V_M+V_{RESET}}{c+|V_M+V_{RESET}|}, & V_M\in[-V_0,V_{RESET}) \\ b\cdot V_M, & V_M\in[V_{RESET},V_{SET}] \\ a_{SET}\cdot\frac{V_M-V_{SET}}{c+|V_M-V_{SET}|}, & V_M\in(V_{SET},+V_0] \end{cases}. \tag{2.6}$$

Equation 2.6 comprises one-parameter sigmoid functions for the regions above the thresholds (first and last branch), whereas a linear relation of the applied voltage is used for the region below the thresholds. Parameters a_{RESET}, a_{SET}, b, and c are fitting constants that are used to shape the intensity of the state variable dynamics, i.e. the rate of memristance change, with $a_{\mathrm{x}} \gg b$ and $0 < c < 1$. Setting $b = 0$ imposes a hard switching behavior, i.e. there is no state change in the memristor unless a certain voltage threshold is exceeded. Different thresholds and switching rates can be programmed by tuning the shaping parameters of $r(V_{\mathrm{M}}, t)$; a different set of values for the parameters $\{a_{\mathrm{x}}, b, c, m\}$ defines a different set of boundaries for the tunnel barrier-width. The model parameters are certainly determined by material properties of the modeled memristor, such as the effective tunneling distance, etc. However, here they are regarded as fitting parameters that yield visibly different i–v curves. Note that Eqs. 2.4–2.6 are written in such a way that when $\{a_{\mathrm{x}}, b\} > 0$ then a positive (negative) voltage applied to the top terminal with respect to the bottom terminal (denoted by the black thick line in the memristor circuit schematic), tends to decrease (increase) the memristance of the device.

Figure 2.2 qualitatively shows the simulation results for the response of a memristor under sinusoidal applied voltage according to the proposed model (the effect that the different frequencies of the applied voltage have on the switching behavior will be discussed later). In the graphical representation of Eq. 2.6 in Fig. 2.2e, the two separate sigmoid functions were included to facilitate visual correspondence. It is obvious that in the region $[-V_0, V_{th})$ the black line follows the green sigmoid graph whereas in the region $(V_{th}, V_0]$ it follows the red graph.

Figure 2.3 shows some calibration options offered by the model. More specifically, Fig. 2.3a, b show how the memristance range can be adjusted by modifying the L_0 and f_0 parameter values. A higher L_0 enlarges the [R_{ON}, R_{OFF}] memristance range in an exponential manner, whereas different values for the parameter f_0

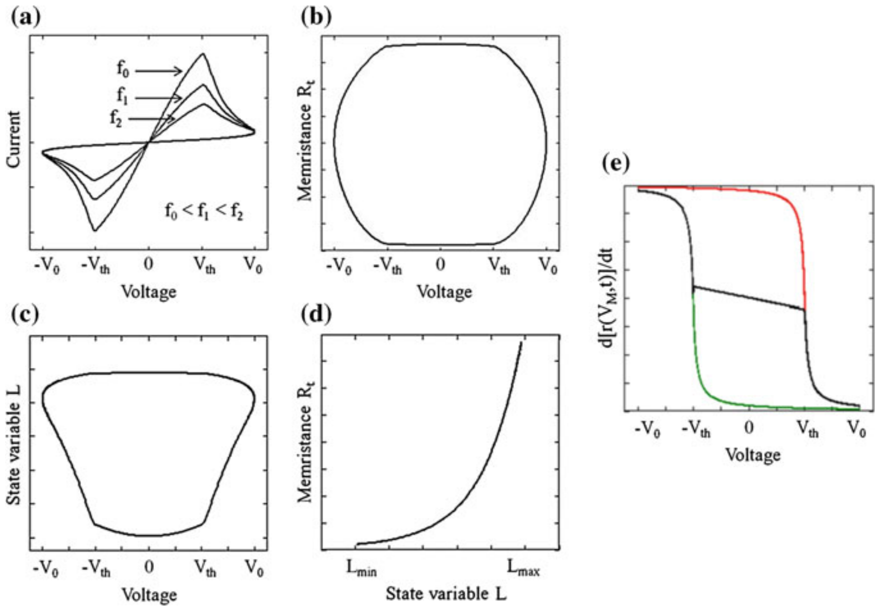


Fig. 2.2 **a** *I*–*V* characteristic of a memristor under AC voltage $V(t) = V_0 \cdot \sin(2\pi ft)$ for different frequencies $f_0 < f_1 < f_2$ of $V(t)$ with threshold voltages $V_{\mathrm{RESET}} = V_{\mathrm{SET}} = V_{\mathrm{th}}$. **b** The memristance *Rt* with the applied voltage. **c** Response of the state variable *L* according to the applied voltage. **d** The memristance *Rt* for a restricted range of *L* according to Eq. 2.4. **e** Graphical demonstration of Eq. 2.6; in the regions above the thresholds the *black line* follows either the *green* (region $[-V_0, -V_{\mathrm{th}})$) or the red (region $(V_{\mathrm{th}}, V_0]$) sigmoid function

displace equally the above range so that $[R_{\mathrm{ON}}, R_{\mathrm{OFF}}]_{\mathrm{NEW}} = (f_{0,\mathrm{NEW}}/f_0) \times [R_{\mathrm{ON}}, R_{\mathrm{OFF}}]$. Except the threshold voltages V_{RESET} and V_{SET} which can be set asymmetric, since α is the max value for the rate of change of parameter *r*, different a_{RESET} and a_{SET} can lead to different switching times which depend on the polarity of the applied voltage.

The time derivative of the state variable in Eq. 2.2 is interpreted as the speed of movement of the barrier between the two layers due to the applied voltage. However, several memristive devices have been proposed using different material structures [22], so the resistance switching mechanism is not always due to the change in thickness of a specific material layer. This model has the potential to describe memristive functionality in a more generalized way if the state variable is normalized between 0 and 1. This can be done by dividing $L(V_M, t)$ with L_0 and by multiplying with L_0 the exponent and also the denominator of Eq. 2.4. Therefore, when $L \approx 0$ the memristor is in the most conductive state, whereas the least conductive state occurs when $L \approx 1$ (instead of $L \approx L_0$). This change in the state variable represents a generalization of the model so that it can represent more types of memristive devices.

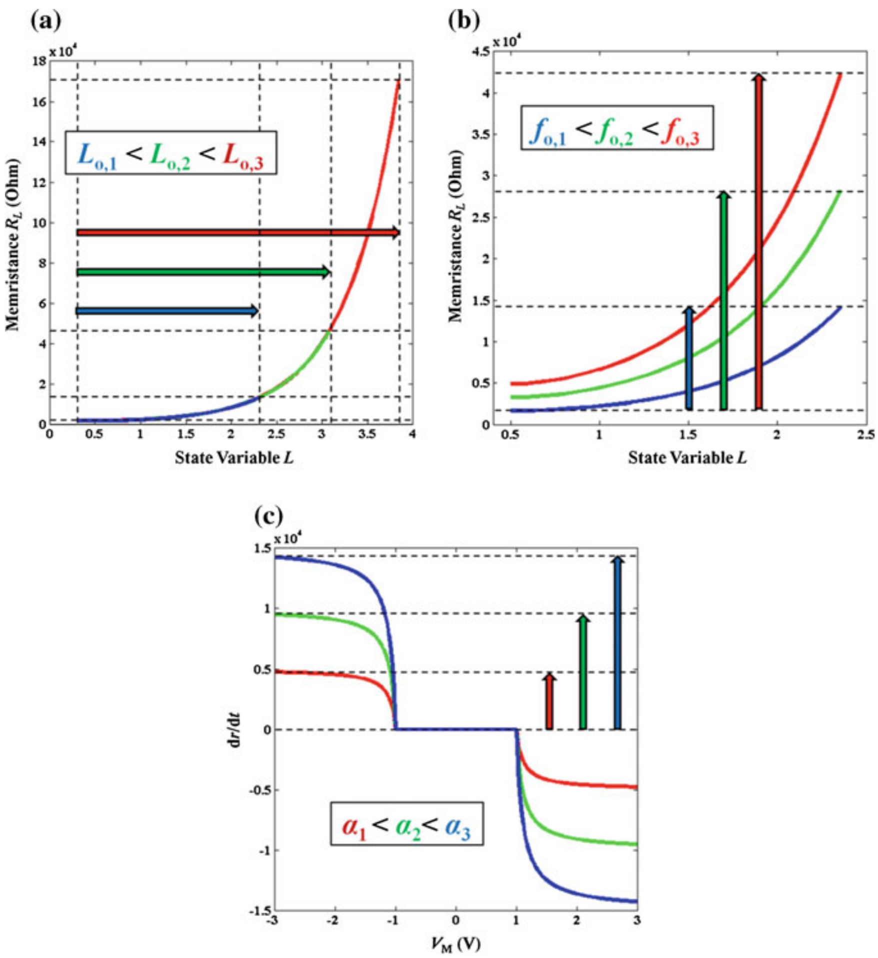


Fig. 2.3 Model calibration options. **a** The effect of different L_0 parameter values on the memristance range. **b** The effect of different f_0 parameter values on the memristance range. **c** The effect of different a parameter values on the rate of change of parameter r

2.3 Modeling Memristors in SPICE

We developed a behavioral model of a memristor at device level using the SPICE circuit description language by following the mathematical equations presented before [26]. We implemented the voltage-controlled memristor model into a simple netlist where the memristive device is realized as a sub-circuit consisting of several elements, thus making it easy to comprehend and ready to be used in memristor-based systems.

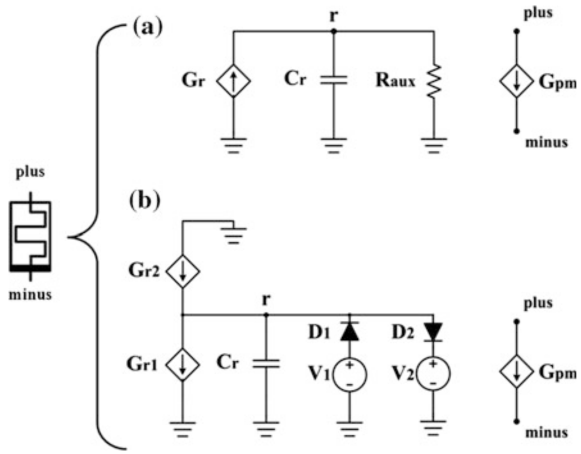


Fig. 2.4 Two different equivalent versions of the corresponding circuit schematic of the SPICE memristor model

The circuit layout for the SPICE model based on Eq. 2.1 and on Eq. 2.4 through Eq. 2.6 is shown in Fig. 2.4, where two different versions are presented. Memristor SPICE models have been previously proposed using a similar setup in [14]. In Fig. 2.4a the memristive system is realized as a sub-circuit combining two current sources G_{pm} and G_r, an integrating capacitor C_r (modeling the memory effect of the memristor) and a resistor R_{aux}. This is the most compact corresponding schematic. The current source G_r generates a current based on Eq. 2.6. The voltage across the capacitor (at node *r*: V_r) defines the value of parameter $r(V_M, t)$. In both versions the two terminals (*plus* and *minus*) of the additional current source G_{pm}, which plays the role of a behavioral resistor, represent the top and bottom electrodes of the device. The output of the current source G_{pm} is set using the voltage drop across the terminals of the device and the memristance given by Eq. 2.4. However, in this setup, $r(V_M, t)$ can step out of the valid interval, which would yield invalid and unstable solution. Therefore, an appropriate smoothing function, which takes this into account, is necessary to avoid convergence problems. The purpose of such function is to limit $r(V_M, t)$ inside the valid value interval between the defined boundaries r_{MIN} and r_{MAX}. The exact use of the aforementioned function can be seen in the respective SPICE netlist in Table 2.1.

In Table 2.1 the first lines briefly comment on the most important parameters of the model. Initialization of the parameters takes place in lines 3–4 and the selected values for the parameters $\{r_{MIN}, r_{MAX}, L_0, m, f_0\}$ provide a resistance ratio of two orders of magnitude with $\{R_{ON}, R_{OFF}\} = \{2, 200\}$ kΩ. In the first netlist, line 7 specifies the capacitor C_r with an initial condition. By setting the initial value of the voltage across the capacitor *rinit* equal to either of the boundary values (or to any valid value in between) we indicate the initial state of the device. The value of the current source declared in line 5 is equal to the right hand side of Eq. 2.6, where the smoothing function *st_f*(·) (step function) is used to define which branch of Eq. 2.6 applies each time according to the applied voltage at the terminals of the device. Line 10 describes the current source G_{pm} which defines the current running through

Table 2.1 Voltage-controlled memristor SPICE model netlists

```
* rmin,rmax : Boundary values for r             *
* rinit     : Initial value of f                *
* Lo        : Tunnel barrier width              *
* VtL,VtR   : Left and Right Threshold voltages *
* alpha,                                        *
* beta,gamma: Parameters for modeling non-linear *
*             threshold-based behavior          *
* m,fo      : Model's fitting parameters        *
* yo        : Smoothing function's parameter    *
************************************************
```

```
     *Netlist corresponding to Fig. 2.4(a)
 1   .subckt memristor plus minus PARAMS:
 2   *Parameters’ values
 3   +rmin=100 rmax=390 rinit=390 alpha=1E15
     beta=10 gamma=0.1 VtR=1.5 VtL=-1.5 yo=0.0001
 4   +m=82 fo=310 Lo=5
 5   Gr 0 r value={dr_dt(V(plus)- V(minus))*(
     st_f(V(plus)-V(minus))*st_f(V(r)-rmin)+
 6   +st_f(-(V(plus)-V(minus)))*st_f(rmax-V(r)))}
 7   Cr r 0 1 IC={rinit}
 8   Raux r 0 1E12
 9   *Current equation Imem = V / R(L)
10   Gpm plus minus value={(V(plus)-
     V(minus))/((fo*exp(2*L(V(r))))/L(V(r)))}
11   *Func. for non-linear threshold-based behavior
12   .func dr_dt(y)={-alpha*((y-VtL)/(gamma+abs(y-
     VtL)))*st_f(-y+VtL)-beta*y*st_f(y-VtL)*
13   +st_f(-y+VtR)-alpha*((y-VtR)/(gamma+abs(y-
     VtR)))*st_f(y-VtR)}
14   *Smoothing function
15   .func st_f(y)={1/(exp(-y/yo)+1)}
16   *L(V) function
17   .func L(y)={Lo-Lo*m/y}
18   .ends memristor
```

```
     *Netlist corresponding to Fig. 2.4(b)
 1   .subckt memristor plus minus PARAMS:
 2   *Parameters values
 3   +rmin=100 rmax=390 rinit=390 alpha=1E15
     beta=10 gamma=0.1 VtR=1.5 VtL=-1.5 yo=0.0001
 4   +m=82 fo=310 Lo=5
 5   Gr1 0 r value={dr_dt(V(plus)-
     V(minus))*st_f(-(V(plus)-V(minus)))}
 6   Gr2 0 r value={dr_dt(V(plus)-
     V(minus))*st_f(V(plus)-V(minus))}
 7   D1 k r Dbreak
 8   V1 k 0 {rmin}
 9   D2 r g Dbreak
10   V2 g 0 {rmax}
11   Cr r 0 1 IC={rinit}
12   *Current equation Imem = V / R(L)
13   Gpm plus minus value={(V(plus)-
     V(minus))/((fo*exp(2*L(V(r))))/L(V(r)))}
14   *Func. for non-linear threshold-based behavior
15   .func dr_dt(y)={-alpha*((y-tL)/(gamma+abs(y-
     VtL)))*st_f(-y+VtL)-beta*y*st_f(y-VtL)*
16   +st_f(-y+VtR)-alpha*((y-VtR)/(gamma+abs(y-
     VtR)))*st_f(y-VtR)}
17   *Smoothing function
18   .func st_f(y)={1/(exp(-y/yo)+1)}
19   * L(V) function
20   .func L(y)={Lo-Lo*m/y}
21   .ends memristor
```

Content of lines without numbers continues from previous lines.

the device according to the applied voltage and the memristance given by Eq. 2.4. The resistor R_{aux}, described in line 8, has an auxiliary role. It is used to model the memory retention capability, which is an important aspect of experimental memristor realizations, thus taking into account the case where memristance can change over time even when no voltage is applied [27]. The desired changing pace depends on the particular value of the resistor. R_{aux} does not affect the switching behavior of the device when being accessed; hence it can be omitted if retention is not considered.

Figure 2.4b shows a more thorough way of modeling both the memristive effect as well as the control of the boundary conditions. Here, the current source G_r is replaced by two current sources G_{r1} and G_{r2} which have opposed polarities and operate in such a way so that G_{r2} is responsible for charging the capacitor, and G_{r1} for discharging it. Their operation is better understood in the corresponding netlist shown in Table 2.1, where the necessary step functions are used to determine which source is active each time, according to the applied voltage. Moreover, the problem of limiting the boundaries of $r(V_M, t)$ is here addressed by using elementary SPICE diodes and DC voltage sources. More specifically, two more circuit branches are added to the initial sub-circuit of Fig. 2.4a, each one comprising a diode with a specific polarity and a DC voltage source. Their role is summarized as follows: if the voltage across the capacitor V_r [i.e. the value of parameter $r(V_M, t)$] falls below V_1 (rises above V_2) then diode D_1 (D_2) is forward biased and thus V_r is maintained equal to V_1 (V_2). In this setup we have set the values of the DC sources equal to the boundary values of $r(V_M, t)$; i.e. $V_1 = r_{\mathrm{MIN}}$ and $V_2 = r_{\mathrm{MAX}}$. However, since there is a value for the forward voltages of the diodes, the user either has to adjust their corresponding internal parameters and threshold values, or has to accept a slightly shifted value for the modeled borderlines of V_r. In the corresponding netlist presented in Table 2.1, the current sources G_{r1} and G_{r2} are declared in lines 5–6, whereas the circuit elements responsible for controlling the boundary conditions are defined in lines 7–10.

The second version of the model does not have the auxiliary resistor, which however can be included in order to extend the modeling capabilities by taking into account state retention, as mentioned before. Both presented versions of the SPICE equivalent circuit were tested and the simulation results were found identical.

The SPICE implementation was tested using the Cadence PSPICE simulation environment. Figure 2.5 illustrates the presented model response to a 3 V and 100 Hz sinusoidal voltage applied for a set of consecutive waveform periods. Existence of thresholds is obvious at the hysteretic *i*–*v* graph, whereas the nonlinear conducting behavior is also noted in the *i*–*t* characteristic. The voltage across the capacitor V_r is successfully restricted within the desired boundaries, which guarantee the stable operation of the model within the valid memristive region defined in the range $\{R_{\mathrm{ON}}, R_{\mathrm{OFF}}\} = \{2, 200\}$ kΩ.

Furthermore, we shortly remind here the fingerprints of all memristors and memristive devices [25, 28] which were described previously in Sect. 1.3 of Chap. 1. The first characteristic is the pinched hysteresis loop which must hold for all amplitudes, for all frequencies, and for all initial conditions of any periodic

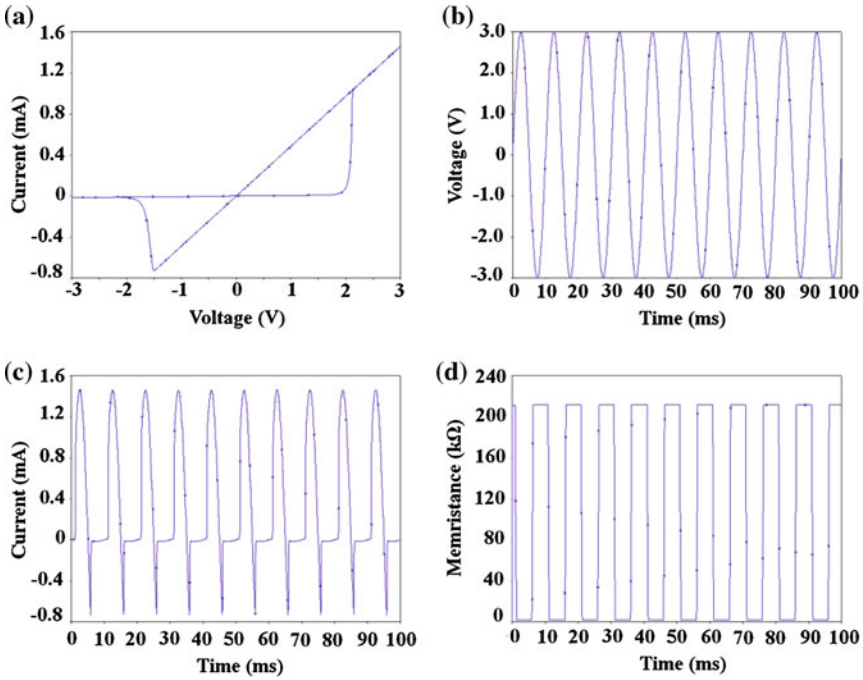


Fig. 2.5 SPICE model response to a 3 V and 100 Hz sinusoidal applied voltage. Values of the model parameters are used as given in Table 2.1 with *alpha* = 1e6. **a** The *i*–*v* characteristic shows the existence of threshold voltages around |1.5| V. The **b**, **c**, and **d** plots illustrate the applied voltage, the flowing current, and the memristance as a function of time

waveform which assumes both positive and negative values over each period (it is also pinched at the origin for any non-sinusoidal periodic waveform). The other fingerprint is that, for high frequencies of the applied periodic signal, the *i*–*v* loop collapses to a straight line. Thus, any considered memristor device model, based on an explicit memristive mechanism, should be capable of delivering such properties.

To this end, we include in Fig. 2.6 the dynamic response of the model to the application of voltage signals of different frequencies and also to the application of consecutive positive and negative pulses. Increasing the frequency of the external voltage leads to decreased hysteretic behavior of the memristor until it asymptotically passes over to the characteristic curve of a conventional resistor. The effect of the different frequencies (100, 110, 150 Hz) are depicted in sub-figure (a) where it can be seen that the memristive effect diminishes as the frequency grows. Thus, the *i*–*v* characteristic of a memristor degenerates to a straight line because the device is not given the necessary time to change its resistance while being biased. Also, sub-figure (b) shows the simulation results of the model when several consecutive sinusoidal voltage pulses are applied to the device in a stepwise manner. The input pulses are applied multiple times with the same polarity to study how the model switches to intermediate levels between the maximum and minimum resistance.

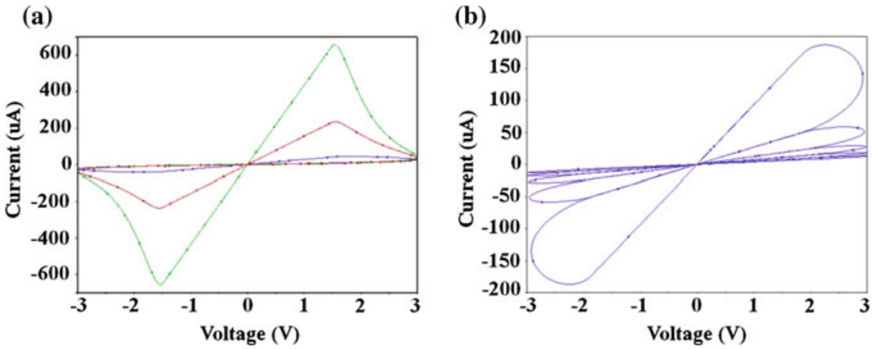


Fig. 2.6 Memristor SPICE model response to **a** a sinusoidal applied voltage of 3 V and 100 Hz (*green*), 110 Hz (*red*) and 150 Hz (*blue*), respectively, and to **b** the application of stepwise 3 V and 4 Hz consecutive sinusoidal voltage pulses. Parameter values are used as given in Table 2.1 with *alpha* = 1e5 in (**a**) and *alpha* = 1e3 in (**b**)

The first five positive voltage pulses correspond to the right half of the *i*–*v* curve, where the memristance continually decreases. For the rest of the simulation the current is negative and the opposite trend is seen. Therefore, no matter which the initial condition is, the hysteretic *i*–*v* loop is always pinched to the origin of the axes. So, the presented SPICE-compatible model complies adequately with the desired memristive fingerprints.

Another important issue, concerning the majority of existing modeling approaches being currently pursued by the design community, is that it is not always known how closely the SPICE models match the respective theoretical models on which they are based. The presented approach constitutes a highly parameterizable generalized model which has a direct correlation to the theoretical model that it was designed to match. Figure 2.7 indicates the *i*–*v* hysteretic curves obtained both from the theoretical model implemented in MATLAB and from the corresponding SPICE implementation. The selected values for the set of adjustable parameters of the model, used to generate the simulation scenarios of Fig. 2.7, were taken equal in both cases to indicate compliance of the SPICE model with the theoretical model. Two simulation cases of a memristor under AC applied voltage of either low or high frequency, with symmetric or asymmetric thresholds, are shown. The simulation results are found in very good qualitative and quantitative agreement.

2.4 Model Verification

2.4.1 Fitting to a Reference Model

As it has been shown so far, the hysteretic *i*–*v* curve obtained from simulation of a memristor under AC applied voltage using the proposed model, exhibits the expected

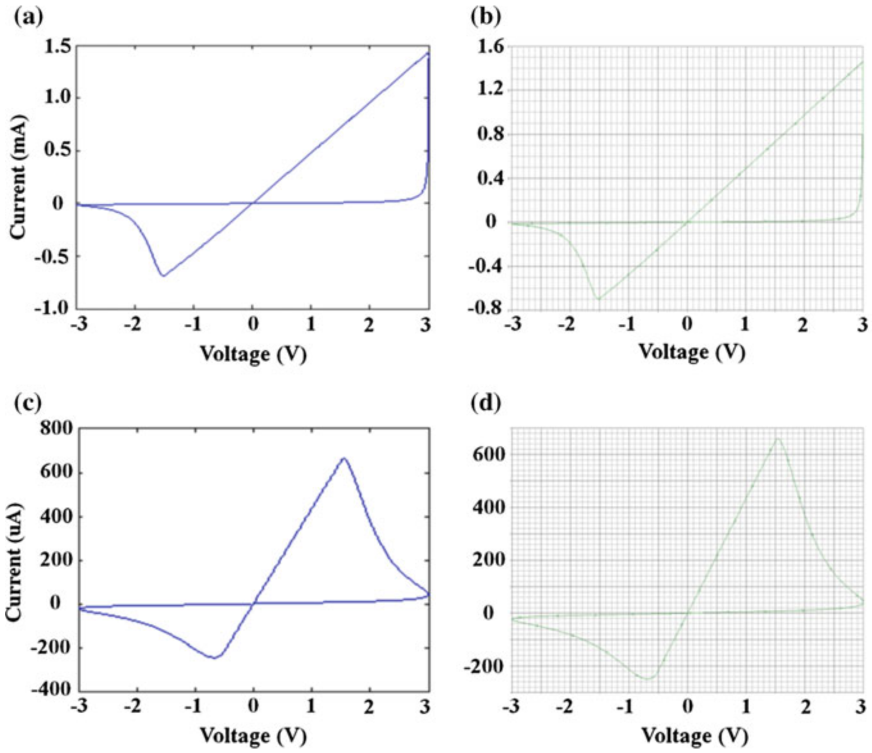


Fig. 2.7 SPICE implementation compliance with the theoretical model. **a**, **b** Comparison between **a** MATLAB and **b** SPICE implementations for the model parameters values as given in Table 2.1 with *alpha* = 1e3 and signal frequency $f = 0.5$ Hz. **c**, **d** Comparison between **c** MATLAB and **d** SPICE implementations under different threshold voltages and frequency, with *alpha* = 1e6 and $f = 1$ kHz

"bow tie" shape. In order to illustrate its versatility, in Fig. 2.8 we show the *i*–*v* and *M*–*v* (*M*-Memristance) characteristics calculated using the presented model and the model proposed in [10], which is used as a reference. The latter is a widely used extension of the linear ionic drift model proposed by HP [1], where a particular window function was incorporated to illustrate nonlinearities in ionic transport.

More specifically, the memristor is modeled as a thin oxide film of length *D* comprising a conductive layer of oxygen-deficient Titanium dioxide with length *w* (chosen as the state variable) serially connected with an insulating layer of stoichiometric Titanium dioxide of length *D*–*w*. The memristor input–output relation is modeled as:

$$v = M\left(\frac{w}{D}\right)i \tag{2.7}$$

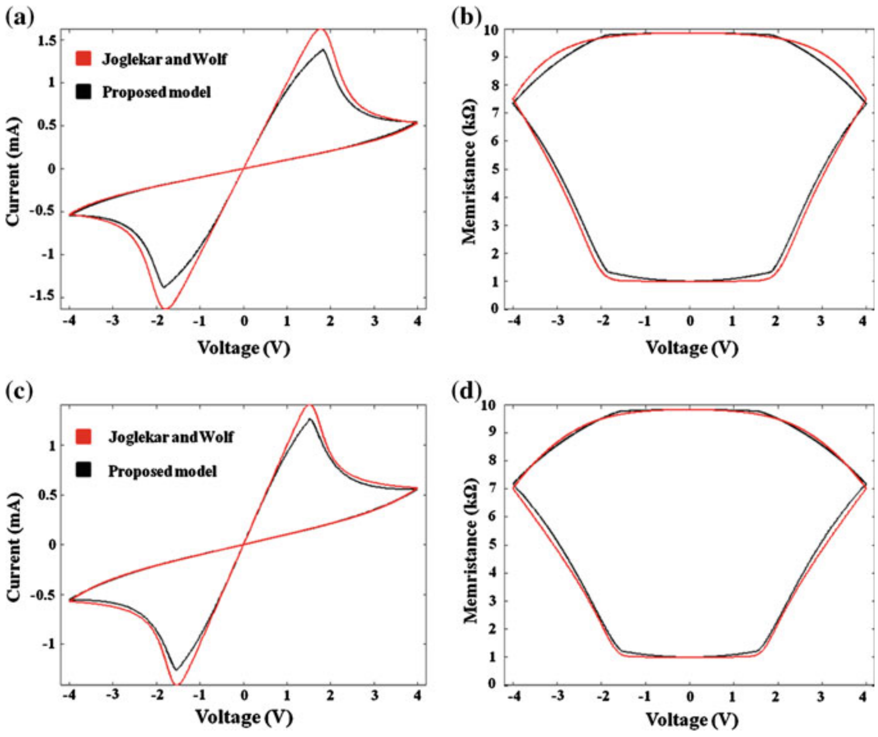


Fig. 2.8 Calculated *i–v* **a**, **c** and *M–v* **b**, **d** characteristics for memristors simulated using the presented model and the model of Joklekar and Wolf [10]. Our model successfully reproduces the characteristic responses of the selected reference model

where $M(\cdot)$ denotes the memristance function defined as:

$$M\left(\frac{w}{D}\right) = R_{OFF} - \Delta R\frac{w}{D} \tag{2.8}$$

where R_{ON} and R_{OFF} are the memristor boundary resistances when the whole nano-film is respectively enriched and depleted with oxygen vacancies, whereas $\Delta R = R_{\mathrm{OFF}} - R_{\mathrm{ON}}$. The equation governing the time evolution of the state is:

$$\frac{\mathrm{d}w}{\mathrm{d}t} = \mu\frac{R_{ON}}{D}f(w,i)i \tag{2.9}$$

where μ denotes the average mobility of the oxygen vacancies, whereas $f(w, i)$ is the window function which was introduced to take into account boundary behavior and various nonlinear dynamical effects such as nonlinear oxygen vacancy drift.

In [10] the window function $f(\cdot)$ takes values within [0, 1], is independent of the current, and may assume its maximum unitary value when $w = ½ \times D$ according to

$$f(w,i) = f(w) = 1 - \left(\frac{2w}{D} - 1\right)^{2p} \qquad (2.10)$$

where p is a positive integer controlling the rate of decrease of the state variable as it approaches either of the boundaries.

In order to obtain a fairer comparison, wherever it applies we use the same parameters for both models. In specific, we use an 8 V peak-to-peak triangular AC voltage of period $T_1 = 2.6$ s and $T_2 = 5.5$ s to simulate memristors with total width $D_1 = L_{0,1} = 3$ nm and $D_2 = L_{0,2} = 5$ nm, respectively. We consider a R_{OFF}/R_{ON} ratio of ≈10, a dopant-mobility of 3×10^{-8} $m^2/(Vs)$ [29] and we set the exponent variable of the window function $p = 2$ [10]. Figure 2.8 summarizes the simulation results for both the first (a, b) and the second (c, d) memristor while fitting the presented model to the reference model. In each simulation scenario we set the parameters of our model $\{a_x, b, c, f_0, m, V_{SET} = |V_{RESET}|\}$ to the values {1000, 50, 0.1, 86.49, 56.06, 1.7 V} and {350, 20, 0.1, 2.67, 29.97, 1.5 V}, respectively. In both cases our model delivers satisfying quantitative results which closely coincide with the results of the reference model. The small difference in the maximum observed currents is attributed to the slightly different moments when the maximum memristance is achieved, as particularly shown in Fig. 2.8b, d.

2.4.2 *Testing in Complex Memristive Circuits*

Up to this point we have thoroughly examined the dynamic behavior of the model when considering a single device under AC applied voltage. However, we have further studied the functionality of the model when simulating multiple memristive elements combined together. Using a single voltage-dependent current source between the memristor terminals (as shown in Fig. 2.4), which is a common practice also followed by other models in the literature (e.g. in [19]), might be problematic when trying to combine more than one memristors in complex resistive (ohmic) networks. As shown in Fig. 2.9a, when trying to connect two devices in the simplest in-series configuration, the resulting circuit branch contains two ideal current sources connected in series, which is not acceptable. In order to get over this shortcoming, we elaborated the presented model and replaced the single ideal current source with a real current source, i.e. a current source which has an additional parallel resistor. The value of the resistor must be high enough for the whole device to better approximate the ideal current source, affecting as less as possible the total current flowing through the memristor. During simulations this value is set to $1000 \times R_{OFF}$.

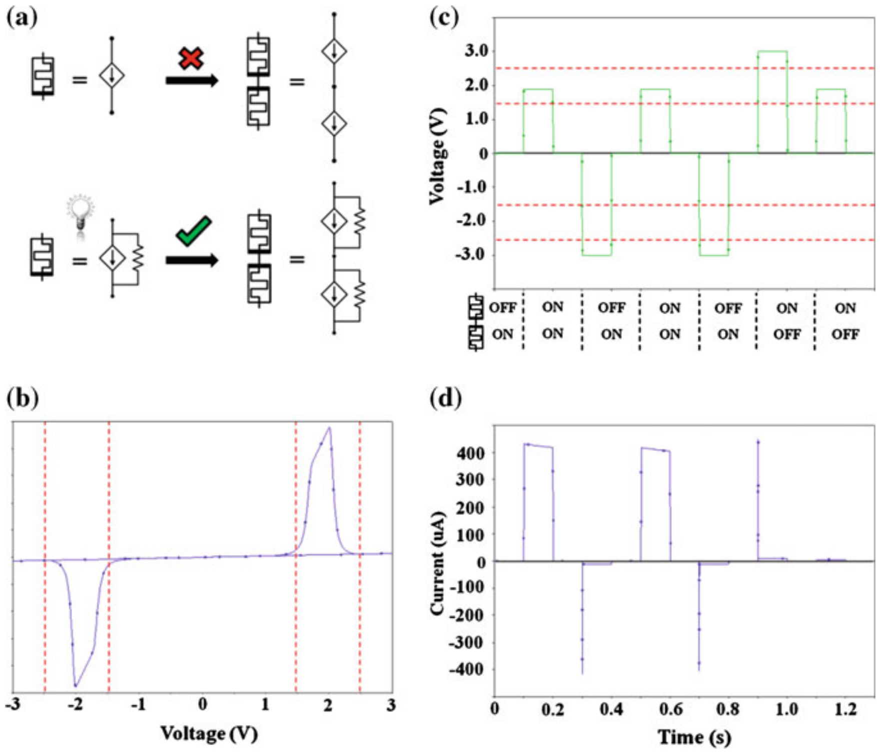


Fig. 2.9 SPICE simulation of a complementary resistive switch (CRS). **a** The model adjustment introduced to allow the in-series connection of memristors. **b** The CRS response to a 3 V and 100 kHz sinusoidal applied voltage. The *red-dotted lines* designate the area where the state-transitions take place. **c** The series of applied programming (±3 V) and read-out (1.9 V) 100 ms duration voltage pulses. **d** The corresponding measured currents denoting the different resistive states of the CRS. In between **c** and **d** graphs there are the device transitions which take place every time a voltage pulse is applied. Values of the parameters of the model are the same as given in Table 2.1 with *alpha* = 1e7 and threshold voltages $V_{SET} = |V_{RESET}| = 1$ V

We simulated a circuit branch comprising two memristors connected in-series with opposite polarities, forming a complementary resistive switch (CRS) [30]. In the CRS concept, a memory cell is formed by two bipolar memristive elements vertically stacked in an anti-serial manner. The unique aspect of this device combination is in using a series of R_{OFF} and R_{ON} states to represent a stored ‘0’ or a stored ‘1’. As an example, the memristance combinations $M_{UP}/M_{DOWN} = R_{ON}/R_{OFF}$ or R_{OFF}/R_{ON} are used to represent the aforementioned binary values. Figure 2.9b illustrates the *i*–*v* response of a simulated CRS, which comprises a forward-polarized memristor (FPM) and a reversely polarized memristor (RPM), after programming the individual devices into the state FPM/RPM = R_{OFF}/R_{ON} prior to further processing. Single memristors with opposite polarities demonstrate reversed behavior to the applied signals. During a single period of the applied AC voltage, the complementary devices change their states in a reciprocal way, delivering this perfectly symmetric composite *i*–*v* curve.

Assuming the given initialization, a positive applied voltage creates the necessary conditions to first change the state of the FPM from R_{OFF} to R_{ON} and later that of the RPM from R_{ON} to R_{OFF}, resulting in a flipped resistive configuration. Next, the memristors exhibit an ohmic behavior until the voltage exceeds the respective negative thresholds and forces the memristors to successively switch to their initial states (first the RPM and then the FPM). In the resulting *i*–*v* characteristic, the current is linear with the voltage except in two finite voltage intervals.

In order to utilize a CRS as a memory cell, starting from Fig. 2.9b one has to select appropriate programming and reading voltages. The first must exceed the voltage limits where the state-transitions are completed, whereas the latter must lie within the region denoted by the red dotted lines, i.e. at a particular point where presence of high (low) current will determine reading a R_{OFF}/R_{ON} (R_{ON}/R_{OFF}) state. In our simulations we program the device using ±3 V pulses and read it using 1.9 V pulses. Figure 2.9c shows the series of applied voltage pulses and Fig. 2.9d includes the resulting measured currents, whereas in between the aforementioned graphs there are the state transitions happening each time a voltage pulse is applied. Whenever the less resistive combination occurs, i.e. the R_{ON}/R_{ON}, we observe an instant current peak which is characteristic of this transition.

Overall, the simulation results confirm the successful reproduction of the CRS operation with the presented graphs qualitatively matching the experimental results reported in [30]. Therefore we proved that the presented model can be readily used to simulate complex memristor-based circuits/networks.

2.5 Overview and Comparison

Using SPICE is common practice in all device level simulations and helps in the development of new circuit architectures applicable to novel emerging technologies. A variety of SPICE memristor models have been presented over the last few years and performing a fair comparison between them is definitely a rigorous task, given that the original papers often omit fundamental details and adjustments of secondary parameters.

Here we briefly comment on the most noteworthy, in our opinion, models of the literature, and we define a set of metrics while trying to characterize them. These metrics include: (i) the consideration of programming thresholds; (ii) the low complexity of the equations of the model; and (iii) the support for high working frequencies of the applied signals. The aforementioned selection is based on the fact that: (i) threshold-type switching is a common feature of experimental memristive devices; (ii) the desired memristor model should involve the lowest possible complexity capable of delivering efficient performance; and (iii) a versatile model should support a wide set of working frequencies to make possible the simulation of novel fabricated devices which present very fast switching times [31]. Table 2.2 presents

Table 2.2 Memristor SPICE model comparison

SPICE models	Programming thresholds	Support for high frequencies	Low complexity
[2]	■	■	
[5]	■	■	
[6]			■
[7]	■	■	
[8]	■	■	
[14]			■
[15]			■
[16]			■
[18]			■
[19]	■		
This work	■	■	■

■ Compliance with the property of the corresponding column

the summary of the comparison based on both our experience and on the material presented in the corresponding original works. From the ten selected SPICE models, only half of them permit threshold programming [2, 5, 7, 8, 19] whereas only four are capable of working under application of high-frequency signals [2, 5, 7, 8]. Possibly, these four could be the most accurate SPICE memristor models currently available, which explains the large number of equations and parameters that they include; hence none of these models involves low-complexity operation.

The presented generalized and versatile SPICE-compatible memristor model efficiently complies with all of the aforementioned metrics. It allows for multiple-threshold programming and has an unlimited set of working frequencies, whereas it is based on simple equations which guarantee low-complexity operation. In SPICE it is represented by a two-terminal sub-circuit, whose parameters can be setup at sub-circuit instantiation for each device. It can be easily parameterized to adequately match several types of memristive behavior and can be readily included in existing electronic circuit designs. All the simulation results presented throughout the rest of this book will be based on this model.

References

1. D.B. Strukov, G.S. Snider, D.R. Stewart, R.S. Williams, The missing memristor found. Nature **453**(May), 80–83 (2008)
2. K. Eshraghian, O. Kavehei, K.R. Cho, J.M. Chappell, A. Iqbal, S.F. Al-Sarawi, D. Abbott, Memristive device fundamentals and modeling: applications to circuits and systems simulation. IEEE Proc. **100**(6), 1991–2007 (2012)
3. T. Prodromakis, B.P. Peh, C. Papavassiliou, C. Toumazou, A versatile memristor model with nonlinear dopant kinetics. IEEE Trans. Electron Dev. **58**(9), 3099–3105 (2011)

4. M. Di Ventra, Y.V. Pershin, L.O. Chua, Circuit elements with memory: memristors, memcapacitors and meminductors. IEEE Proc. **97**(10), 1717–1724 (2009)
5. E. Lehtonen, M. Laiho, CNN using memristors for neighborhood connections, in *12th International Workshop Cellular Nanoscale Network Applications (CNNA)*, Berkeley, CA (2010)
6. S. Shin, K. Kim, S. Kang, Compact models for memristors based on charge-flux constitutive relationships, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(4), 590–598 (2010)
7. C. Yakopcic, T.M. Taha, G. Subramanyam, R.E. Pino, S. Rogers, A memristor device model. IEEE El. Dev. Lett. **32**(10), 1436–1438 (2011)
8. S. Kvatinsky, E.G. Friedman, A. Kolodny, U.C. Weiser, TEAM: threshold adaptive memristor model. IEEE Trans. Circ. Syst. I Reg. Papers. **60**(1), 211–221 (2013)
9. M.D. Pickett, D.B. Strukov, J.L. Borghetti, J.J. Yang, G.S. Snider, D.R. Stewart, R.S. Williams, Switching dynamics in titanium dioxide memristive devices. J. Appl. Phys. **106**, 074508 (2009)
10. Y.N. Joklekar, S.J. Wolf, The elusive memristor: properties of basic electrical circuits. Eur. J. Phys. **30**, 661–675 (2009)
11. D.B. Strukov, J.L. Borghetti, R.S. Williams, Coupled ionic and electronic transport model of thin-film semiconductor memristive behavior. Small **5**(9), 1058–1063 (2009)
12. D.B. Strukov, R.S. Williams, Exponential ionic drift: fast switching and low volatility of thin-film memristors. Appl. Phys. A Mater. Sci. Process. **94**(3), 515–519 (2009)
13. R.S. Williams, M.D. Pickett, J.P. Strachan, Physics-based memristor models, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Beijing (2013)
14. Z. Biolek, D. Biolek, V. Biolkova, SPICE model of memristor with nonlinear dopant drift. Radioengineering **18**(2), 210–214 (2009)
15. D. Batas, H. Fiedler, A memristor SPICE implementation and a new approach for magnetic flux controlled memristor modeling. IEEE Trans. Nanotechnol. **10**(2), 250–255 (2011)
16. A. Rak, G. Cserey, Macromodeling of the memristor in spice. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(4), 632–636 (2010)
17. S. Benderli, T. Wey, On SPICE macromodelling of TiO_2 memristors. Electron. Lett. **45**(7), 377–379 (2009)
18. M. Mahvash, A.C. Parker, A memristor SPICE model for designing memristor circuits, in *53rd IEEE International Midwest Symposium on Circuits Systems (MWSCAS)*, Seattle, WA (2010)
19. H. Abdalla, M.D. Pickett, SPICE modeling of memristors, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Rio de Janeiro (2011)
20. M.J. Sharifi, Y.M. Banadaki, General SPICE models for memristor and application to circuit simulation of memristor-based synapses and memory cells. J. Circuit Syst. Comp. **19**(2), 407–424 (2010)
21. A. Ascoli, F. Corinto, V. Senger, R. Tetzlaff, Memristor model comparison. IEEE Circuits Syst. Mag. **13**(2), 89–105 (2013)
22. Y.V. Pershin, M. Di Ventra, Memory effects in complex materials and nanoscale systems. Adv. Phys. **60**(2), 145–227 (2011)
23. I. Vourkas, G.C. Sirakoulis, A novel design and modeling paradigm for memristor-based crossbar circuits. IEEE Trans. Nanotechnol. **11**(6), 1151–1159 (2012)
24. L.I. Schiff, Quantum mechanics, in *International Series in Pure and Application Physics*, 3rd ed. (McGraw-Hill, New York, 1968), pp. 100–104
25. S.P. Adhikari, M.P. Sah, H. Kim, L.O. Chua, Three fingerprints of memristor. IEEE Trans. Circuits Syst. I Reg. Papers **60**(11), 3008–3021 (2013)
26. I. Vourkas, A. Batsos, G.Ch. Sirakoulis, SPICE modeling of nonlinear memristive behavior. Int. J. Circ. Theor. Appl. **43**(5), 553–565 (2015)
27. H.-S.P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F.T. Chen, M.-J. Tsai, Metal-oxide RRAM. IEEE Proc. **100**(6), 1951–1970 (2012)
28. L.O. Chua, Resistance switching memories are memristors. Appl. Phys. A Mater. Sci. Process. **102**(4), 765–783 (2011)

29. K. Eshraghian, K.-R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, S.-M. Kang, Memristor MOS content addressable memory (MCAM): hybrid architecture for future high performance search engines. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **19**(8), 1407–1416 (2011)
30. E. Linn, R. Rosezin, C. Kugeler, R. Waser, Complementary resistive switches for passive nanocrossbar memories. Nat. Mater. **9**(5), 403–406 (2010)
31. A.C. Torrezan, J.P. Strachan, G. Medeiros-Ribeiro, R.S. Williams, Sub-nanosecond switching of a tantalum oxide memristor. Nanotechnology **22**(48), 485203 (2011)

Chapter 3
Dynamic Response of Multiple Interconnected Memristors

3.1 Introduction

Currently there is a growing variety of systems that exhibit memristive behavior, as academia and industry keep on with their research and prototyping [1]. While non-volatile resistive memory is the most prominent application of the resistance switching phenomenon of memristors [2, 3], several other emergent applications have also been presented in the literature, including reconfigurable memristive computational circuits and architectures [4–7]. Memristor-based circuits open new pathways for the exploration of advanced computing paradigms and architectures where the dynamical behavior of a memristor's resistance is primarily used for computation. The major distinction from the present day's computing technology lies in that, memory and processing units are not physically separated; in fact, computational results are maintained in the states of the computing devices, which are memristors.

However, while most of the research has so far focused on the properties of memristive devices, composite memristive behavior has not yet been fully explored and no high-level approaches to composite memristive systems have been published; so little is still known about the extraordinary features of the composite response and the application prospects of multiple interconnected memristors. Depending on the circuit topology, the device orientation (polarity), their current memristance (state), and their internal switching properties, which in simulation are reflected in the values of the parameters of the used model, their overall behavior turns up complicated and is difficult to predict. It has been shown that appropriately interconnected memristors significantly improve the efficiency of computations via massive parallelism, where computation consists in the concurrent state-evolution of all the involved devices (something difficult to realize within the conventional circuit design domain) [8]. Some interesting statistical properties, regarding regular

I. Vourkas and G.Ch. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Emergence, Complexity and Computation 19,
DOI 10.1007/978-3-319-22647-7_3

circuit configurations of multiple connected memristive devices, were reported in [9], whereas the composite switching characteristics and the relationships among individual memristors were studied in [10, 11]. Large groups of interconnected memristors have been primarily studied in the crossbar geometry [12].

In this chapter we focus on the architectural perspectives that arise in circuits with multiple interconnected memristors, which demonstrate threshold-dependent switching behavior [13]. The latter is in line with the fact that, in the majority of experimental realizations, the change-rate of the memristance essentially depends on whether the applied voltage exceeds some specific thresholds [1, 14]. We investigate the dynamic switching response and analyze the characteristics of both regular and irregular serial/parallel memristive circuit compositions; i.e. memristive combinations which are structured using either repetitive or non-repetitive interconnection patterns [15]. We show how composite memristive systems can be efficiently built out of individual memristors, presenting different electrical characteristics from their structural elements. Following the proposed generalized synthesis concept, by appropriately selecting and interconnecting the constitutive circuit components, we construct composite memristive systems which exhibit behavior of programmable multi-state conducting elements [16]; their current state is dependent on the previously applied voltage pulsing protocol. We provide several examples of such memristive implementations, combining different polarities and different initial states and/or switching characteristics, thus causing highly non-trivial, composite responses to the applied voltages.

Finally, we present a novel approach for the construction of robust fine-resolution programmable memristive switches. Programmable resistors are nowadays required for various circuit applications (e.g. amplifiers, filters, etc.) mostly in order to facilitate adaptation to particular conditions [17, 18]. Although memristors are indeed programmable variable resistors which could be arbitrarily programmed to theoretically all intermediate conducting states, a given memristance precision requires biasing with very precise amplitude and duration, thus is strongly dependent on device variations [19]. To this end, the presented methodology makes unnecessary the need for high precision tuning. Application examples use the proposed multi-state memristive switches in a closed loop computing circuit, able to generate scalable output voltage levels in a step-wise manner.

Overall, the presented analysis provides intuition into the composite and dynamic complexity of both regular and irregular memristive circuit topologies. It is also expected to assist both in the interpretation of their response, as well as in the creation of large sophisticated memristive systems and architectures. Based on the memristor device model presented previously in Chap. 2 [20, 21], we provide a SPICE simulation-based evaluation of all the composite memristive system examples. Their stable operation and rich, threshold-dependent, nonlinear, switching behavior could be utilized in various applications, including analog computing devices [17], programmable filters [22], passive sensing systems [23], signal/pattern recognition [24], etc.

3.2 Study of Composite Memristive Structures

Memristive devices are essentially non-volatile switches commonly found with a set of voltage thresholds. This means that there is negligible change of the internal resistive state if the magnitude of the applied voltage remains small. However, if the voltage exceeds either of the SET or RESET effective thresholds for an interval longer than the switching time of a device, then the device undergoes a rapid and large resistance change. In this section we analyze the composite behavior of circuits comprising memristors connected in a serial or parallel manner. Being associated with the totally nonlinear behavior of individual memristive elements, circuits with multiple memristors may work in very complicated way due to the polarity-dependent, nonlinear memristance-variation of single memristors. We explore the dynamics of several circuits considering series or parallel connection, same or opposite polarities, as well as different state-initializations. Hereinafter, we will refer to forward (reversely) polarized memristors as FPMs (RPMs). Regarding the memristor circuit schematic, for a FPM the top terminal is the one with the thin line whereas for a RPM it is the one with the thick line.

All circuit simulations are based on the memristor device model of Chap. 2. Figure 3.1 illustrates the simulated memristor response to an AC triangular voltage. Model parameter values are used as given in $\{\alpha_x, b, c, m, f_o, L_o, V_{SET}, V_{RESET}\}$ = $\{5 \times 10^3, 0, 0.1, 82, 310, 5, 1\text{ V}, -1\text{ V}\}$ and the resulting resistance ratio is $R_{OFF}/R_{ON} \approx 10^2$ with $R_{OFF} \approx 200$ kΩ and $R_{ON} \approx 2$ kΩ. Such value-set corresponds to a memristor which switches steeply as soon as the applied voltage exceeds either of its thresholds; otherwise its state is unaffected ($b = 0$). Memristance ratio is set to two orders of magnitude to facilitate the better distinction between the two boundary resistive states. With $a_x > 0$ the memristance of a FPM will decrease (increase) when it is forward (reversely) biased, whereas a RPM has the opposite behavior. It is worth mentioning that all assumptions regarding the thresholds, the programming voltages, as well as the memristance ratio, have been made only in the context of this study; thus, they do not relate to any real, manufactured and/or characterized device. Throughout this chapter, the characteristics of Fig. 3.1 will serve as a reference while studying the composite behavior of multiple interconnected devices under similar applied voltages of the same frequency.

3.2.1 *Memristors Connected in Series*

Considering a single memristive device as a structural element, in this section we analyze the composite voltage-dependent switching behavior of circuit branches with more than one device connected in series. In the series connection the elements form a voltage divider. Therefore, depending on the amplitude of the applied voltage, the voltage drop on each memristor during simulation may exceed its threshold and cause a state-drift, or it may be below it and, thus, leave the device

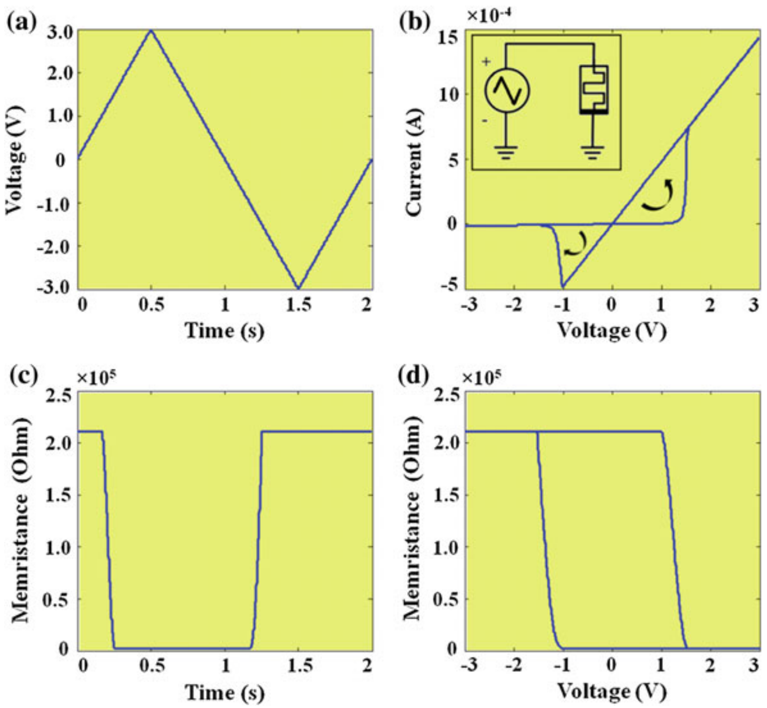


Fig. 3.1 Simulation result for the response of a memristor to a triangular AC applied voltage. **a** The applied voltage; **b** the hysteretic current-voltage (*i*-*v*) characteristic where the *inset* shows the corresponding circuit schematic; **c**, **d** the change of the memristance with time and with the applied voltage, respectively

unaffected (existence of noise in the supply and/or device mismatch are not taken into consideration).

3.2.1.1 Same Polarities

Assuming the same polarity for all memristors, let us focus first on the smallest configuration which consists of only two devices. Figure 3.2 shows the simulation results for a pair of memristors subject to AC triangular applied voltage. Three possible combinations for the initial states are examined: OFF/OFF, OFF/ON, and ON/ON, where the “upper/lower” notation is used in the text to reflect the corresponding placement of the two memristors.

In Fig. 3.2a the two FPMs are initialized in R_{OFF}. During simulation the voltage drop on each memristor is the same because the devices are identical; hence they both switch to R_{ON} together when the voltage across each of them exceeds its SET threshold. Specifically, both devices are toggled from R_{OFF} to R_{ON} (and vice versa) when the total applied voltage exceeds |2 V|. Therefore, the threshold of the

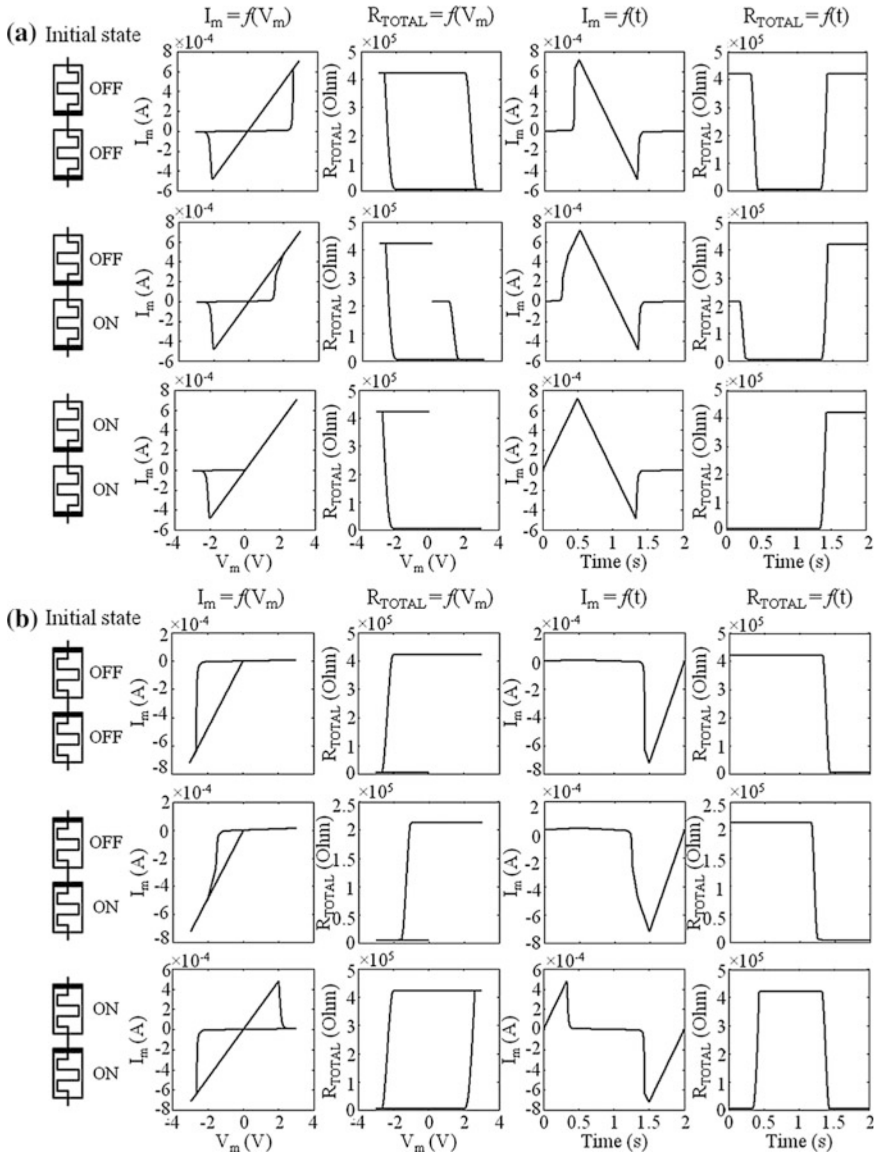


Fig. 3.2 Simulation results for two memristors of the same polarity connected in series when being both **a** FPMs or **b** RPMs. I_m and V_m denote current and applied voltage, respectively, whereas total memristance (R_{TOTAL}) is the sum of the individual memristances

composite device is the sum of the individual thresholds, i.e. 2 × {V_{RESET}, V_{SET}}, and the composite memristance ranges within 2 × [R_{ON}, R_{OFF}]; compared to Fig. 3.1, here the maximum current is smaller because of the twofold less-resistive composite state (2 × R_{ON}).

In the case of being initialized as OFF/ON, during the positive half of the voltage sweep the upper device changes its state. However, the state-change this time occurs much sooner and at a lower voltage. This is due to the high ratio of the boundary memristance values ($R_{\mathrm{OFF}} \gg R_{\mathrm{ON}}$) which, as a consequence, causes almost the entire applied voltage to drop on the memristor which is in R_{OFF}. On the contrary, the state of the lower device cannot be further changed since it is already found in the ON state. During the negative part of the voltage sweep, both devices change from ON to OFF simultaneously. Finally, when initialized as ON/ON, both devices are unaffected by a positive applied voltage (the *i*-*v* graph resembles that of a typical resistor of resistance equal to $2 \times R_{\mathrm{ON}}$), but normally change to the OFF/OFF combination with a negative applied voltage of appropriate magnitude.

Figure 3.2b presents the simulation results when the devices are both RPMs. In the OFF/OFF initialization case, both devices are unaffected during the positive half of the voltage sweep because they are already found in the boundary R_{OFF} memristance, so a very small current is observed. Only when the applied voltage exceeds −2 V then both elements switch states together. This behavior is similar to that of the FPMs in the last case of Fig. 3.2a, only that here the initially observed current in the *i*-*v* plot is smaller. In the OFF/ON initialization case, during the positive half of the voltage sweep the lower device is reversely biased, thus it is expected to change state from ON to OFF. However, owing to the high memristance ratio, the corresponding voltage drop on this device never exceeds its RESET threshold; thus the total memristance remains equal to $R_{\mathrm{OFF}} + R_{\mathrm{ON}} \approx R_{\mathrm{OFF}}$. Of course, a higher applied voltage would have successfully switched this memristor. The last case, where the devices are initially in the ON/ON combination, evidently is the opposite of the first example of Fig. 3.2a; the overall behavior is that of a composite and reversely polarized memristor. Inspecting the simulation results of Fig. 3.2 reveals the following: when employing devices with identical properties (i.e. equal memristance ratios, switching rates, voltage thresholds, etc.), if they are all placed with the same polarity (i.e. either all FPMs or all RPMs), then their overall behavior resembles that of a single memristor of the same polarity. However, the composite switching properties combine the properties of the individual memristors.

3.2.1.2 Opposite Polarities

In Fig. 3.2 we observed that memristors with opposite polarities present a flipped *i*-*v* characteristic and generally demonstrate reversed behavior to the applied signals; e.g. a positive voltage tends to switch a FPM (RPM) from OFF to ON (from ON to OFF). Opposite polarities along with different initial states may cause highly non-trivial composite responses to the applied voltages. In Fig. 3.3 we show the simulation results for two memristors with opposite polarities connected in series, studied for a triangular AC applied voltage. We inspect the composite behavior using the same applied voltage and assuming the same initial state combinations for the individual elements, as in Fig. 3.2.

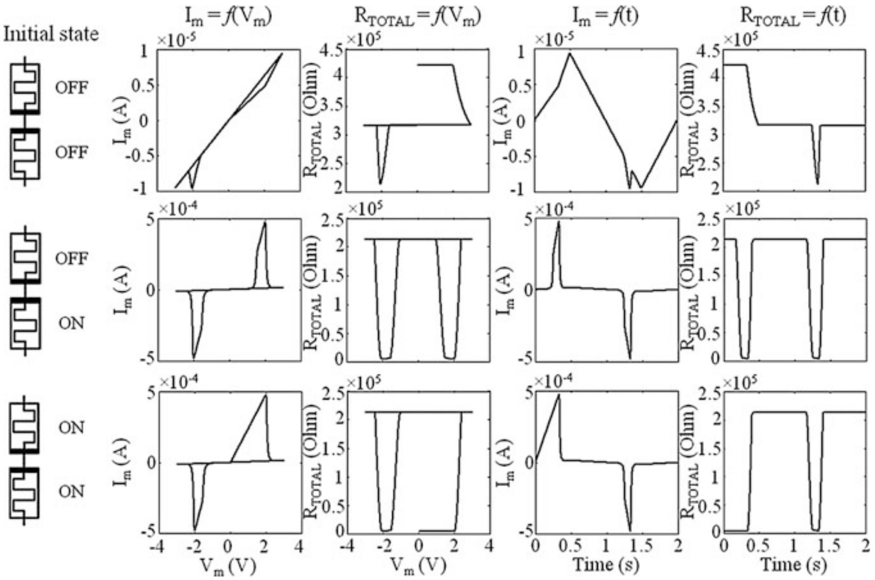


Fig. 3.3 Simulation results for two memristors connected in series with opposite polarities. I_m and V_m denote current and applied voltage, respectively. Total memristance (R_{TOTAL}) is the sum of the individual memristances

If the devices are initialized as OFF/OFF, the positive applied voltage tends to switch ON the upper device, whereas the lower device will remain in the OFF state. During the switching procedure, though, the lower the memristance of the upper device, the smaller the corresponding voltage drop which quickly falls below the SET threshold; thus the state-change is never complete. Next, the negative voltage sweep tends to change the lower device from OFF to ON and the upper device from its last intermediate state (due to the incomplete SET procedure) back to the OFF state. The observed spike-like change of the total memristance (R_{TOTAL}) is explained as follows: the bottom device starts the switching first thanks to its high resistance, which draws most of the applied voltage. Nevertheless, as the resistance of the bottom element is reduced, the value of the voltage drop on its terminals is reduced as well. Therefore, at some point the voltage drop on the top device exceeds its threshold and initiates the RESET switching process. Thereupon, the top device draws almost the entire applied voltage due to its increasing resistance, so the voltage drop on the bottom device never exceeds its threshold; the total memristance remains unaffected until the end of the voltage sweep.

Regarding the OFF/ON initialization case, the complementary devices change their states in a reciprocal way delivering a perfectly symmetric composite *i-v* curve. A positive applied voltage creates the necessary conditions to first change the state of the FPM from OFF to ON and later that of the RPM from ON to OFF, resulting in a flipped resistive configuration. Next, the memristors exhibit an ohmic behavior until the voltage exceeds the respective negative thresholds and forces the memristors to successively switch to their initial states (first the RPM and then the FPM). This is the concept of the complementary resistive switch (CRS); the current is linear with the voltage except in two finite voltage intervals [25]. Generally, in such configurations we observe two different memristance levels, whose exact value can be modified by including more devices in series (this property is discussed later in this section). Finally, in the last case the devices are both initialized in the low memristance state (ON/ON). So, in the positive part of the voltage sweep only the state of the bottom device is changed, whereas during the rest of the AC sweep the overall composite behavior is the same with that of the previously studied case; when the voltage sweep is complete, the memristors are found in the OFF/ON combination.

Up to this point we have thoroughly examined the dynamic behavior of circuit branches comprising at most two devices in serial or anti-serial configurations. However, the collective time-evolution of a large number of memristors coupled together, possessing memory of their past dynamics, proves quite intriguing and it is very difficult to capture. Even for the same initial memristance, the final composite state may differ depending on the applied signal because of the collective effect of all the interconnected components. In order to emphasize such complex collective dynamics, Fig. 3.4 presents the switching characteristics of ten memristors connected in series, subject to either a ramp (or saw-tooth) waveform voltage slowly increasing from 0 V to $v_0 = 10$ V (left column) or a step-like applied voltage of $v_0 = 10$ V (right column). We monitor the memristance of the devices under the aforementioned biasing schemes. In all simulations, the values of the parameters of the model were common for all the involved devices, whereas different initial resistances were applied. The maximum magnitude of the applied voltage was purposely selected to be $v_0 = 10 \times V_{\mathrm{SET}}$.

In the first simulation scenario all memristors are FPMs initialized at the highest resistive state (R_{OFF}). During simulation, the corresponding voltage drop on each device is the same because they are all identical; so all the devices would switch ON together when the voltage at their terminals exceeds their SET threshold (V_{SET}). However, as shown in Fig. 3.4a, b, all the devices fail to switch because the corresponding voltage drop at most reaches the voltage threshold but never exceeds it ($V \leq V_{\mathrm{SET}}$).

In the second scenario the forward polarity of the memristors was kept but the individual devices were now arbitrarily initialized, according to the uniform distribution, to any possible intermediate memristance within the range [R_{ON}, R_{OFF}]. Figure 3.4c, d show that in both cases the memristive ensemble practically has the same initial and final memristance. This can be attributed to a "domino" switching effect; i.e. the situation where a single rapid switching event induces switching in other memristive devices. As it was explained previously, as soon as the voltage drop across a particular element exceeds its SET threshold, its memristance is sharply reduced and at the same time the instantaneous voltage drop across it is also

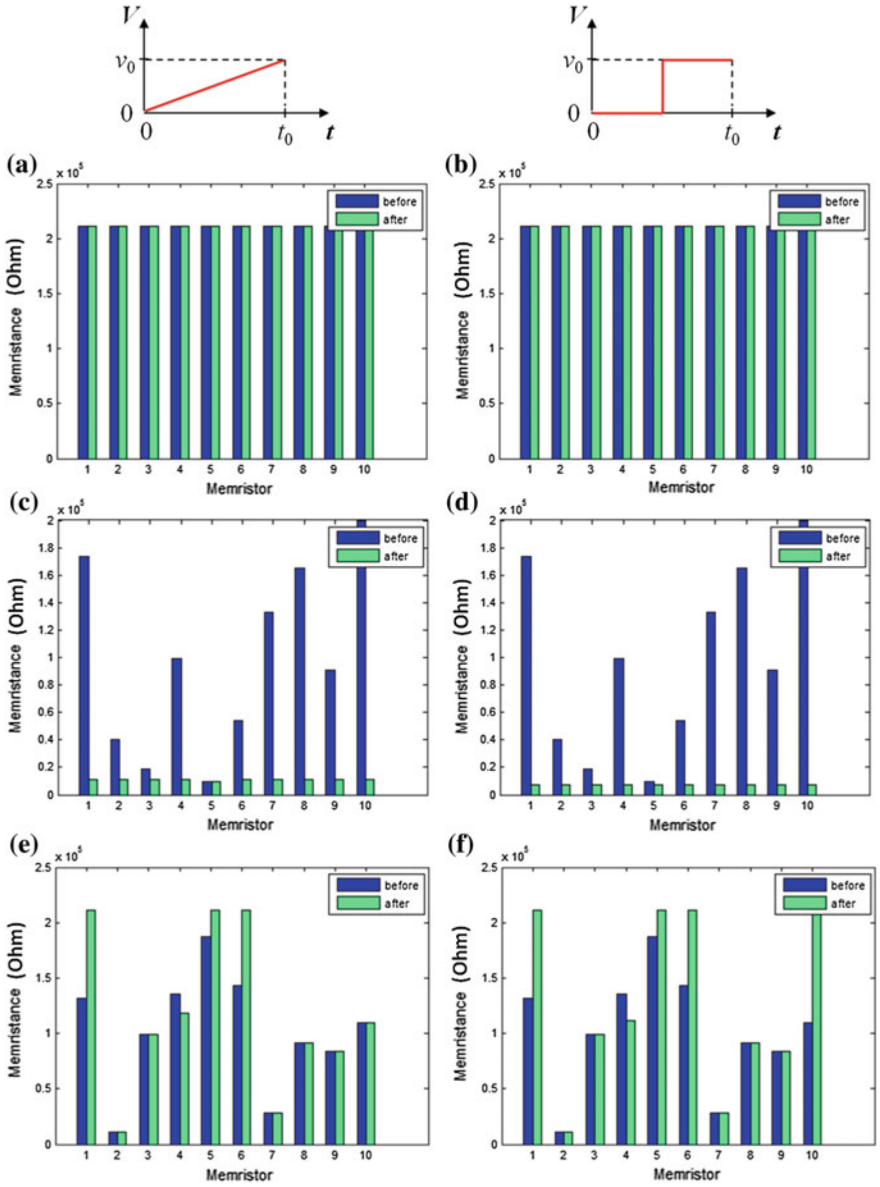


Fig. 3.4 Initial and final memristance for ten memristors connected in series, subject to either a slowly increasing ramp waveform voltage from 0 V to v_0 = 10 V (*left* column) or a step-like voltage of v_0 = 10 V applied at $t = t_0/2$ (*right* column). The simulation scenarios involve: **a**, **b** all devices as FPMs initialized in the high resistive state; **c**, **d** all devices as FPMs arbitrarily initialized according to the uniform distribution within the interval [R_{ON}, R_{OFF}]; **e**, **f** both the polarity and the initial memristance of all memristors being assigned randomly

decreased due to the voltage divider circuit properties. As a consequence, the voltage drop on other devices is simultaneously increased proportionally to their memristance and finally causes them to switch when their SET threshold is surpassed. Such behavior is, however, facilitated by the actual polarity of the circuit elements. For example, an accelerated state-switching towards R_{OFF} cannot induce a domino effect in memristive ensembles which comprise devices with opposite polarities. Indeed, such effect is suppressed since the higher the memristance of a particular element gets, the more voltage it draws across it, thus it reduces the voltage drop on the rest of the elements in the circuit.

Figure 3.4e, f enable the comparison of the final states of the memristors in the circuit when both their initial resistances and their polarity are randomly assigned. The charts indicate successful switching of memristors which initially had the higher memristances (the devices where the voltage drop exceeded the voltage threshold). However, while the switching of RPMs towards R_{OFF} is always complete, FPMs may only partially switch towards R_{ON} due to the aforementioned suppressive effect caused by RPMs. In this particular situation, the main difference between the observed final states is associated with the memristor No. 10. Specifically, only the suddenly applied voltage pulse achieved to switch OFF this element, whereas the slowly increasing ramp waveform voltage left it unaffected.

In the examples of Fig. 3.4, the intentional selection of the magnitude of the applied voltage to be at most equal to the accumulated switching threshold of all the devices in series, served to emphasize the dependence of their final states on the biasing scheme. Nevertheless, regardless of the initial state of the devices, a sufficiently high input pulse (either positive or negative) would have eventually forced all memristors to switch to a boundary resistive state depending on their polarity. Therefore, it makes sense to investigate the effect of the polarity on the overall time-response to AC periodic inputs.

Figure 3.5 shows the simulation results for ten memristors connected in series, while considering different polarities and device-specific properties. The circuit is subject to consecutive AC voltage periods of high enough amplitude, able to cause several switching events. More specifically, in Fig. 3.5a half of the devices are FPMs set in the OFF state, and half of them are RPMs found in the ON state. The response of such circuit resembles that of the typical CRS shown in Fig. 3.3, with the only difference that the two high-conduction intervals have moved horizontally in the v-axis; here the state-change starts when the applied voltage exceeds the accumulated threshold of the devices with the same polarity. However, the overall composite memristance ratio remains the same due to the symmetrical distribution between FPMs and RPMs.

Moreover, in Fig. 3.5b, c we have the same total number of devices but this time we have included (b) less FPMs and more RPMs or (c) less RPMs and more FPMs, respectively. Inspecting the simulation results reveals that, when the RPMs outnumber the FPMs (or when then FPMs outnumber the RPMs), we can selectively widen and shorten the specific high-conduction current lobes and thus dominate

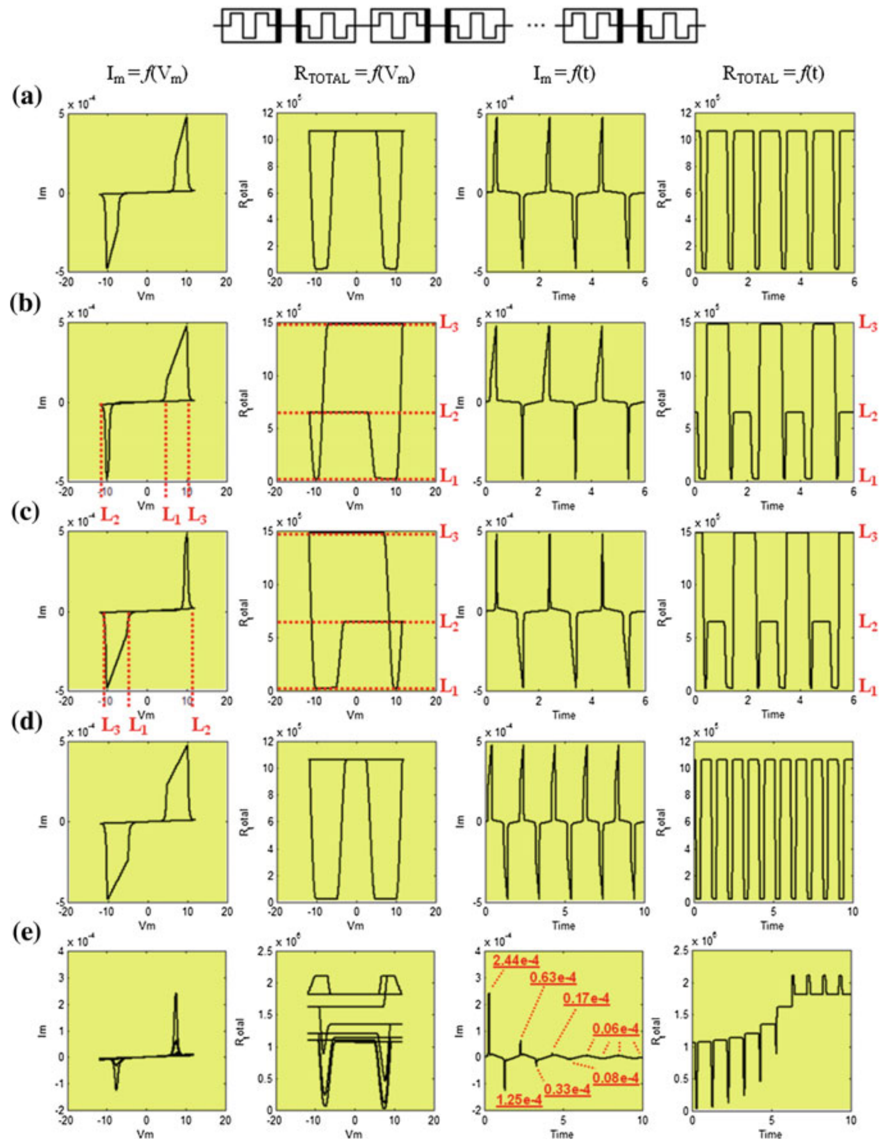


Fig. 3.5 Simulation results for ten memristors connected in series subject to consecutive triangular AC voltage periods. FPMs and RPMs are initially set to R_{OFF} and R_{ON}, respectively. Presented examples involve: **a** half FPMs and half RPMs, **b** four FPMs and six RPMs or **c** six FPMs and four RPMs, during three voltage periods when all devices have symmetric thresholds. The rest of the examples refer to half FPMs and half RPMs with: **d** $\{V_{\mathrm{RESET}}, V_{\mathrm{SET}}\} = \{-1, 0.5\}$ V or **e** $\{V_{\mathrm{RESET}}, V_{\mathrm{SET}}\} = \{-0.5, 1\}$ V, during five voltage periods. *Dashed* (*red*) *lines* in (**b**, **c**) indicate three different memristance levels and their corresponding voltage thresholds $\mathrm{L}_{1\text{–}3}$. In all graphs the units for $\{I_{\mathrm{m}}, V_{\mathrm{m}}, R_{\mathrm{TOTAL}}, t\}$ are {Ampere, Volt, Ohm, sec}

their duration while we also selectively create an additional composite memristance level. Using more RPMs (FPMs) results in a wider current pulse during the positive (negative) part of the sweep and a shorter current pulse during the negative (positive) part. Furthermore, in Fig. 3.5b, c the first defined resistive level L_1 corresponds to having all memristors set in R_{ON} [i.e. R_{L1} = (FPMs + RPMs) × R_{ON}]. Similarly, L_2 corresponds to the case when the less-populated devices are set in R_{OFF} and the most-populated devices are set in R_{ON}, respectively [i.e. assuming $R_{OFF} \gg R_{ON}$ in (b) it is $R_{L2} \approx$ FPMs × R_{OFF}]. Finally, when the less-populated devices are set in R_{ON} and the most-populated devices are set in R_{OFF}, the highest resistive level L_3 is achieved [e.g. in (b) it is $R_{L3} \approx$ RPMs × R_{OFF}]. The level-sequence during simulation in Fig. 3.5b is: $L_2 \rightarrow L_1 \rightarrow L_3 \rightarrow L_2 \rightarrow L_1 \rightarrow \cdots$ etc. Likewise, including more FPMs than RPMs in Fig. 3.5c only affects the corresponding voltage thresholds which define the aforementioned memristance transitions. It is worth noticing that, during three consecutive periods of the applied voltage, the total resistance always returns to its initial value after the completion of each of the voltage sweeps; hence such device combinations notably guarantee stable function.

Furthermore, Fig. 3.5d, e concern the use of asymmetric voltage thresholds (i.e. $V_{SET} \neq V_{RESET}$) for the individual devices. Particularly, in Fig. 3.5d we reduce the V_{SET} threshold value from 1 to 0.5 V to finally have the following set for all the memristive components: $\{V_{RESET}, V_{SET}\} = \{-1, 0.5\}$V. After this modification we observe that, when having equal number of FPMs and RPMs, a smaller V_{SET} threshold produces wider high-conduction intervals for both the positive and the negative part of the voltage sweep. Hence, when $|V_{RESET}| > V_{SET}$, the higher the absolute difference between them, the higher the duration of the high-conducting intervals will be.

However, this does not apply to the case of having $|V_{RESET}| < V_{SET}$. As shown in Fig. 3.5e, adjusting the respective thresholds to the following set: $\{V_{RESET}, V_{SET}\} = \{-0.5, 1\}$ V, ruins the complementary resistive behavior of the circuit. We deliberately run the simulation for five consecutive periods of the applied voltage to note that the duration of the high-conducting lobes significantly shrinks and the maximum measured current decreases with time. In particular, the maximum current ranges between 244 and 6 μA, to finally become stable after three periods. Nevertheless, no high-conduction can be then distinguished since the final effective memristance ratio is now significantly reduced. This behavior is attributed to the "accelerated switching" phenomenon discussed before. During simulation, regarding the memristors which tend to be switched OFF, the closer they get to the OFF state, the greater the corresponding voltage drop on them becomes; hence the remaining voltage over the rest of the devices which should flip from OFF to ON is never sufficient (or never applied for sufficient time) and, thus, their switching process in never complete. This causes the minimum less resistive state to increase with time.

In the same context, we have examined the stability of the complementary switching operation in circuits comprising ten memristors with equal number of

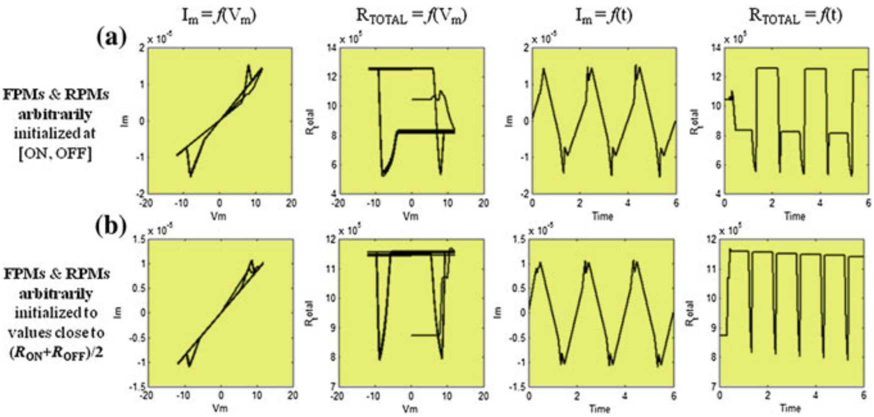


Fig. 3.6 Simulation results for the composite response of a series of ten memristors (half FPMs and half RPMs) subject to three consecutive triangular AC voltage periods. The memristors are arbitrarily initialized to any possible intermediate memristance within: **a** the full interval [R_{ON}, R_{OFF}], or **b** a much shorter value range around the mean value 1/2 × (R_{ON} + R_{OFF}). In all graphs the units for {I_m, V_m, R_{TOTAL}, t} are {Ampere, Volt, Ohm, sec}

FPMs and RPMs, when the memristance of the devices was arbitrarily initialized. In Fig. 3.6a the memristors were set in any possible intermediate memristance within the range [R_{ON}, R_{OFF}]. The exact value of the total memristance after every voltage sweep cannot be foreseen and may take any intermediate value between $n \times R_{ON}$ and $(n/2) \times R_{OFF}$, where n is the number of memristors (here $n = 10$). After conducting several simulations we observed that the resulting i–v always resulted very similar to this one and that the current graph hardly deviated from a prominent gradient. Also, one of the first things to note is that the circuit very quickly reached a stable operation, which resembles that of Fig. 3.5b, c, where one of the high-current lobes is larger than the other.

However, the same does not apply if we select to restrict the range of the initial memristance values around the mean value 1/2 × (R_{ON} + R_{OFF}). According to Fig. 3.6b, the circuit operates stably from the very beginning and the high-current lobes are better distinguished, with the overall function notably resembling that of a CRS shown in Fig. 3.5a, d. Hence, for such circuits it can be concluded that, except for the distribution of FPMs and RPMs, the initial resistive state of the devices plays a premiere role in the programming procedure in order to obtain a specific, desired, complementary resistive operation.

3.2.2 *Memristors Connected in Parallel*

In the same fashion, assuming a single memristive device as a structural element, we analyze the behavior of circuits with multiple memristors connected in parallel.

In the parallel connection the same voltage is applied to all memristors, so the switching thresholds which characterize the individual devices also characterize the composite behavior of the circuits. Unlike the series connection, here the R_{ON} values dominate the total resistance of the circuits.

3.2.2.1 Same Polarities

We first focus on the smallest configuration, which consists of two memristors, and study its composite response to a triangular AC applied voltage, while considering the same state-initialization cases as in the series connection. The simulation results for memristors with the same polarity are shown in Fig. 3.7.

Figure 3.7a concerns two FPMs. In the first case the memristors switch ON and OFF together, so the resulting *i*-*v* plot resembles that of the single memristor of Fig. 3.1. However, here the maximum current is twofold because the total memristance when both devices are ON is $R_{\mathrm{ON}}/2$ (i.e. $R_{\mathrm{ON}}\|R_{\mathrm{ON}}$). Therefore, connecting two identical memristors with the same polarity in parallel produces typical memristive behavior with half the initial boundary memristance range $1/2 \times [R_{\mathrm{ON}}, R_{\mathrm{OFF}}]$ and the same memristance ratio. The influence of having either of the two devices in the ON state can be observed in the second case of Fig. 3.7a; the total memristance is $\approx R_{\mathrm{ON}}$. During the positive half of the voltage sweep, the device already in the ON state is not further affected, whereas the other memristor changes its state as soon as the applied voltage exceeds its SET threshold. Their composite response during the negative part of the sweep is the same with that of the previous example, with both devices switching OFF together. In the last case of Fig. 3.7a, both devices are initially ON, so they are not affected by the positive voltage, whereas the negative voltage causes the same response as in previous examples.

In Fig. 3.7b we examine the composite response of a pair of RPMs. The simulation results demonstrate a behavior very similar to that of Fig. 3.7a. In fact, the first case here resembles the last case of Fig. 3.7a, only that now the positive voltage does not affect the devices that are already OFF, whereas the negative voltage forces both memristors to simultaneously switch ON. In the second case again only the memristor initialized in the OFF state switches ON by the positive applied voltage, whereas in the last case we observe a composite device which functions in the opposite way than the first case of the FPMs. Overall, the negative part of the voltage sweep causes always the same response regardless of the initial states of the memristors.

Having already noticed that identical memristors (or groups of memristors) with opposite polarities can deliver symmetric individual (composite) behavior, it is of great interest to explore their composite response when they are connected together. Therefore we next examine the total response of groups of two or more than two devices with anti-parallel configurations.

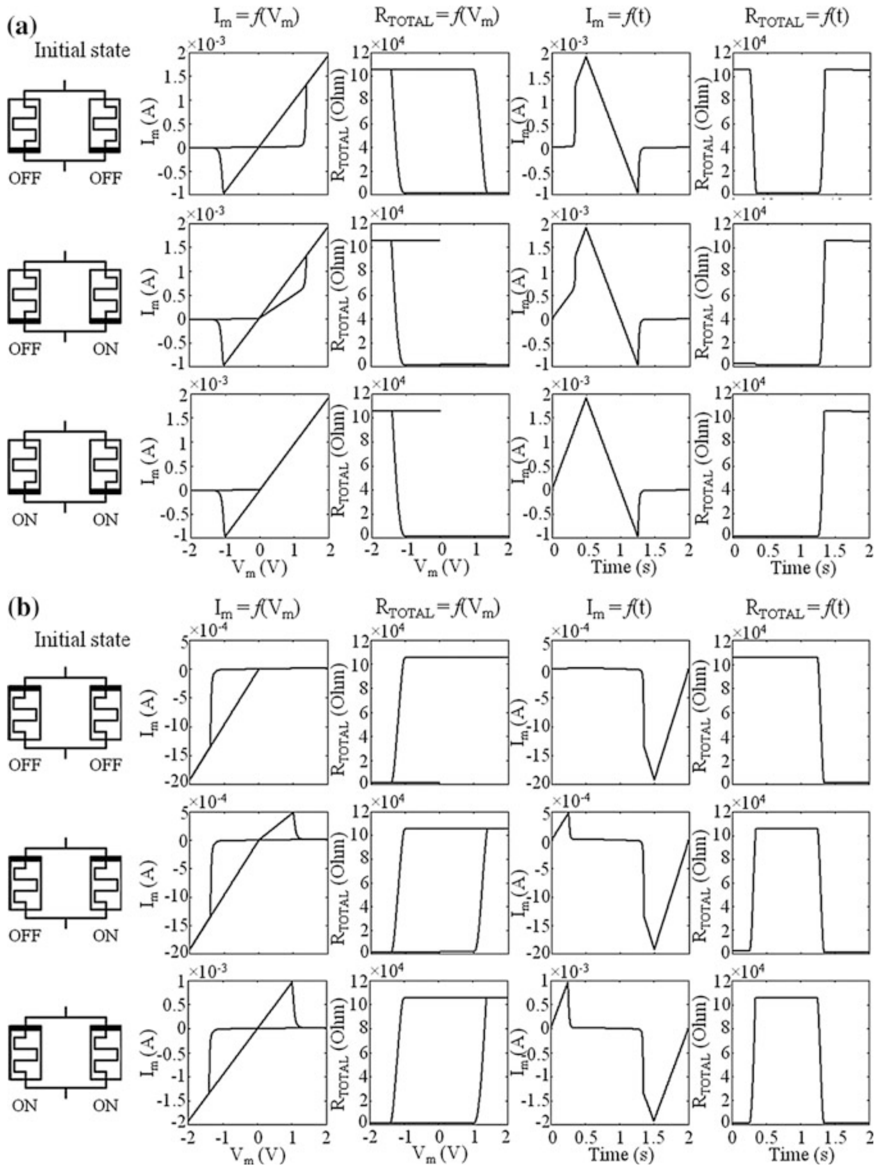


Fig. 3.7 Simulation results for two memristors connected in parallel with the same polarity when being both **a** FPMs or **b** RPMs

3.2.2.2 Opposite Polarities

We start our analysis by studying the smallest anti-parallel memristive configuration which comprises two devices. Observing Fig. 3.8 reveals that the overall

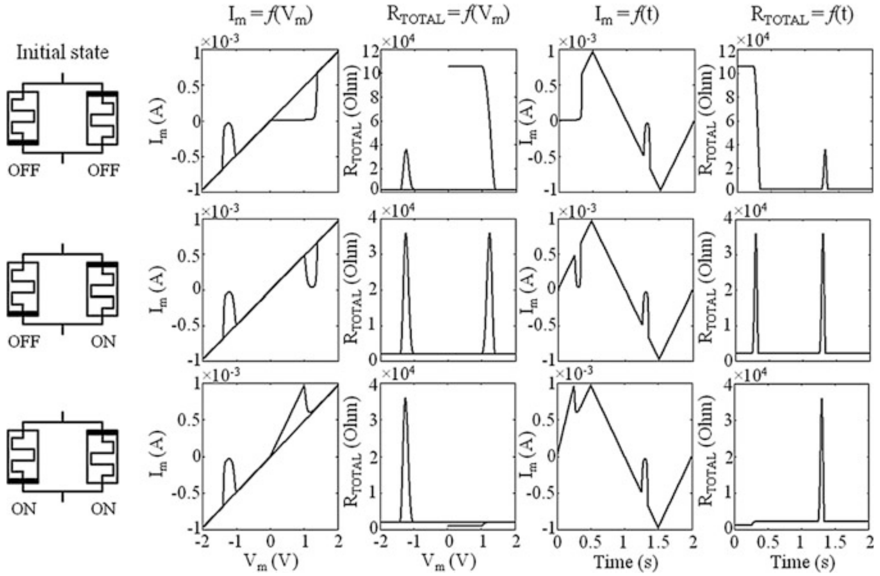


Fig. 3.8 Simulation results for two memristors connected in parallel with opposite polarities

composite memristance in all demonstrated cases is kept at very low values except for certain intervals, which are marked with spike-like transitions. This is because, as we have concluded before, devices with opposite polarities have opposite switching characteristics; each time a voltage is applied, one of the devices tends to switch to the OFF state and the other to the ON state. Hence, there is almost always a device in the ON state dominating the overall memristance.

Except for the first example where both devices are initially OFF, in simulation the OFF/OFF combination occurs only as an intermediate state during the alternate state-transitions; duration of the OFF/OFF combination depends on the voltage thresholds of the combined memristors. The most characteristic case in Fig. 3.8 involves a FPM initially in the OFF state and a RPM in the ON state, together making an anti-parallel resistive switch (ARS) [26]. Similar to the CRS concept, the resulting *i-v* characteristic again resembles a truncated Ohm’s Law, though ARS functions opposite to CRS. It is worth noting that after a complete voltage sweep, the anti-parallel pair of memristors behaves as ARS regardless of the memristance initialization.

Although only a pair of anti-parallel memristors was used to illustrate functionality of ARS, the same principles apply for more than two connected devices. The magnitude of the total current depends on the instant combination of all the memristances. Likewise in the series connection, here we studied the composite behavior of ten parallel memristors subject to consecutive AC voltage periods. In Fig. 3.9a we use equal numbers of FPMs and RPMs with identical symmetric thresholds: $\{V_{RESET}, V_{SET}\} = \{-1, 1\}$ V; FPMs (RPMs) are initially in the OFF

(ON) state. The simulation results are the same with the ARS case shown in Fig. 3.8, only that here the maximum flowing current is fivefold due to the sub-multiple minimum composite memristance. Therefore, introducing more devices to an anti-parallel configuration while maintaining equal distribution between FPMs and RPMs, produces the same function but with higher achieved currents.

Furthermore, in Fig. 3.9b, c we observe how additional composite memristance levels can be created by optimizing the distribution between FPMs and RPMs, likewise we did for the series connection. Here the different memristances are denoted with the multiple gradients of the current in the *i*-*v* plot. The finite voltage intervals, when all memristors are switched OFF, are maintained. However, the current memristance before and after crossing these intervals, is different. With such configurations we can make programmable composite switches providing three different conduction levels, e.g. high, medium, and (almost) no conductive, depending on the prior applied pulsing protocol. Compared with the corresponding implementation using anti-serially connected memristors, here the switching thresholds (thus the operating voltages) are smaller and clearly defined.

In Fig. 3.9d, e we investigated again the effect of having devices with asymmetric voltage thresholds while considering equal numbers of FPMs and RPMs. Particularly, for all memristors we have: $V_{\mathrm{RESET}} = -1.5$ V with $V_{\mathrm{SET}} = 0.5$ V in Fig. 3.9d and $V_{\mathrm{RESET}} = -0.5$ V with $V_{\mathrm{SET}} = 1.0$ V in Fig. 3.9e. Apparently, having $|V_{\mathrm{RESET}}| > V_{\mathrm{SET}}$ converts the function of ARS to that of CRS, though with much higher currents due to the smaller effective boundary memristance ratio. On the contrary, according to Fig. 3.9e, defining $|V_{\mathrm{RESET}}| < V_{\mathrm{SET}}$ reveals broader low-conduction intervals in the *i*-*v* plot. Therefore, compared to the case shown in Fig. 3.9a, asymmetric thresholds facilitate the better distinction of the low-conduction periods of an ARS and, thus, are preferable for such circuit configurations.

Finally, likewise in the anti-serially connected devices, we examined the stability of the anti-parallel switches when the individual devices were arbitrarily initialized to any possible intermediate memristance within the range [R_{ON}, R_{OFF}]. However, as mentioned before, the parallel connection of memristors always renders stable operation regardless of the initialization of the single devices. In the simulation result shown in Fig. 3.10 we notice the following: during the positive part of the voltage sweep the circuit settles to a resistive state close to the lowest composite memristance; during the negative voltage sweep it continues functioning as a ARS, but with notably higher currents and smaller overall memristance ratio than previously shown examples, due to the larger number of parallel connected devices.

Eventually, all simulated circuits up to this point are useful examples showing how groups of individual memristors can be effectively combined to deliver composite structures that produce combinatorial behavior. The interaction among devices is determined by the states of all interconnected elements. The dependence of the final states on the voltage pulsing protocol could find useful applications in electronic systems, e.g. in signal/pattern recognition [23, 24].

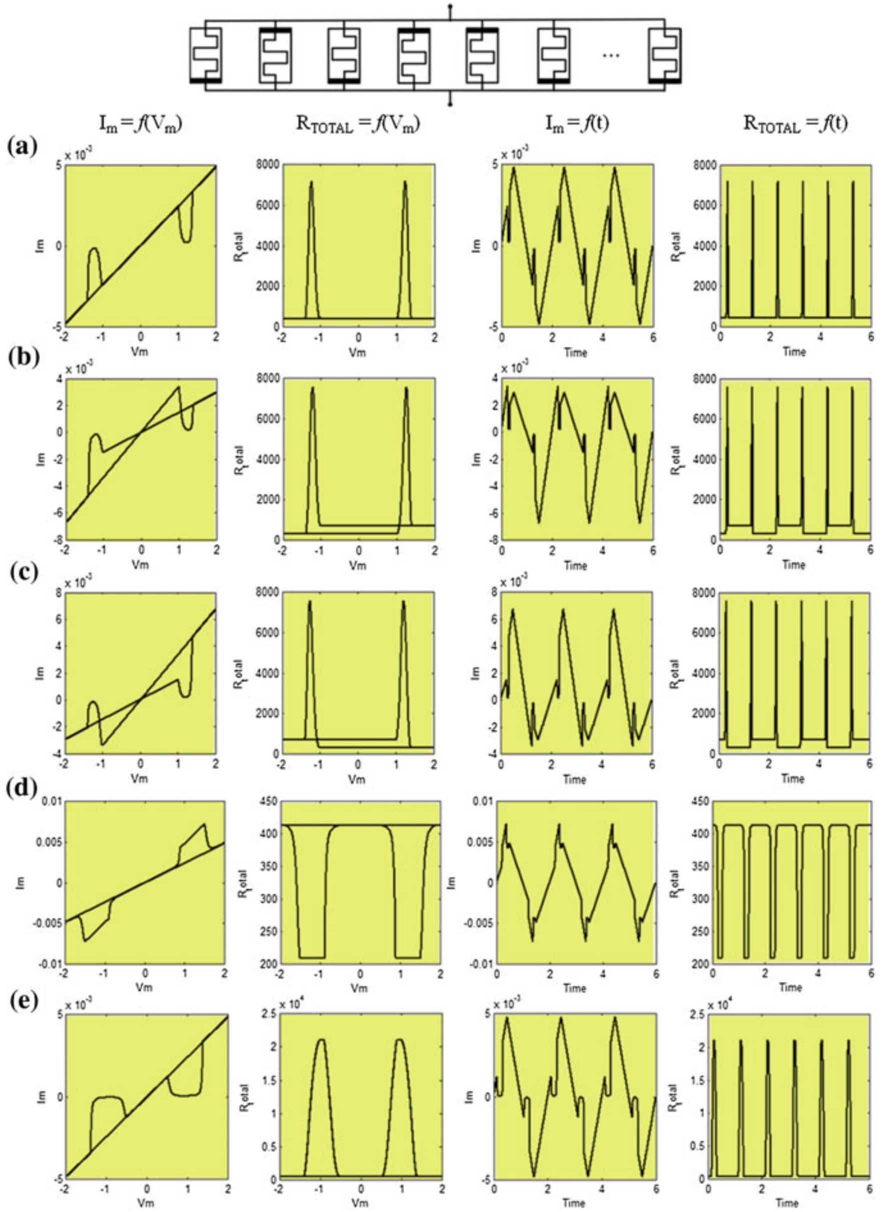


Fig. 3.9 Simulation results from the composite response of ten anti-parallel memristors subject to consecutive AC voltage sweeps. Examples for symmetric threshold voltages concern: **a** half FPMs and half RPMs; **b** four FPMs and six RPMs; **c** six FPMs and four RPMs. The last two examples refer to half FPMs and half RPMs with **d** (V_{RESET}, V_{SET}) = (−1.5 V, 0.5 V) and **e** (V_{RESET}, V_{SET}) = (−0.5 V, 1 V)

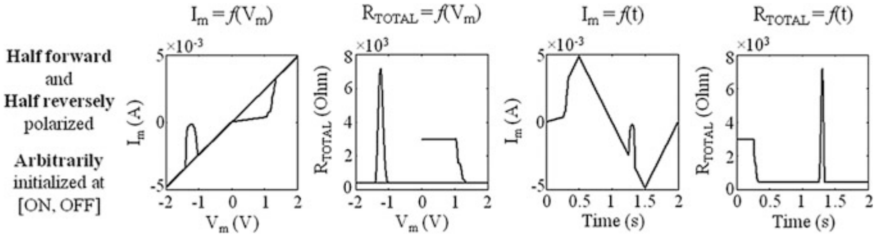


Fig. 3.10 Simulation results for the composite response of ten parallel memristors with half FPMs and half RPMs arbitrarily initialized in any possible intermediate memristance within the range [R_{ON}, R_{OFF}]

3.3 Generalized Concept for the Construction of Composite Memristive Systems

As described in the previous sections, large groups of individual memristors can be effectively combined in serial and/or parallel circuit configurations in order to produce composite structures that deliver intricate combinatorial behavior. The generalized concept for the construction of such composite memristive devices is summarized in Fig. 3.11. The main idea is based on the experimental observation that, in most memristive devices the memristance change-rate depends significantly on the magnitude of the applied voltage [1]; it is very small below (or high above) a threshold value. It should be mentioned, though, that although the memristance change-rate at low voltages has been experimentally verified as normally negligible,

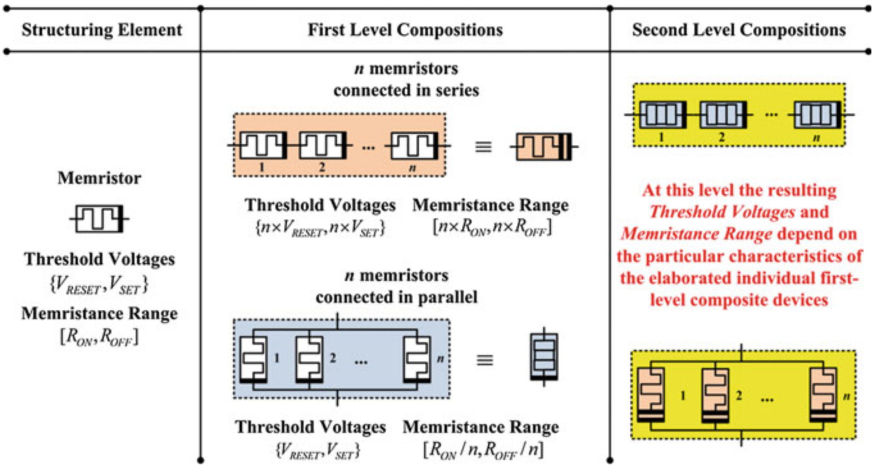


Fig. 3.11 Generalized concept for the construction of composite memristive structures

it can eventually lead to a finite memristance drift in circuits after long time of operation. Nevertheless, this parasitic effect becomes less important at significantly low operating voltages applied to the memristors and, in practice, can be corrected by periodic re-setting or/and circuit calibration.

Considering the single memristor as the basic structural element, in Fig. 3.11 we show how we can build higher-level compositions which are then handled as stand-alone two-terminal components. The final structures exhibit switching characteristics which combine the characteristics of all the employed structural elements; the latter are either single memristors or lower-level combinations. In first-level memristive compositions, which comprise serially connected identical FPMs, the state-transition will begin only when the applied voltage exceeds the resulting accumulated threshold of the n employed devices. Apparently, the range of the composite memristance is now $n \times [R_{\mathrm{ON}}, R_{\mathrm{OFF}}]$. The same principles apply when the memristors are connected in-parallel. A practical difference, though, lies in the magnitude of the total flowing current, which depends on the instantaneous combination of the memristances. When assuming n identical parallel FPMs with symmetrical thresholds, the flowing current is n times ($n\times$) larger than that of a single FPM because of the sub-multiple composite memristance boundaries which become $(1/n) \times \{R_{\mathrm{ON}}, R_{\mathrm{OFF}}\}$.

Let us now focus on the general case where a memristive component with given specifications for the composite memristance range, is required. We assume that the intended memristance range is related to that of the individual structural memristor as $k \times [R_{\mathrm{ON}}, R_{\mathrm{OFF}}]$ and that k can be given as (or approximated by) a fraction m/n; hence for the desired set of memristance boundaries, it is:

$$[desired] = k \times [memristor] = \frac{m}{n} \times [R_{ON}, R_{OFF}] = \begin{cases} (m \times [R_{ON}, R_{OFF}]) \times \frac{1}{n} & (a) \\ m \times \left([R_{ON}, R_{OFF}] \times \frac{1}{n}\right) & (b) \end{cases} \tag{3.1}$$

Formulations (a) and (b) of Eq. 3.1 correspond to two equivalent possible implementations of a composite memristive device, which meets the aforementioned requirement. Numerator m denotes the memristors connected in-series whereas denominator n corresponds to the parallel circuit branches. Therefore, one may choose between having: (a) n parallel branches each one comprising m memristors in series, or (b) m serially connected groups each one consisting of n parallel connected memristors. The respective circuit schematics are shown in Fig. 3.12. This general case follows by analogy the concept summary of Fig. 3.11 and assumes that all employed devices are identical and have the same polarity. Regarding the voltage thresholds of the composite devices, they are determined only by the number of memristors (or the groups of memristors) that are connected in series, as a consequence of the voltage divider circuit properties; i.e. in this case it is $m \times \{V_{\mathrm{RESET}}, V_{\mathrm{SET}}\}$ for both implementations of Fig. 3.12. Of course, composite voltage thresholds which are smaller than those of the basic structural element

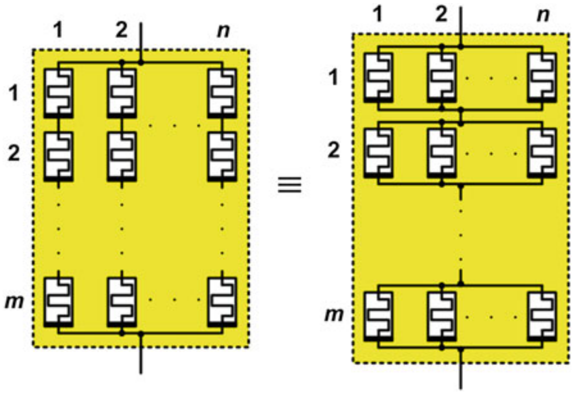


Fig. 3.12 Circuit schematics of two equivalent implementations of a composite memristive structure whose memristance range is related to that of the individual structural memristor as $(m/n) \times [R_{ON}, R_{OFF}]$. Numerator m denotes the memristors connected in-series whereas denominator n corresponds to the parallel circuit branches

cannot be obtained, unless different structural elements (with different threshold values) are available to be combined together.

In the following sections we provide several examples of such memristive implementations, combining different polarities, different initial states and/or switching characteristics, thus causing highly nontrivial composite responses to the applied voltages. We particularly focus on how the threshold-dependent switching behavior can be exploited in the presented memristive structures in order to build programmable memristive circuits and systems to be used in computing applications.

3.3.1 Circuit Examples Combining First/Second-Level Memristive Compositions

In order to show some of the computational characteristics of composite memristive circuit behavior, we simulated circuits which combine up to second-level memristive compositions. In the presented circuits the memristors are used in such a way that only voltages below the thresholds are applied to the devices during the analog mode of operation, whereas voltages of higher-amplitude (above the thresholds) are used only for programming.

In the first circuit example we utilize a single memristor and eight first-level compositions, each one comprising between two (2) and nine (9) memristors, jointly forming a second-level structure. For the purpose of this example, we selected each subsequent branch to include one more device than its previous one, and the total of nine (9) circuit branches are all connected in parallel, as shown in Fig. 3.13a. In the circuit schematic the symbol "#*x*" denotes *x* memristors in series. We note here that, although single memristors could be switched to all intermediate conducting states by using appropriate biasing [19], their behavior is strongly dependent on device variations, thus stable operation cannot be guaranteed. Programming a desired value of memristance with a given precision requires enough information on the memristor operation parameters and pulses of precise amplitude and duration. However, the presented memristive composition exhibits up to ten different and stable conducting states, distinguished with red dashed lines in Fig. 3.13d. All employed devices are initially found in the high resistive state (R_{OFF}), thus the memristance corresponding to each circuit branch is {R_{OFF},

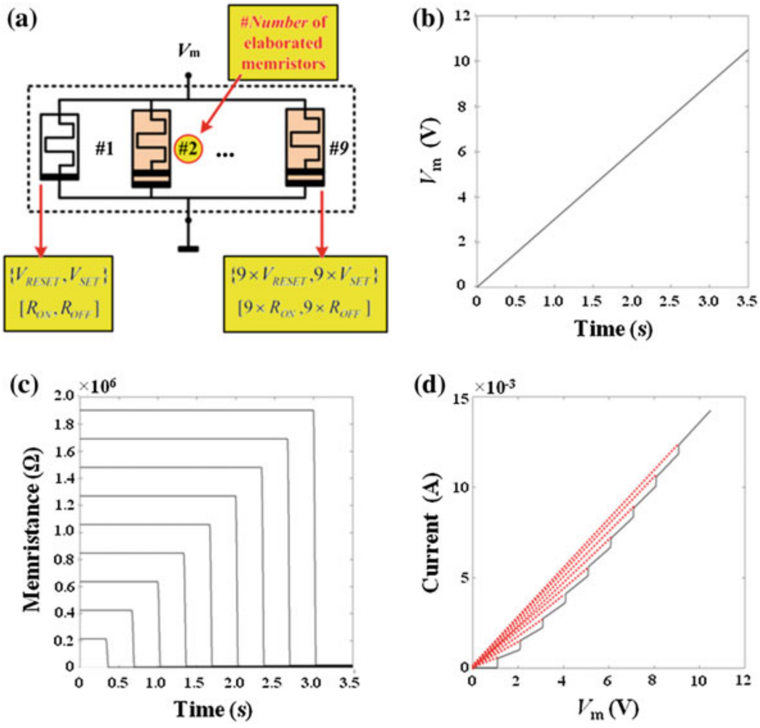


Fig. 3.13 Simulation of a multi-state memristive switch which combines a single memristor and eight (8) first-level structures, each one comprising between two (2) and nine (9) memristors in series. The circuit schematic is shown in (**a**). The applied ramp waveform voltage is given in (**b**). The time-evolution of the memristance of each circuit branch is shown in (**c**), whereas **d** presents the corresponding *I-V* characteristic. *Dashed* (*red*) *lines* highlight up to ten (10) different possible conducting states

2 × R_{OFF}, 3 × R_{OFF}, …, 9 × R_{OFF}} and the equivalent initial (k = 0) conductance (memductance) is calculated as follows:

$$\begin{aligned} G_{EQ}^{k=0} &= G_{1,OFF} + G_{2,OFF} + G_{3,OFF} + \cdots + G_{9,OFF} \\ &= G_{1,OFF} + \frac{G_{1,OFF}}{2} + \frac{G_{1,OFF}}{3} + \cdots + \frac{G_{1,OFF}}{9} \approx 2.83 \times G_{1,OFF} \end{aligned} \tag{3.2}$$

According to Eq. 3.2, the initial memductance is very low, hence it could be roughly approximated as almost equal to the memductance of the first branch ($G_{1,\mathrm{OFF}}$) for simplicity; possible R_{OFF} variation will negligibly affect this state. Passing from lower to higher conducting states is achieved by applying voltages which exceed a threshold value corresponding to one of the first-level connected components. We show this particular capability by applying the slowly increasing ramp waveform voltage of Fig. 3.13b to the terminals of the composite memristive circuit. Here, since each subsequent circuit branch includes one more memristor than its previous one, the resulting successive cumulative thresholds mutually differ by |1 V| because V_{SET} = 1V. The maximum magnitude of the applied ramp waveform voltage is selected sufficiently high to finally evoke switching to all the memristors. When the first memristor switches to the ON state (k = 1), its memductance increases by two orders of magnitude (because $G_{\mathrm{ON}}/G_{\mathrm{OFF}} = R_{\mathrm{OFF}}/R_{\mathrm{ON}} = 100$). According to Eq. 3.2, the equivalent memductance could then be approximated by $G_{\mathrm{EQ}}^{k=1} \approx G_{1,\mathrm{ON}}$. Next, when the memristors of the second (k = 2) and the third (k = 3) branch successively switch to the ON state, the equivalent memductance becomes:

$$\begin{aligned} &G_{EQ}^{k=2} = G_{1,ON} + G_{2,ON} + G_{3,OFF} + G_{4,OFF} + \cdots + G_{9,OFF} \\ &\approx G_{1,ON} + \frac{G_{1,ON}}{2} = \frac{3}{2} \times G_{1,ON} \end{aligned} \tag{3.3}$$

$$\begin{aligned} &G_{EQ}^{k=3} = G_{1,ON} + G_{2,ON} + G_{3,ON} + G_{4,OFF} + \cdots + G_{9,OFF} \\ &\approx G_{1,ON} + \frac{G_{1,ON}}{2} + \frac{G_{1,ON}}{3} = \frac{11}{6} \times G_{1,ON} \end{aligned} \tag{3.4}$$

It is therefore evident that this structure can function as a *multi-state resistive switch* of variable precision; the analog operating voltage range could be defined below the smallest of the switching thresholds of the composite device.

In the next example we employ first-level compositions which are arranged in a complementary serial manner. Specifically, we use five (5) pairs of composite anti-serial memristive elements, as shown in Fig. 3.14a. In each parallel branch there are as many FPMs as RPMs whereas each subsequent branch comprises twice (2×) as many FPMs and RPMs as the previous branch plus two more devices; the number of the memristors in each one of the five branches is: {2, 6, 14, 30, 62}. The aforementioned numbers were selected after experimentation in order to deliver

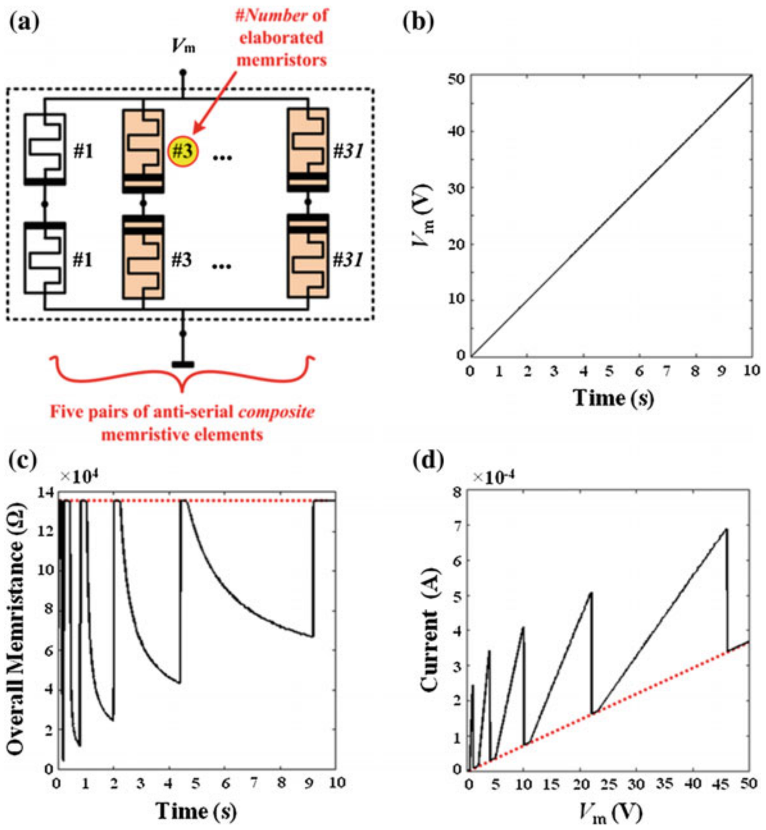


Fig. 3.14 Simulation of a composite device including first-level compositions arranged in a complementary serial manner. **a** The circuit schematic; **b** the ramp waveform voltage applied to the terminals of the composite structure; **c** the time-evolution of the overall memristance; **d** the stepwise incremental spike-like *i-v* characteristic. The *red dashed-lines* in **c** and **d** correspond to the highest achieved composite memristance

easily observable difference in the composite memductance values for the purposes of this study. The FPMs are initially found in the high resistive state (R_{OFF}), whereas the RPMs are set in the low resistive state (R_{ON}).

The positive applied voltage causes each time the FPMs of a particular branch to switch to the low resistive state first. At this moment and before the voltage drop on the RPMs of the same branch gets high enough to cause them to switch OFF, this particular branch comprises only low resistive devices. Hence the composite memductance of the memristive composition temporarily increases substantially until when the RPMs switch their state, thus restoring the composite memductance to a very low value; the latter is particularly designated with a dashed line in Fig. 3.14d.

In the same fashion, all employed anti-serially connected memristive compositions switch states and result in this stepwise incremental spiking *i-v* response, which could find useful applications mainly in electronic sensing systems [4, 17, 22]. However, the more memristors are added in series the higher the overall voltage threshold gets. Thus, in practical implementations the tolerance of individual memristors for the applied programming voltage will have to be considered. The total number of parallel branches will be limited. Otherwise, additional protecting circuitry will be required for the circuit branches with fewer elements if the voltage drop on each of them exceeds the specified tolerance limits.

3.3.2 Fine-Resolution Programmable Memristive Switches

In the multi-state resistive switch example of Fig. 3.13, the equivalent conductance (memductance) increment between subsequent state-changes was not equal. Nevertheless, appropriate selection of composite memristive components can deliver the desirable memductance values at the specified voltage thresholds. According to Fig. 3.11, when n identical memristors of the same polarity are connected in parallel, the overall memductance range is $n \times [G_{\mathrm{ON}}, G_{\mathrm{OFF}}]$, where $[G_{\mathrm{ON}}, G_{\mathrm{OFF}}]$ corresponds to a single memristor. In other words, increasing the number of memristors in a parallel group increases the conductance of this group. Connecting n such parallel groups (each having n memristors) in series increases the cumulative voltage threshold n times ($n\times$), whereas maintains the range of the total memductance of the branch equal to $(1/n) \times (n \times [G_{\mathrm{ON}}, G_{\mathrm{OFF}}]) = [G_{\mathrm{ON}}, G_{\mathrm{OFF}}]$.

Based on this property, Fig. 3.15 shows how memristive fine-resolution multi-state switches (MSS) of variable precision can be constructed. For representation purposes we use the particular symbol showed in Fig. 3.15a to define a group of x parallel devices. Assuming a number of circuit branches in parallel, as shown in Fig. 3.15b, if all the devices are in high memristance (R_{OFF}), the total memductance becomes very low and is roughly approximated here by the memductance of the first branch ($G_{1,\mathrm{OFF}}$) for simplicity. Moving to any of the higher conducting states is achieved by applying a voltage which exceeds the aggregate threshold of any of the successive vertical branches. The distinct conducting states and their approximate overall memductances are given by the following equations:

$$G_{EQ}^{k} = \begin{cases} G_{1,OFF} + G_{2,OFF} + \cdots + G_{n,OFF} \approx G_{OFF}, & k = 0 \\ G_{1,ON} + G_{2,OFF} + \cdots + G_{n,OFF} \approx G_{ON}, & k = 1 \\ G_{1,ON} + G_{2,ON} + \cdots + G_{n,OFF} \approx 2 \times G_{ON}, & k = 2 \\ \vdots & \vdots \\ G_{1,ON} + G_{2,ON} + \cdots + G_{n,ON} = n \times G_{ON}, & k = n \end{cases} \tag{3.5}$$

where $G_{EQ}^{k} \approx G_{EQ}^{k-1} + G_{ON}$.

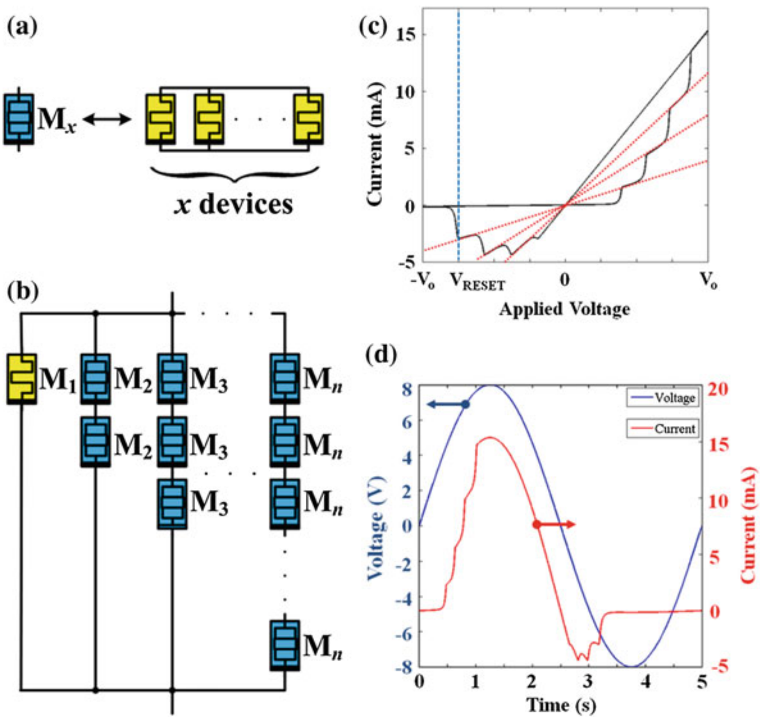


Fig. 3.15 **a**, **b** General methodology for the construction of memristive multi-state switches (MSS). **c** *i*-*v* and **d** *i*-*t* characteristics for a five-state simulated MSS under 8 V, 0.2 Hz sinusoidal signal particularly shown in **d**

According to Eq. 3.5, every subsequent step increases the composite memductance by a constant amount equal to G_{ON} (this does not include the first step from $k = 0$ to $k = 1$). Figure 3.15c, d show the *i*-*v* and *i*-*t* characteristics of a five-state memristive switch comprising four branches, under AC applied voltage. In simulation we used the following symmetrical voltage thresholds for all memristors: $\{V_{\mathrm{RESET}}, V_{\mathrm{SET}}\} = \{-1.5, 1.5\}$ V. The simulated MSS exhibits up to five different and stable conducting states, distinguished with red dashed lines in Fig. 3.15c, whereas a high-enough negative voltage resets the switch to its initial state ($k = 0$). In the same fashion one might expand this circuit in order to use up to n groups of n parallel memristors connected in series, so as to have n-fold (n×) threshold values but maintain the same memristance range in each additional circuit branch. Similarly to previous circuit examples, though, in any practical implementation the tolerance of individual memristors for the applied programming voltages will have to be considered.

Assuming that in practical devices both G_{ON} and G_{OFF} can vary, we next provide a short analysis regarding the susceptibility of the MSS to such variations. In fact, G_{OFF} variation among the devices will negligibly affect the performance of the

MSS. This is because when all memristors are in R_{OFF} ($k = 0$), the composite device still exhibits a very low conductance regardless of any existing variation in actual R_{OFF} values. Afterwards ($k > 0$), each time the memristors of a particular branch switch to R_{ON}, having a high $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ ratio (e.g. $\geq 10^2$) ensures that possible G_{ON}-variation among the employed devices still won't affect significantly the overall memristance. Moreover, owing to the robust nature of the proposed MSS approach, any device mismatch (e.g. failure to completely switch to R_{ON}) can be located by tracing the total current increment after each switching event during operation. For example, for a branch which comprises two pairs of parallel memristors connected between them in series (i.e. four memristors in total), according to some rough calculations we get that the overall memristance of the branch, due to a number of {1, 2 (one faulty device in each pair), 2 (both faulty devices in the same pair), 3, 4} device-mismatches, results approximately in {$1.5R_{\mathrm{ON}}$, $2R_{\mathrm{ON}}$, $1/2R_{\mathrm{OFF}}$, $1/2R_{\mathrm{OFF}}$, R_{OFF}} respectively, instead of R_{ON} as expected. Likewise, the possible effect of device mismatch can be calculated for even more memristors.

Compared with currently pursued methods for the realization of programmable resistors, which normally use arrays of weighted resistors and (typically MOS) switches [27, 28], the presented composite MSS are purely passive and comprise only memristors; hence they could prove advantageous in terms of circuit area, ease of fabrication, switching speed, and power consumption, to name a few. Moreover, by keeping only the parallel branches which provide a binary weighted sequence of successive memductances (i.e. where the ith branch gives a conductance of $2^i \times G_{\mathrm{ON}}$) the proposed switches could find application in n-bit memristive DAC circuits [29], making unnecessary the high precision tuning of single memristors.

3.4 Application of Composite Memristive Systems in Computing Circuits

Hybrid systems which integrate conventional technologies and memristors, are considered promising for analog computing applications. In this context, here we show how the fine-resolution multi-state switches (MSS) could be used to create a pulse-controlled stepwise signal generator.

The proposed closed loop circuit combines memristive computation with memristive feedback. As shown in Fig. 3.16a, the circuit consists of two identical parts placed reciprocally so that one's output becomes input to the other; the only difference lies in the type of CMOS transistors used in corresponding symmetric positions. In the MSS we replaced the first memristor with a resistor whose resistance is equal to $R_{\mathrm{ON},}$ so that the initial composite memductance ($k = 0$) is G_{ON} instead of $\approx G_{\mathrm{OFF}}$. Each MSS receives either a small voltage (V_{LOW}) or a high programming voltage of variable amplitude, which is internally produced during circuit operation; there is only one common external input, i.e. *step*. The current flowing through each MSS is driven to a current-to-voltage converter (I/V). The corresponding output voltage of each I/V component in a particular step k,

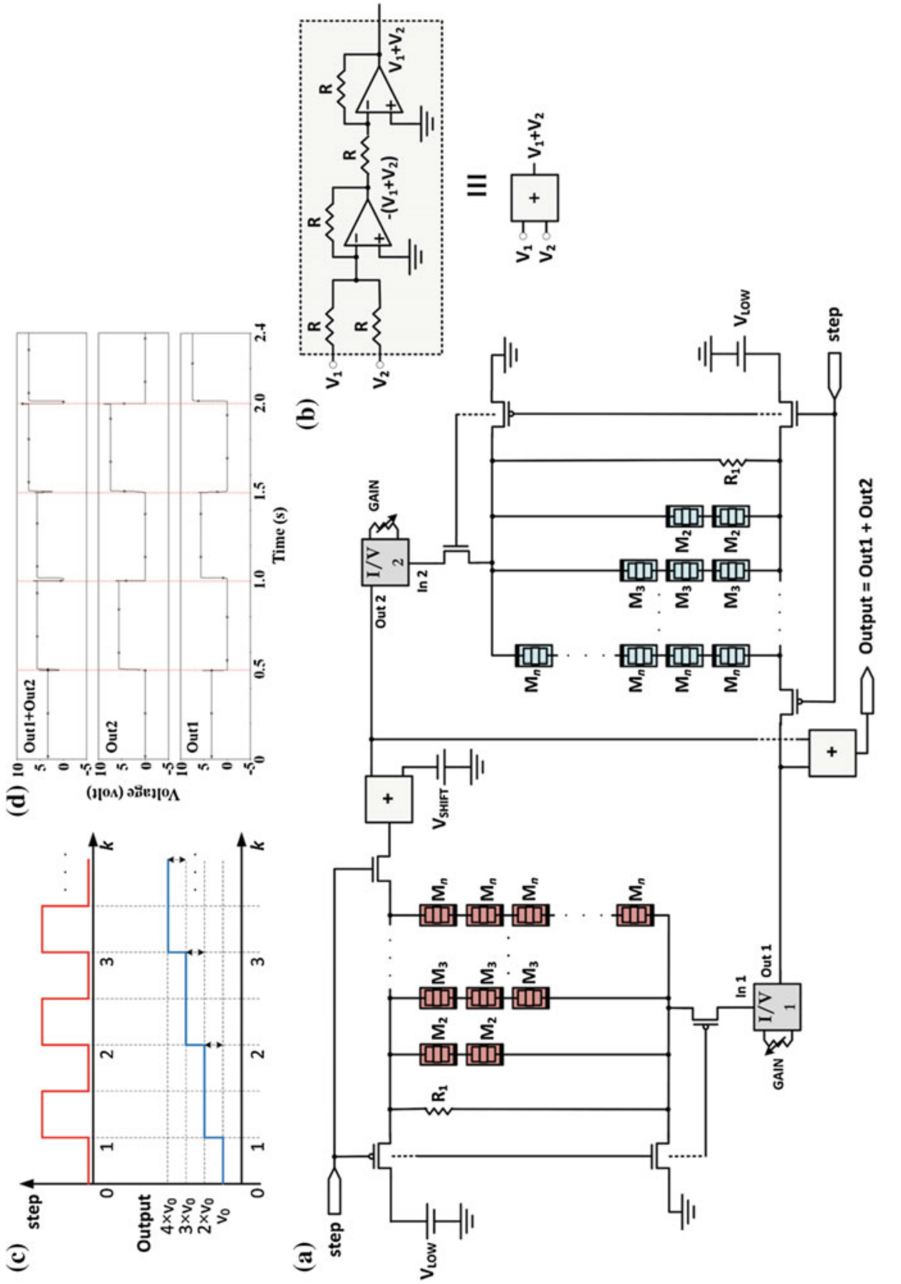


Fig. 3.16 Application of the proposed multi-state switches (MSS) in a pulse-controlled stepwise signal generator shown in **a**. **b** Explains the equivalent circuit of a summing block. The expected theoretical behavior of the circuit is shown in **c**. **d** Shows the SPICE simulation-based validation of the circuit when using three-state MSS

namely *Out*1(*k*) and *Out*2(*k*), are subsequently driven to a summing amplifier which produces the circuit output *Out*(*k*). Figure 3.16c shows qualitatively the expected behavior of the circuit when the two symmetric parts operate alternately as described below.

Assuming all memristors in the OFF state, initially ($k = 0$) *step* is ‘0’ so V_{LOW} is applied to the left multi-state switch (MSS$_1$) and I/V$_1$ produces *Out*1 ($k = 0$) ≈ GAIN × V_{LOW} × G_{ON}. The *Out*1 voltage is also fed to the MSS on the right (MSS$_2$) so that the memristors of its second branch (M$_2$) are forced to switch ON because *Out*1($k = 0$) exceeds the respective threshold. At the same time, the input of I/V$_2$ is grounded so *Out*2($k = 0$) is zero. Therefore, the circuit output is *Out* (0) = *Out*1(0) + *Out*2(0) = *Out*1(0). Next, when *step* becomes ‘1’, V_{LOW} is applied to MSS$_2$ and I/V$_2$ gives *Out*2($k = 1$) ≈ GAIN × V_{LOW} × 2 × G_{ON} = 2 × *Out*1(0). *Out*2(1) is fed to the MSS$_1$ after summing a negative DC voltage V_{SHIFT} whose value is equal to GAIN × V_{LOW} × G_{ON}. This way, each time *step* is ‘1’, the feedback voltage applied to MSS$_1$ is the same with the one which was previously applied to MSS$_2$. Hence, the memristors of branch M$_2$ of MSS$_1$ now switch ON. Afterwards *step* returns to ‘0’ and I/V$_1$ gives *Out*1($k = 1$) ≈ GAIN × 2 × V_{LOW} × G_{ON} = *Out*2(1), i.e. the output is maintained when *step* returns to ‘0’. *Out*1 now forces the memristors of branch M$_3$ of MSS$_2$ to switch ON. In the same fashion, setting *step* = ‘1’ increases the circuit output by a constant amount (given as v_0) which is equal to the initial output of the circuit. Returning *step* to ‘0’ keeps the output unaffected and enables receipt of feedback. This way, computation takes place on the right part (MSS$_2$) whereas the state of the system is stored on the left part (MSS$_1$) of the proposed circuit.

The total number of distinct output levels depends on the two MSS. Values for V_{LOW} and for the external GAIN of I/V are selected according to the switching thresholds of memristors. The circuit was simulated in SPICE where we used: $\{V_{\mathrm{RESET}}, V_{\mathrm{SET}}\} = \{-2, 2\}$ V, $V_{\mathrm{LOW}} = 4$ V, GAIN = 4 K, and R = 1 K. The two identical MSS comprised three branches, so in the simulation results of Fig. 3.16d we increase *step* up to $k = 3$ and, thus, observe three different output voltage levels. Resetting the circuit could be easily done by resetting the state of the MSS as described in the previous section.

3.5 Overview and Discussion

The greatest impact of memristors lies in their analog nature, on which most of the proposed relevant applications are based. However, it still remains to be seen whether such analog operation will guarantee the desired accuracy and reliability as well. So far, programming memristors to any intermediate state requires input pulses of precise amplitude and duration, i.e. two control parameters. The latter not only is difficult to achieve but also susceptible to device variation regarding both $\{R_{\mathrm{ON}}, R_{\mathrm{OFF}}\}$ as well as $\{V_{\mathrm{RESET}}, V_{\mathrm{SET}}\}$.

The presented methodology for the construction of multi-state memristive switches, though, is based on threshold-type switching memristors and aims to alleviate their programming procedure by eliminating pulse-duration from the control parameters. The programming pulses only need to be of appropriate amplitude so as to exceed the threshold which corresponds to a particular final resistive state. Composite multi-state memristive switches can be achieved using complex interconnections of memristors in series or in parallel. The final selection depends on the number of desired intermediate states, the applied voltage regime, and the actual value of the resistive states.

Complex combinations of memristors could be also used to improve the performance characteristics of multi-state and/or complementary resistive switches (CRS), providing better-defined reading regions and higher switching voltages which may suit better the target application. As discussed before, potential applications range from adaptive electronics and sensing systems to computing circuits; the spiking switching behavior demonstrated by several memristive circuit configurations, could find application in neuromorphic computations.

All-inclusive, this chapter presented a nice overview of the rich dynamic response of complex memristive element combinations, whose structural element exhibits threshold-type switching behavior. Additionally, it provided all necessary introductory material for the better comprehension of the following chapter, where collective threshold-type response is used for the construction of logic circuits.

References

1. Y.V. Pershin, M. Di Ventra, Memory effects in complex materials and nanoscale systems. Adv. Phys. **60**(2), 145–227 (2011)
2. L.O. Chua, Resistance switching memories are memristors. Appl. Phys. A Mater. Sci. Process. **102**(4), 765–783 (2011)
3. S. Hamdioui, H. Aziza, G.C. Sirakoulis, Memristor based memories: technology, design and test, in *9th IEEE International Conference on Design and Technology of Integrated System in Nanoscale Era (DTIS)*, Santorini island, Greece (2014)
4. Y. Pershin, M. Di Ventra, Practical approach to programmable analog circuits with memristors. IEEE Trans. Circ. Syst. I Reg. Papers **57**(8), 1857–1864 (2010)
5. M. Gholipour, N. Masoumi, Design investigation of nanoelectronic circuits using crossbar-based nanoarchitectures. Microelectron. J. **44**(3), 190–200 (2013)
6. K.H. Kim, S. Gaba, D. Wheeler, J.M. Cruz-Albrecht, T. Hussain, N. Srinivasa, W. Lu, A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. Nano Lett. **12**(1), 389–395 (2012)
7. E. Lehtonen, M. Laiho, Stateful implication logic with memristors, in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, San Francisco, CA (2009)
8. Y.V. Pershin, M. Di Ventra, Solving mazes with memristors: a massively parallel approach. Phys. Rev. E **84**, 046703 (2011)
9. Y.V. Pershin, M. Di Ventra, Self-organization and solution of shortest-path optimization problems with memristive networks. Phys. Rev. E **88**, 013305 (2013)

10. R.K. Budhathoki, M.P. Sah, S.P. Adhikari, H. Kim, L.O. Chua, Composite behavior of multiple memristor circuits. IEEE Trans. Circuits Syst. I Reg. Papers **60**(10), 2688–2700 (2013)
11. Y. Pershin, V. Slipko, M. Di Ventra, Complex dynamics and scale invariance of one-dimensional memristive networks. Phys. Rev. E **87**, 022116 (2013)
12. J.R. Heath, P.J. Kuekes, G.S. Snider, R.S. Williams, A defect-tolerant computer architecture: opportunities for nanotechnology. Science **280**(5370), 1716–1721 (1998)
13. I. Vourkas, G.C. Sirakoulis, A threshold-based approach for modeling memristive devices and systems, in *4th International Conference from Nanoparticles and Nanomaterials to Nanodevices and Nanosystems (IC4N)*, Corfu, Greece (2013)
14. International Technology Roadmap for Semiconductors (ITRS) (2013). Available: http://www.itrs.net/. Accessed June 2014
15. I. Vourkas, G.C. Sirakoulis, Modeling memristor-based circuit networks on crossbar architectures, in *Memristor Networks*, ed. by A. Adamatzky, L. Chua (Springer, Switzerland, 2014), pp. 505–535
16. I. Vourkas, G.C. Sirakoulis, On the generalization of composite memristive network structures for computational analog/digital circuits and systems. Microelectron. J. **45**(11), 1380–1391 (2014)
17. S. Shin, K. Kim, S. Kang, Memristor applications for programmable analog ICs. IEEE Trans. Nanotechnol. **10**(2), 266–274 (2011)
18. S. Shin, K. Kim, S.M. Kang, Memristor-based fine resolution programmable resistance and its applications, in *IEEE International Conference on Communications, Circuits and Systems (ICCCAS)*, Milpitas, CA (2009)
19. F. Alibart, L. Gao, B.D. Hoskins, D.B. Strukov, High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. Nanotechnology **23**(7), 075201 (2012)
20. I. Vourkas, G.C. Sirakoulis, A novel design and modeling paradigm for memristor-based crossbar circuits. IEEE Trans. Nanotechnol. **11**(6), 1151–1159 (2012)
21. I. Vourkas, A. Batsos, G.Ch. Sirakoulis, SPICE modeling of nonlinear memristive behavior. Int. J. Circ. Theor. Appl. **43**(5), 553–565 (2015)
22. T. Driscoll, J. Quinn, S. Klein, H.T. Kim, B.J. Kim, Y. Pershin, M. Di Ventra, D.N. Basov, Memristive adaptive filters. Appl. Phys. Lett. **97**(9), 093502 (2010)
23. S.-J. Lee, S.-J. Kim, K. Cho, S.-M. Kang, K. Eshraghian, Complementary resistive switch-based smart sensor search engine. IEEE Sens. **14**(5), 1639–1646 (2014)
24. F. Corinto, A. Ascoli, M. Gilli, Class of all i–v dynamics for memristive elements in pattern recognition systems, in *IEEE International Joint Conference on Neural Networks*, San Jose, CA (2011)
25. E. Linn, R. Rosezin, C. Kugeler, R. Waser, Complementary resistive switches for passive nanocrossbar memories. Nat. Mater. **9**(5), 403–406 (2010)
26. T. Liu, Y. Kang, M. Verma, M.K. Orlowski, Switching characteristics of antiparallel resistive switches. IEEE Trans. Electron Device Lett. **33**(3), 429–431 (2012)
27. A. Torralba, J. Galan, C. Lujan-Martinez, R.G. Carvajal, J. Ramirez-Angulo, A. Lopez-Martin, Comparison of programmable linear resistors based on quasi-floating gate MOSFETs, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seattle, WA, USA (2008)
28. E. Ozalevli, P.E. Hasler, Tunable highly linear floating-gate CMOS resistor using common-mode linearization technique. IEEE Trans. Circuits Syst. I Reg. Papers **55**(4), 999–1010 (2008)
29. L. Gao, F. Merrikh-Bayat, F. Alibart, X. Guo, B.D. Hoskins, K.-T. Cheng, D.B. Strukov, Digital-to-analog and analog-to-digital conversion with metal oxide memristors for ultra-low power computing, in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Brooklyn, NY (2013)

Chapter 4
Memristor-Based Logic Circuits

4.1 Introduction

Amongst several emergent applications of the memristance switching phenomenon [1–7], implementation of logic circuits is gaining considerable attention [8–19]. Memristor-based circuits open new pathways for the exploration of advanced computing architectures as promising alternatives to conventional integrated circuit technologies which are facing serious challenges related to continuous scaling [1, 8, 20].

Memristors provide an unconventional computation framework, different from familiar paradigms, which combines information processing and storage in the memory itself. Such framework is determined more by the device properties than any previously conceived logic paradigm; this justifies the growing interest in the development of computational methods and circuits which exploit the favorable performance merits of memristors, concerning their nonvolatility, fast switching speed, small area and energy dissipation [21–23]. However, up to now no standard logic design methodology exists. Related work on memristor-based logic/computational circuits could be classified as follows:

- *Material implication* (*IMPLY*): computation of Boolean functions using *imply* and *reset* logic operations [9, 11, 13, 14];
- *Hybrid memristor/CMOS*: combination of memristors and CMOS components in Boolean logic [10, 24] and threshold logic computations [15, 25];
- *Programmable interconnects*: logic operations in crossbar arrays relying on the use of programmable interconnections between crossbar nanowires [26–29];
- *Network-based computations*: massively parallel computations relying on array-like structures which accommodate networks of memristive components [30–34].

All mentioned categories constitute novel design paradigms which make possible fundamental logic operations as well as computations that are not possible or

I. Vourkas and G.Ch. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Emergence, Complexity and Computation 19,
DOI 10.1007/978-3-319-22647-7_4

have inefficient realization in the present and established design domain. However, they all suffer from some limitations. For instance, the sequential nature of memristive IMPLY logic is a major disadvantage; lengthy sequences of logic operations are required in order to synthesize Boolean functions. Moreover, the majority of memristor/CMOS logic design concepts use linear memristors with no switching thresholds, which are much slower than threshold-type switching devices. Also, in threshold logic the state of memristors has to be continuously controlled using programming pulses of precise amplitude and duration. Similarly, crossbar-based programmable circuits suffer from large amount of leakage current flowing though the cells that are not participating in computation, whereas network-based computations require that the entire problem-space is projected on the computing medium (network); something unfeasible or very difficult to accomplish in any case. Therefore, it is not immediately clear what kind of computing architectures would in practice benefit the most from the computing capabilities of memristors.

This chapter addresses memristive logic design and computational methodologies aiming to approach this novel area of research while motivating for further research on innovative design strategies that comply with emerging technologies. First, a summary of the most recognized memristive logic design concepts is provided. Then two novel logic design paradigms are presented which aim to address several of the aforementioned drawbacks and to facilitate the incorporation of memristors in currently established logic circuit architectures; thus they could be promising candidates to be used in future electronic system design. The proposed design paradigms are validated through SPICE-based simulations for a variety of complex combinational logic circuits.

4.2 Switching Dynamics of Threshold-Type Memristors and Memristive Compositions

Operation of the memristor-based circuits, presented in this chapter, is mainly based on the collective dynamics of multiple connected threshold-type switching memristors. Therefore, understanding their overall behavior requires the deep comprehension of the switching dynamics of individual memristors and of memristive compositions, comprising a network of interconnected devices with either the same or opposite polarities.

In Fig. 4.1 we qualitatively show the current-voltage (*i*–*v*) characteristics of memristors with opposite polarities and of their serial/parallel combination under AC applied voltage. We assume only bipolar devices, i.e. memristors which require both polarities to switch their state. We refer to a memristor being forward/reversely polarized (FPM/RPM) when the voltage is applied to the top/bottom terminal with the bottom/top terminal being grounded; bottom terminal is always denoted by the black thick line in the circuit schematic. For the employed devices we assume asymmetric thresholds and the following initial states: {RPM, FPM} = {R_{ON}, R_{OFF}}. It can be observed that memristors with opposite polarities generally

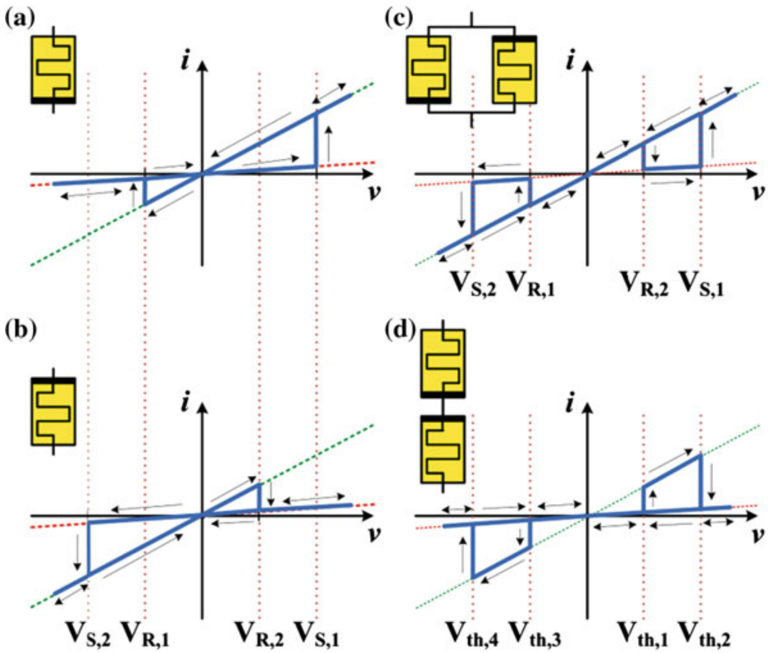


Fig. 4.1 Qualitative current-voltage (*i*–*v*) characteristics of **a**, **b** individual memristors with opposite polarities and **c**, **d** their serial/parallel combination

demonstrate reversed behavior to the applied signals. Thereinafter, we assume that the memristance of a FPM will decrease/increase when the latter is forward/reversely biased, whereas a RPM has the opposite response. According to Fig. 4.1a, when a positive voltage applied to a FPM reaches its “set threshold” ($V_{S,1}$), the device switches to its low resistive state (R_{ON}). Then, an ohmic behavior is observed until a “reset threshold” ($V_{R,1}$) is reached and the device returns to the high resistive state (R_{OFF}). The *i*–*v* graph of a RPM, shown in Fig. 4.1b, is symmetric to that of a FPM and has the opposite voltage thresholds.

Figure 4.1c illustrates the composite *i*–*v* response of two reciprocal memristors connected in parallel. When a positive applied voltage reaches the “reset threshold” of the RPM ($V_{R,2}$), its state is changed and the total current drops. Next, when the voltage exceeds the “set threshold” ($V_{S,1}$) of the FPM, it switches to R_{ON} and the total current rises again. Afterwards, the composite device exhibits an ohmic behavior unless a negative voltage is applied. This behavior is opposite to that of the anti-serially connected memristors (forming a complementary resistive switch—CRS [35]) presented in Fig. 4.1d. However, here the voltage thresholds ($V_{th,1} - V_{th,4}$) cannot be formerly known exactly because the memristors form a voltage divider; the voltage drop over each element depends on the total applied voltage, on the internal states of the devices, and on their particular switching characteristics. In such composite memristive structures there is no need to particularly access each memristive device so as to adjust its memristance. In fact, regardless of the current state of the devices, resetting can be done by applying a negative programming voltage which exceeds the leftmost threshold shown in the corresponding *i*–*v* graph. In both types of connection, such pulse will eventually reset all memristors to the initial boundary resistive states (being either R_{ON} or R_{OFF}) depending on their polarity.

An outstanding feature that appears for both the series and the parallel connection of reciprocal memristors, is the perfectly symmetric *i*–*v* curve which resembles a truncated Ohm's Law; current is piece-wise linear with the voltage. However, if the devices are all placed with the same polarity, their overall behavior resembles that of a memristor of the same polarity whose properties combine the properties of the individual devices. For example, two FPMs in series will both switch their states simultaneously when the applied voltage exceeds the sum of the individual thresholds, which are $2 \times \{V_{\mathrm{RESET}}, V_{\mathrm{SET}}\}$, whereas the composite memristance will range within $2 \times [R_{\mathrm{ON}}, R_{\mathrm{OFF}}]$. Similarly, having two memristors with the same polarity in parallel, reproduces the individual memristive behavior while achieving higher total current values because of the lower composite memristance, which ranges within $1/2 \times [R_{\mathrm{ON}}, R_{\mathrm{OFF}}]$ [36].

4.3 Popular Logic Design Concepts Based on Memristors

Some of the most recognized memristive Boolean logic design concepts are summarized in this section. Amongst all available identified concepts in the recent literature, the selected ones are all based on collective memristive dynamics and have a number of characteristics in common with the novel design methodologies which are presented afterwards; thus this short prior discussion is useful and facilitates a fairer comparison between them.

4.3.1 Material Implication (IMPLY)—Based Logic

In 1910, Whitehead and Russell in their book *Principia Mathematica* [37] described four fundamental logic operations. Three of them were the well-known in the electrical engineering and computing communities AND, OR, and NOT which form a computationally complete set. However, there was another operation which Russell named "material implication", *p* IMP *q* (i.e. "*p* implies *q*" or "if *p*, then *q*") whose truth table is shown in Fig. 4.2a. The IMP and FALSE operations (where the FALSE operation always yields logic '0') form a computationally complete logic basis. In [11] it was first shown by the Hewlett-Packard (HP) Labs group that material implication is naturally realized in a simple circuit combining a conventional resistor with two memristors, as shown in Fig. 4.2b. The logic state variable

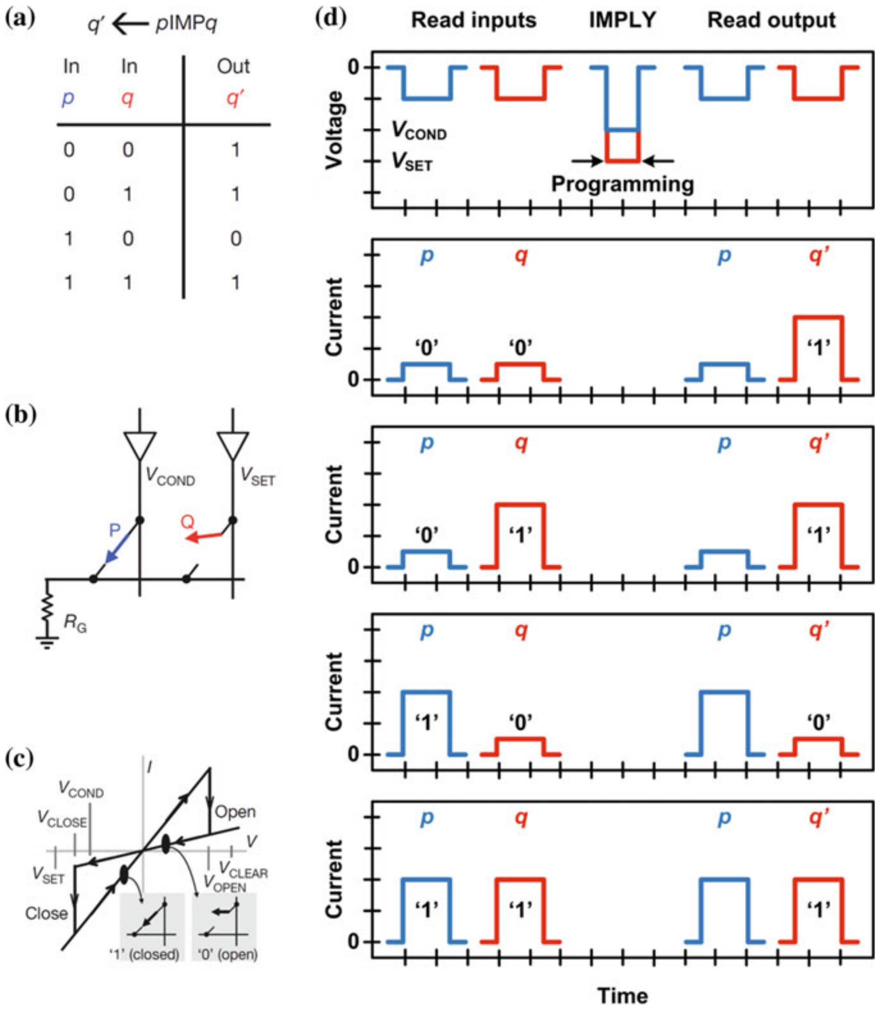


Fig. 4.2 Illustration of the IMP operation for all valid input combinations of variables p and q. **a** The truth table for the operation $q' \leftarrow p$ IMP q. **b** The corresponding circuit schematic comprising two memristors and a load resistor. **c** The qualitative (idealized) memristive electrical characteristics with abrupt effective voltage thresholds. **d** Qualitative *v*-*t* and *i*-*t* graphs of memristive IMP; the *blue* (*red*) *curves* qualitatively demonstrate the applied voltages and the (absolute) corresponding currents read at memristor $P(Q)$ before and after the IMP voltage pulses. The measured low- and high-current values reproduce the IMP truth table

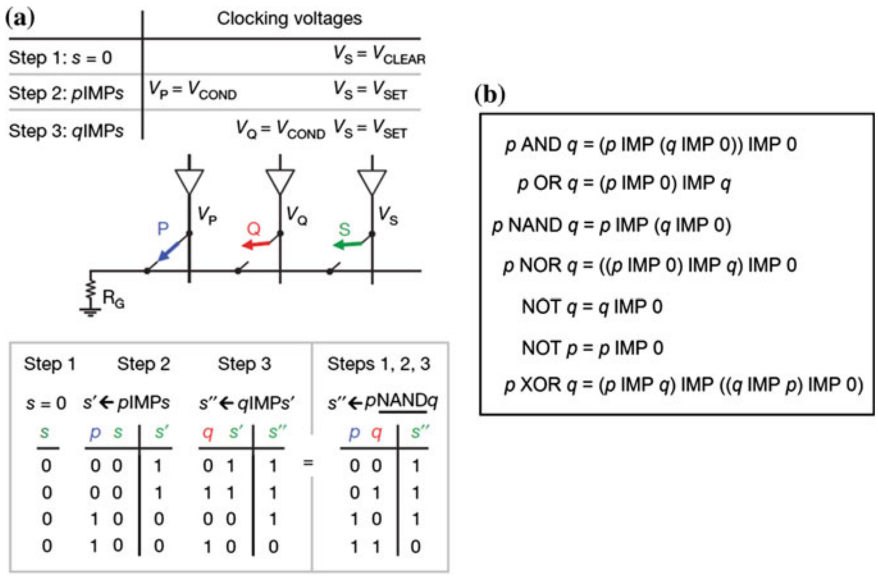
of this unconventional computation framework is the memristance of the devices, which is why IMPLY logic operation was characterized in the literature as "stateful".

Memristive IMPLY logic relies on using threshold-type switching memristors; the devices undergo a rapid memristance change if the applied voltage exceeds either the V_{OPEN} or V_{CLOSE} switching thresholds. For readability reasons we illustrate the qualitative *i*–*v* graph for a single memristor in Fig. 4.2c) (similar to the *i*–*v* of a RPM in Fig. 4.1b): we define the R_{CLOSED} (R_{ON}) state to represent logic '1' and the R_{OPEN} (R_{OFF}) state logic '0', respectively. The memristors are driven by tri-state voltage drivers which provide a high-impedance output state when not driven. A given memristor may be "set" (assigned logic '1') by applying a voltage V_{SET} (this is the operation TRUE). Similarly, the device may be "cleared" (assigned logic '0') by applying a voltage V_{CLEAR} (i.e. the operation FALSE). An auxiliary voltage V_{COND}, whose magnitude is smaller than V_{SET}, is also used to facilitate conditional switching.

The key to perform a memristive IMP operation is to understand the conditional toggling property. The operation $q \leftarrow p$ IMP q is implemented by simultaneously applying a V_{SET} pulse to Q and a V_{COND} pulse to P to execute a conditional switching operation. If V_{SET} is applied to Q alone, it executes the unconditional operation $q \leftarrow 1$, while the V_{COND} pulse applied to P alone implements $p \leftarrow p$. When applied together, though, the two pulses interact through P, Q and the load resistor R_{G} to cause state-changes depending on the current states of p and q: if P is in R_{OFF} state ($p = 0$) it has little influence on the voltage divider formed by Q and R_{G}, thus Q is set ($q \leftarrow 1$) while P is left unchanged ($p \leftarrow p$). However, if P is in R_{ON} state ($p = 1$), the V_{COND} pulse on P "shorts out" the voltage divider and both P and Q are not affected ($q \leftarrow q$, $p \leftarrow p$). The selection of R_{G} is important to the outcome of the operation. Its resistance is chosen such that $R_{\mathrm{ON}} < R_{\mathrm{G}} < R_{\mathrm{OFF}}$. When both P and Q are in R_{OFF}, then V_{COND} and V_{SET}, respectively, drop mainly across each device because $R_{\mathrm{G}} < R_{\mathrm{OFF}}$; this leaves P in R_{OFF} but switches Q to R_{ON}. However, if P is in R_{ON}, the voltage on the common node is $\approx V_{\mathrm{COND}}$ since $R_{\mathrm{ON}} < R_{\mathrm{G}}$, hence the voltage drop across Q is $\approx V_{\mathrm{SET}} - V_{\mathrm{COND}}$, which leaves Q in R_{OFF}.

In Fig. 4.2d we demonstrate the memristive IMP operation via qualitative *v*-*t* and *i*-*t* graphs, which correspond to the experiments published in [11]. First P and Q memristors were initialized to the desired states by applying appropriate voltage pulses, V_{SET} or V_{CLEAR}; the initial states were verified by applying a small reading voltage and measuring the resulting current. Then V_{COND} and V_{SET} were simultaneously applied to memristors P and Q, respectively. The corresponding initial memristances p and q are the inputs of the gate, whereas the output is the final memristance of Q (the result is written in the logic state q). Note that the memristance of both memristors changes during operation, i.e. the computation is destructive for both inputs. The qualitative conditional toggling results of Fig. 4.2d are in line with the corresponding truth table of the memristive $q \leftarrow p$ IMP q.

A major disadvantage of memristive IMPLY logic is the necessity to perform lengthy sequences of stateful logic operations in order to synthesize a given Boolean function. According to [14] where a multi-input implication operation was introduced with complementary representation of input variables, up to $2^{n-1} + 1$ computational steps are required for the synthesis of an arbitrary *n*-input Boolean function. This drawback is particularly seen in Fig. 4.3a where the logic operation $s \leftarrow p$ NAND q is sequentially performed using three memristors. In the same



	Clocking voltages	
Step 1: $s = 0$		$V_S = V_{CLEAR}$
Step 2: $p$IMP$s$	$V_P = V_{COND}$	$V_S = V_{SET}$
Step 3: $q$IMP$s$	$V_Q = V_{COND}$	$V_S = V_{SET}$

Step 1	Step 2			Step 3				Steps 1, 2, 3		
$s = 0$	$s' \leftarrow p$IMP$s$			$s'' \leftarrow q$IMPs'				$s'' \leftarrow p\overline{\text{AND}}q$		
s	p	s	s'	q	s'	s''		p	q	s''
0	0	0	1	0	1	1	=	0	0	1
0	0	0	1	1	1	1		0	1	1
0	1	0	0	0	0	1		1	0	1
0	1	0	0	1	0	0		1	1	0

p AND q = (p IMP (q IMP 0)) IMP 0
p OR q = (p IMP 0) IMP q
p NAND q = p IMP (q IMP 0)
p NOR q = ((p IMP 0) IMP q) IMP 0
NOT q = q IMP 0
NOT p = p IMP 0
p XOR q = (p IMP q) IMP ((q IMP p) IMP 0)

Fig. 4.3 **a** The logic operation $s \leftarrow p$ NAND q performed as a sequential operation with three memristors. The sequence of voltages applied to obtain the NAND operation with P, Q and S, the corresponding circuit schematic, and the truth tables showing the equivalence of the sequence of operations to NAND. **b** Several binary Boolean operations on two logic variables using series of IMP and/or FALSE operations

fashion, the necessary sequential computational steps for several other binary Boolean functions on two logic variables are given in Fig. 4.3b. Apparently, the practical utility of the memristive IMPLY logic design scheme requires operations to become as parallel as possible; hence further research is still necessary at the circuit and/or architecture level.

4.3.2 MRL—Memristor "Ratioed" Logic

Integrating memristors and CMOS to perform logical operations could be proved beneficial given that the memristors could be fabricated within the CMOS metal layers, thus significant physical integration area could be saved while increasing the device density. To this end, Kvatinsky et al. [24] proposed a hybrid CMOS-memristive logic family, which they called memristor "ratioed" logic (MRL). MRL constitutes a combination of memristors and CMOS transistors which has the potential to save significant amount of chip area as the number of circuit inputs increases. In such CMOS-compatible logic family, AND and OR logic gates are based on memristors, whereas a CMOS NOT gate is used to provide a complete logic gate-set and to restore degraded signals.

This logic family uses the programmable resistance of memristors to compute Boolean AND/OR functions with voltage being the logic state variable. Both AND and OR logic gates consist of two memristors connected in series with opposite polarity as shown in Fig. 4.4a, b; the only difference between the two gates consists in the polarity of the memristors with respect to where the inputs are applied. The output node is the common node of the connected memristors, whereas each input signal is applied to the floating terminal of each memristor.

The overall behavior of both gates resembles that of a complementary resistive switch (CRS) [35]. Owing to their different polarity, the memristors tend to switch their states in a reciprocal way which depends on the applied input signals. Both logic gates react similarly to identical inputs (both being either logic ‘1’ or logic ‘0’); since the voltage drop between inputs is zero, the voltage at the output node follows the input voltage. However, when the inputs are different there is current flowing from the high voltage terminal (where the ‘1’ is applied) to the grounded terminal (where the ‘0’ is applied), thus potentially affecting the memristance of the devices.

This case is shown in Fig. 4.4c for an OR logic gate. Assuming initially $R_1 = R_{\mathrm{OFF}}$ and $R_2 = R_{\mathrm{ON}}$, at the end of the computational process the memristors have changed their initial states. For the AND logic gate the different polarities have as a result the state of each memristor to switch in the opposite manner, as shown in Fig. 4.4d, f. Assuming a high memristance ratio $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$, the output voltage of the logic gates is determined by the voltage divider across both of the memristive devices. However, this voltage divider impacts the level of the output signal. Although the signal degradation is minor when $R_{\mathrm{OFF}} \gg R_{\mathrm{ON}}$, for cascaded logic gates this degradation accumulates and may become significant; therefore signal restoration is occasionally required. The number of inputs for both gates can be

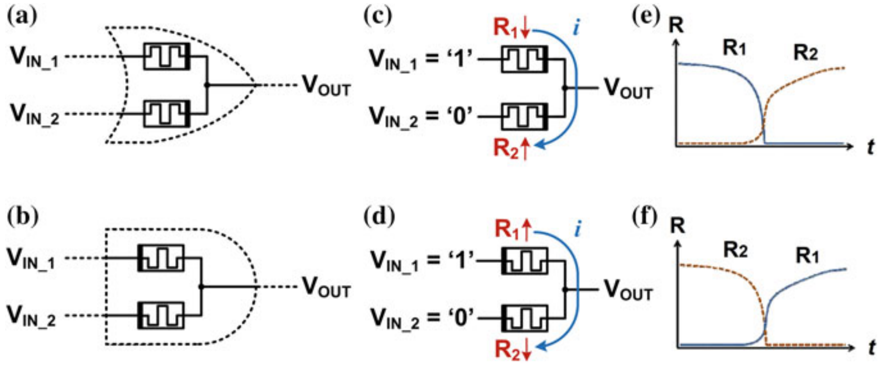


Fig. 4.4 **a**, **b** Circuit schematic of the MRL OR/AND logic gates [24]. **c**, **d** Circuit behavior when different input logic signals are applied. **e**, **f** Qualitative time-evolution of the memristance of the two memristors for the case shown in (**c**, **d**)

extended in a similar way as in diode logic [38], on which the MRL concept is based.

MRL evidently provides only a restricted set of possible Boolean logic operations, while it also relies on using linear memristors, which however are much slower than nonlinear (threshold-type) [21]. Therefore, the switching time is dependent on the applied voltage. A relatively low level of the applied voltage further increases the delay time, whereas it is possible that the memristors will not fully switch their states if the input voltages are not applied for a sufficiently long time. In such case it would be difficult to distinguish between the possibly different output voltage levels.

4.3.3 CMOS/Memristor Threshold Logic

Threshold logic provides powerful computational properties beyond Boolean logic and the development of high-performance threshold logic circuits would benefit a number of important applications in computing. The transfer function of a n-input linear threshold gate (LTG) is defined as:

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & \sum_{i=1}^{n} w_i x_i \geq T \\ 0, & else \end{cases} \tag{4.1}$$

where x_i is a Boolean input variable, w_i is the integer weight of the corresponding input i, and T is an integer threshold [39]. A special case of LTG is symmetric LTG with identical weights for different inputs. A n-input symmetric LTG can implement a maximum of n nontrivial Boolean functions $f^{(k)}(x_1, x_2, \ldots, x_n)$ defined by $w_1^{(k)} = w_2^{(k)} = \ldots = w_n^{(k)} = 1/k$ and $T^{(k)} = 1$, where $k = 1, 2, \ldots, n$.

LTG is a powerful universal gate capable of reducing up to 2× on average the gate count for representative benchmark circuits [40]. Various implementations of LTGs have been investigated [41]. In most cases the weights are fixed and cannot be changed in-field; however, the in-field reconfiguration of the LTG's weights is a very desirable feature for most of the applications. The resistance switching property of memristors renders them well-suited for implementing weights in LTGs; memristors enable much better scaling prospects and hence a much denser and more potent LTG implementation compared to those based on floating-gate transistor approaches [15, 25].

In Fig. 4.5 we present a hybrid CMOS/memristor solution for the implementation of a programmable LTG where memristors implement "ratioed" diode-resistor logic, whereas CMOS circuitry is used for signal amplification and inversion [15]. We consider threshold-type memristors with a linear conductance above the switching threshold with a configurable slope (differential conductance) $1/R$, as shown in Fig. 4.5a. We assume that this slope can be configured to any value between $1/R_{\mathrm{ON}}^{\mathrm{L}}$ and $1/R_{\mathrm{ON}}^{\mathrm{H}}$; owing to their analog nature memristors could be

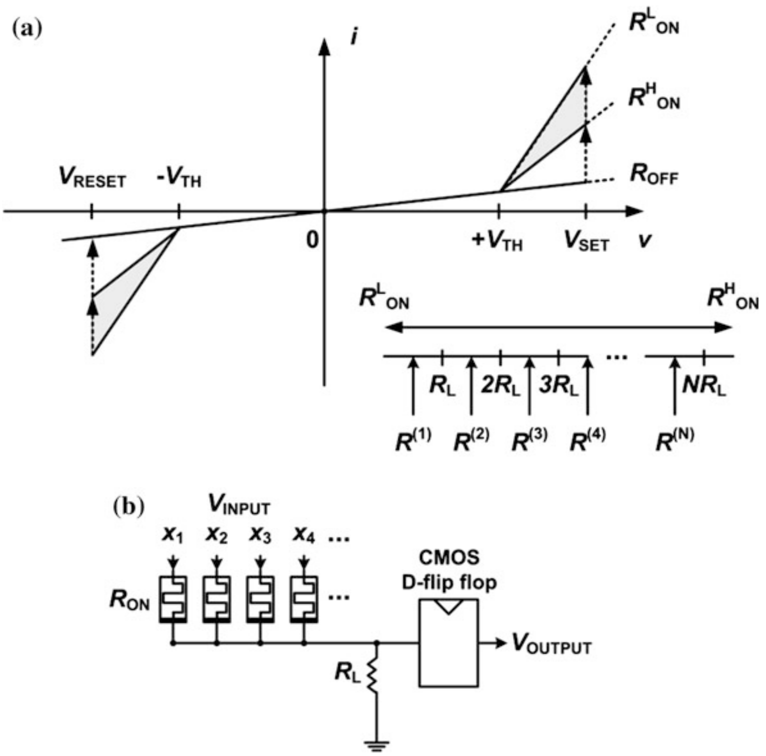


Fig. 4.5 **a** Qualitative *i–v* characteristics of memristors in series with a set of anti-parallel connected diodes. The *shaded area* shows the range of possible intermediate resistive states utilized for the implementation of the weights. **b** Circuit schematic of a LTG implemented with memristors, a load resistor, and a CMOS D flip-flop [15]

(at least theoretically) programmed to any intermediate resistive state between the two boundaries R_{ON} and R_{OFF}. The voltage threshold $|V_{TH}|$ is implemented with two external diodes connected in anti-parallel fashion, in series with the memristor [15].

Figure 4.5b shows the circuit implementation of configurable "ratioed" diode-resistor logic using n memristors connected to a pull-down resistor R_L. Generally, their common connection node is connected to a CMOS component (here a D flip-flop) so that the voltage swing is restored and the output can drive other logic gates. Assuming that the CMOS gate is designed to restore a signal to the supply voltage V_{DD} (logical '1') if the input voltage is larger than $1/2 \times (V_{DD} - V_{TH})$, or otherwise to the ground (logical '0'), the whole circuit implements the LTG defined by Eq. 4.1, where $w_i = 1/R_i$ and $T = 1/R_L$. For symmetric LTG the weights $w^{(k)} = 1/R^{(k)}$ and R_L must satisfy the following condition: $(k - 1) \times R_L < R^{(k)} < k \times R_L$. The inset of Fig. 4.5a shows the possible spectrum of $R^{(k)}$ with respect to R_L. In order to implement all n Boolean functions with a symmetric LTG, the $[R_L, n \times R_L]$ range should fit within a physically permitted range $\left[R^L_{ON}, R^H_{ON}\right]$ therefore $R^H_{ON}/R^L_{ON} = n$. In principle, a simpler design of a symmetric LTG is possible with fixed weights and load resistor implemented with a memristor. However, such implementation is less suitable for circuit integration and cannot be used to implement more general LTGs. Nevertheless, having both R_L and the weights implemented with memristors, may allow flexibility in choosing the optimal value of R_L.

Although being more suitable for artificial neural network applications due to their functional similarity with biological neurons, LTGs have been a popular choice for the implementation of computer arithmetic circuits [42]. Evidently, an important feature of LTGs is that they do not rely on changing the state of memristive devices during the logic operation. However, in memristive LTGs the memristors need to be programmed with very high-precision, which is already a limiting factor for the number of different memory levels that can be achieved, given the property variation among experimentally realizable memristors.

4.4 CMOS-*like* Memristor-Based Logic Circuit Design

In this section we focus on how the composite dynamics of groups of parallel memristors with opposite polarities can be incorporated in a CMOS-*like* circuit design paradigm for the creation of memristive complementary logic circuits [16, 43].

The presented circuit design approach is based on the following CMOS design principles: For every circuit implementation of a logic function $F(x)$, there is a specific formation for the employed field effect transistors (FETs). Figure 4.6 generally shows how appropriately polarized memristors can be used to replace FETs, thus maintaining the well understood CMOS design methodology while delivering similar circuit functionality; i.e. delivering complementary digital logic circuits comprising two-state switching devices. In conventional CMOS the word "complementary" refers to the fact that the typical circuit design uses complementary and symmetrical pairs of p-type and n-type FETs for the implementation of Boolean functions. In the proposed memristor-based design scheme the word "complementary" similarly implies the use of complementary and symmetrical pairs of devices; only in this case complementation refers to the use of the same kind of devices (memristors) but with opposite polarization.

Consequently, assuming that memristors work as two-state resistive switches, the overall circuit functionality remains similar to that of CMOS logic circuits. More specifically, as shown in Fig. 4.6, we use forward polarized memristors (FPMs) in the n-MOS area and reversely polarized memristors (RPMs) in the p-MOS area. The input logic signals are represented using positive voltage for logic '1' and negative voltage (since we use bipolar memristors) for logic '0'. The RPMs (corresponding to p-type FETs) are initially found (programmed) in the R_{OFF} state. Thus a negative applied voltage changes their state from R_{OFF} to R_{ON}, i.e. closes the switch likewise happens with p-type FETs. On the contrary, a positive voltage either restores their state from R_{ON} to R_{OFF} or lets them unchanged in R_{OFF}, i.e. the

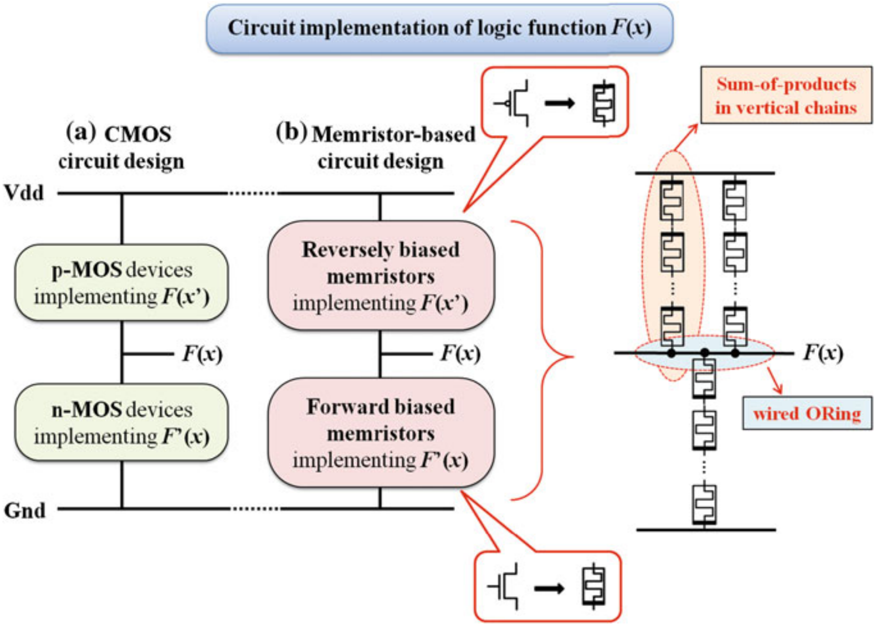

Circuit implementation of logic function F(x)
(a) CMOS circuit design
(b) Memristor-based circuit design
Vdd
Gnd
p-MOS devices implementing F(x')
n-MOS devices implementing F'(x)
F(x)
Reversely biased memristors implementing F(x')
Forward biased memristors implementing F'(x)
F(x)
Sum-of-products in vertical chains
wired ORing
F(x)

Fig. 4.6 **a** General demonstration of the conventional CMOS circuit implementation methodology for a Boolean logic function $F(x)$, compared to **b** the CMOS-*like* design concept comprising appropriately polarized resistance-switching devices (memristors) which replace conventional field effect transistors (FETs)

switch is open. As far as FPMs (corresponding to n-type FETs) are concerned, they are initially found (programmed) in the R_{ON} state. So, a negative applied voltage changes their state from R_{ON} to R_{OFF}, though a positive voltage either restores their state from R_{OFF} to R_{ON} or lets them unchanged in R_{ON}. In other words, positive (negative) voltage turns the switches on (off), similar to how n-type FETs work for the corresponding input signals.

Similar to standard CMOS, any implemented Boolean logic function is determined by the topology of the circuit which consequently consists of an equivalent ohmic resistance for the upper and another one for the lower part of the CMOS-*like* design. Therefore, the output voltage V_{OUT} is always a fraction of the supply voltage (reading voltage − V_{DD}), being dependent on the voltage divider across the two parts of the memristor-based circuit; output voltage values close to V_{DD} correspond to logic ‘1’ and values close to zero (GND) correspond to logic ‘0’. Figure 4.7 demonstrates the CMOS-*like* circuit implementation of the universal digital logic gate-set with memristors. Their general functionality is explained taking as example the NOT gate: Initially (considering {FPM, RPM} set in {R_{ON}, R_{OFF}}) before any applied input, we get $V_{OUT} = V_{DD} \times R_{ON}/(R_{ON} + R_{OFF}) \ll V_{DD}$, since $R_{OFF} \gg R_{ON}$. After a positive applied input voltage pulse (logic ‘1’) the memristors maintain their states, thus we get the same V_{OUT}. However, with a

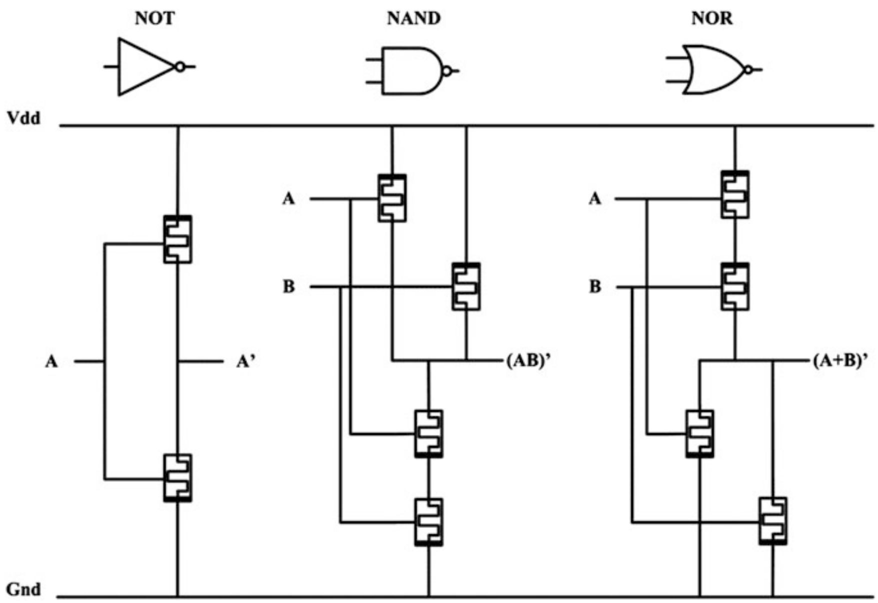


Fig. 4.7 CMOS-*like* nanoscale circuit design of the universal digital logic gate-set utilizing FPMs and RPMs instead of CMOS transistors

negative applied input voltage (logic ‘0’) the state of the two anti-parallel memristors changes, so we get $V_{OUT} = V_{DD} \times R_{OFF}/(R_{ON} + R_{OFF}) \approx V_{DD}$. Afterwards, a positive input applied to the NOT gate will restore the memristors to their initial states. Consequently, V_{OUT} changes as expected for any input variation. Likewise, the proposed paradigm works correctly for the rest of the universal digital logic gates, making the design of every digital logic circuit possible. The notion of collective memristive dynamics is particularly invoked in how input logic signals are simultaneously applied to the appropriate pairs of FPMs and RPMs, either changing or letting unaffected the circuit output.

Overall, the memristive CMOS-*like* circuit design methodology comprises a few straightforward steps to follow, ranging from the definition of a circuit to its final implementation. These steps are listed below:

1. Definition of the inputs and outputs of the circuit and assignment of a logic variable to each of them;
2. Extraction of the corresponding Boolean logic functions describing the outputs of the circuit;
3. For every logic function, e.g. $F_i(x_1, x_2, \ldots, x_n)$, calculation of its complement $F'_i(x_1, x_2, \ldots, x_n)$ and of the expression $F_i(x'_1, x'_2, \ldots, x'_n)$;
4. Design of the logic circuit according to the CMOS-*like* paradigm (Fig. 4.6).

For sum-of-products representations each product term is implemented with a vertical chain of memristors and the final sum is created by wired-ORing the existing products. The listed steps form a complete methodology which should provide any circuit designer with the ability to efficiently design and implement memristive combinational circuits.

It should be mentioned that in Fig. 4.7 the memristors were deliberately shown as three-terminal devices to underline similarities with the CMOS counterparts. In fact, memristors are two-terminal devices and application of an input signal corresponds to the application of an appropriate voltage (positive or negative) to the terminals of the target memristor. Hence, certain circuit modifications are required in order to be able to exploit the favorable memristive properties and to overcome particular circuit design limitations. The latter involve the sequential processing of the input logic signals, the two-terminal nature of memristors (compared to the three-terminal FETs), and the requirement for proper isolation of the groups of memristors where input voltages are simultaneously applied. In the following section we discuss the aforementioned issues and show how the CMOS-*like* memristive circuits can be appropriately mapped onto a nano-CMOS hybrid crossbar geometry.

4.4.1 Implementation in Hybrid Nano-CMOS Memristive Crossbar

In the recent literature, large groups of interconnected memristors have been primarily studied in the crossbar geometry. The crossbar is probably so far the most well-known and well-documented nanoelectronic architecture, offering several benefits which include pattern regularity, defect-tolerance, and the highest possible device density [13, 44–48].

It comprises two parallel planes separated by a thin chemical layer. Each plane contains a set of parallel and uniformly-spaced nanowires which are perpendicular to wires in the other. The region where two wires cross is a junction which may either be configured as an electronic device or left unconfigured so that the two crossing wires do not interact electrically. Owing to their two-terminal structure, memristors can be integrated into crossbar networks with a memristor at each cross-point. Their electrical configuration may be done by placing different voltages on the horizontal and vertical nanowires that define the target junction. However, the pure memristive crossbar suffers from leakage current paths, generally referred as the sneak path problem [49, 50].

Hybrid crossbars, where the cross-points comprise a select transistor in series to a memristor, were proposed in order to control the location of the memristor to be accessed, thus overcoming the sneak-path problem and ensuring reliable circuit operation [3, 7]. In such implementations the select transistors were either integrated along with memristors on the same substrate or the memristor array was integrated directly on top of underlying CMOS circuitry. Therefore, an elaborated hybrid crossbar combining memristive cross-points with access CMOS transistors constitutes a convenient target-architecture for the implementation of CMOS-*like* memristive circuits.

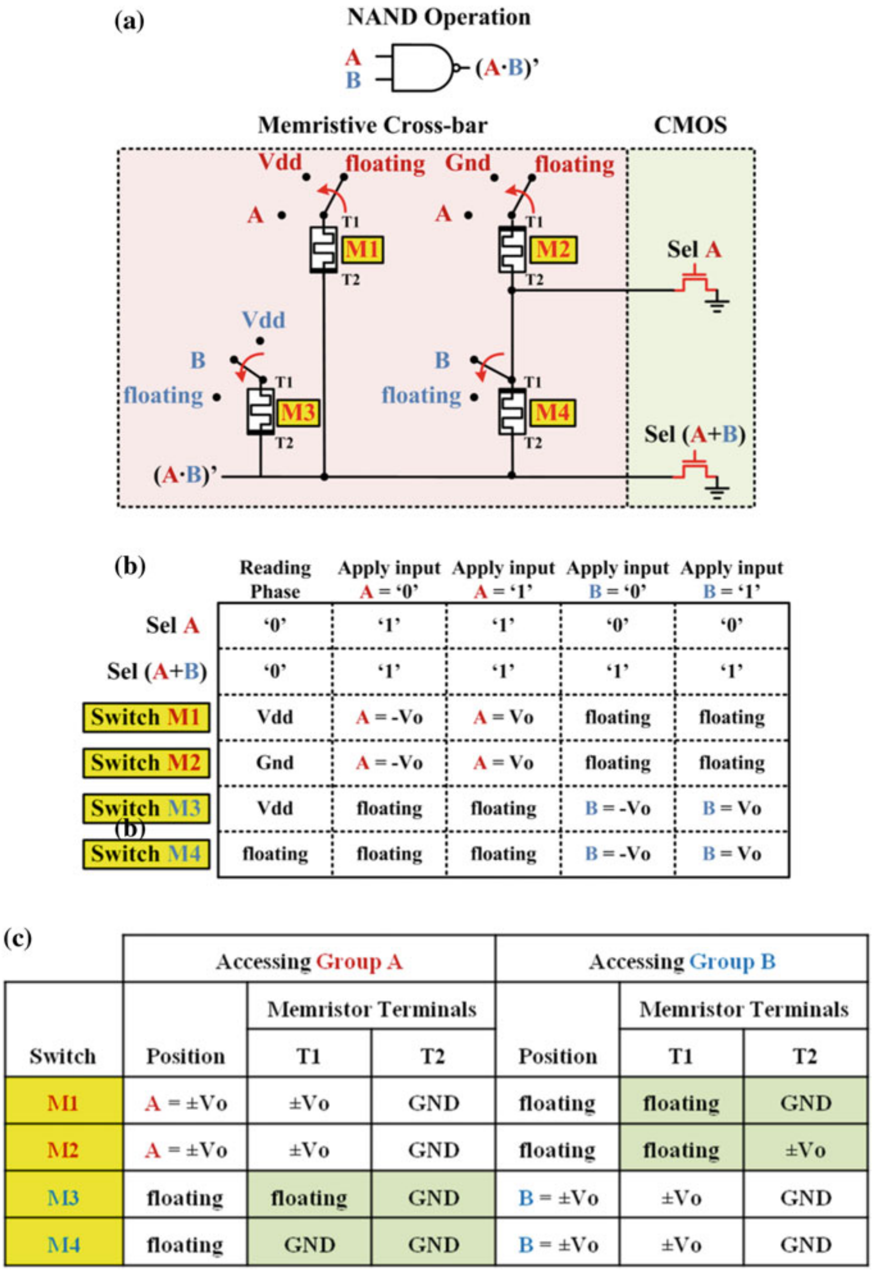
To this end, the geometry of the memristive combinational circuits should first be appropriately altered in order to better fit in an array-like structure like the crossbar. The FPMs and RPMs which receive the same input signal are placed in the same horizontal line and the output is always taken from the lower part of the circuit. As an example we show in Fig. 4.8a the CMOS-*like* memristive design of a 2-input universal NAND logic gate. This circuit consists of both memristors (crossbar plane) and auxiliary transistors (CMOS plane) which are driven by appropriate selection lines in the peripheral/driving circuitry. The latter facilitate correct access operation to multiple memristors where the same input signal is applied. Also, there are four switches, named M1 through M4, which determine the logic gate function while providing the following operation options:

1. Application of input (i.e. programming) signals which affect the internal state of memristors;
2. Reading the circuit output by applying V_{DD} and GND voltages;
3. Idle operation with all the switches set in floating position.

The duration and amplitude of the input voltage pulses, here denoted as $\pm \mathrm{V}_{\mathrm{o}}$, should be selected so as to exceed the V_{SET} and V_{RESET} thresholds and be longer than the corresponding switching time of the devices; thus cause them to completely switch in either direction (i.e. from R_{OFF} to R_{ON} or vice versa). Indeed, besides the continuously accumulating knowledge on resistive switching phenomena ever since the first memristor implementation in 2008, the switching thresholds in experimental memristive devices still demonstrate a small deviation around a nominal value. Therefore, in order to guarantee stable and correct operation, optimal programming and reading voltage pulses have to be selected based on the threshold-type switching characteristics of the individual devices. The V_{DD} amplitude should be selected so that the corresponding voltage drop never exceeds the voltage threshold of any of the invoked memristors. In this way the internal state of memristors remains unaffected during read-out regardless of the current flow from V_{DD} to GND.

Moreover, with their memory function being nonvolatile, memristors do not require power to refresh their states, even if the chip power is turned off, i.e. when set in idle operation. Generally, in any CMOS-*like* circuit each of the input signals will apply to at least a pair of anti-parallel (i.e. FPM || RPM) memristors. The exact number of FPMs and RPMs cannot be formerly defined since it depends on the complexity of the circuit and on the logic function which describes the output; a specific Boolean logic function has many equivalent circuit implementations, though array-based structures like the crossbar can accommodate sum-of-products logic representations (see Fig. 4.6).

In Fig. 4.8b we provide information regarding the positions of the switches and the logic values applied to the access transistors for every possible phase of the circuit operation. Similarly, in Fig. 4.8c we summarize the operational phases of the circuit while denoting the resulting voltage values at the respective terminals of the memristors while updating the circuit inputs. The indicated (highlighted green)



	Reading Phase	Apply input A = '0'	Apply input A = '1'	Apply input B = '0'	Apply input B = '1'
Sel A	'0'	'1'	'1'	'0'	'0'
Sel (A+B)	'0'	'1'	'1'	'1'	'1'
Switch M1	Vdd	A = -Vo	A = Vo	floating	floating
Switch M2	Gnd	A = -Vo	A = Vo	floating	floating
Switch M3	Vdd	floating	floating	B = -Vo	B = Vo
Switch M4	floating	floating	floating	B = -Vo	B = Vo

	Accessing Group A			Accessing Group B		
		Memristor Terminals			Memristor Terminals	
Switch	Position	T1	T2	Position	T1	T2
M1	A = ±Vo	±Vo	GND	floating	floating	GND
M2	A = ±Vo	±Vo	GND	floating	floating	±Vo
M3	floating	floating	GND	B = ±Vo	±Vo	GND
M4	floating	GND	GND	B = ±Vo	±Vo	GND

Fig. 4.8 a Analytical example of a CMOS-*like* implementation of a two-input NAND logic gate. **b** A summary of all appropriate voltages (switch positions) needed to be applied to the corresponding devices during all possible circuit operation phases. **c** Resulting voltage values at the terminals of the memristors during circuit operation

voltage values at the terminals of the memristors which are not addressed during each access operation, underline that there are no main leakage paths in such memristor-based circuits.

Additionally, in order to facilitate understanding of the applied voltage sequence, in Fig. 4.9 we provide a descriptive flow chart where all operation details of the proposed memristive logic circuits are summarized. Given the appropriate driving circuitry, V_{DD} and GND voltages are supposed to be applied only in order to read the circuit output.

Furthermore, it can be observed that the total number of access transistors is smaller than the number of used memristors. Therefore, having one transistor for

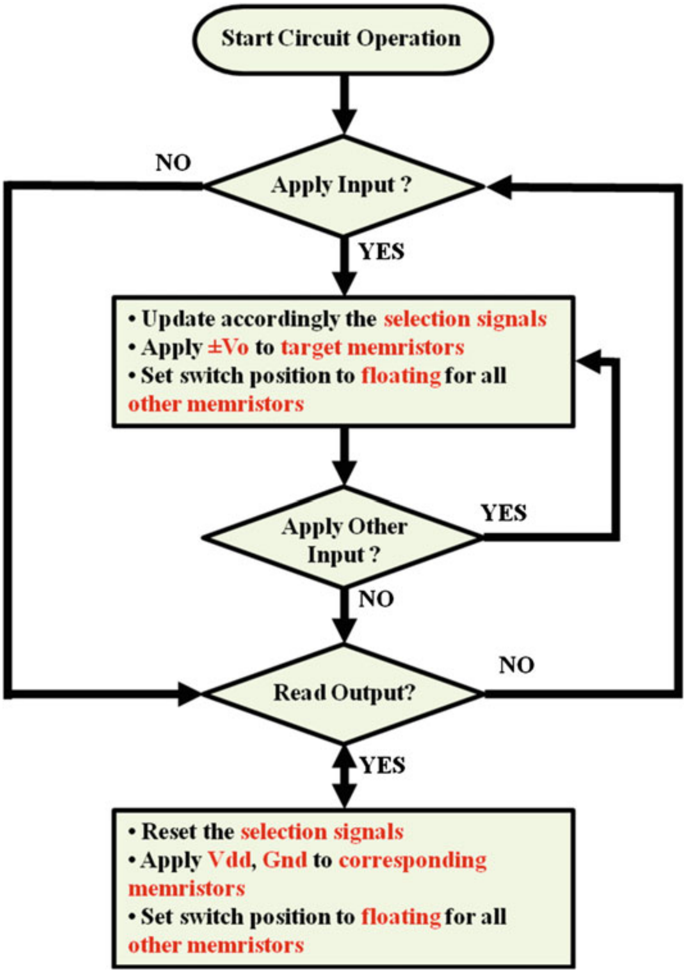

Start Circuit Operation
Apply Input ?
NO
YES
• Update accordingly the selection signals
• Apply ±Vo to target memristors
• Set switch position to floating for all other memristors
Apply Other Input ?
YES
NO
Read Output?
NO
YES
• Reset the selection signals
• Apply Vdd, Gnd to corresponding memristors
• Set switch position to floating for all other memristors

Fig. 4.9 A flow chart explaining the applied signal sequence and all operation details of the proposed CMOS-*like* memristive implementation of digital logic circuits

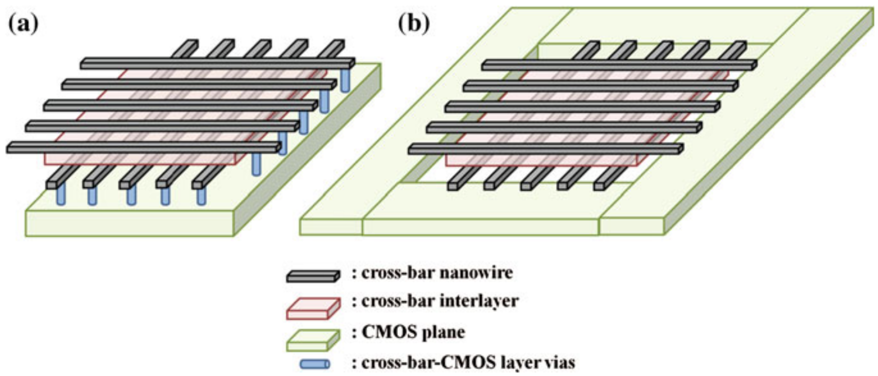


Fig. 4.10 Visualization of the implementation concept for the proposed hybrid crossbar architecture. **a** Memristors are fabricated directly above the CMOS plane. **b** The driving CMOS components are integrated on the same substrate with the memristive crossbar

every memristor results redundant for the CMOS-*like* circuit architecture; indeed, in all circuit branches the devices immediately connected to the output line can be accessed using a single transistor. This explains the notation e.g. $Sel(A + B)$ which refers to a transistor used for both input signals A and B. Consequently, the necessary transistors need not be at every cross-point of the crossbar but alternatively can be found in the CMOS driving circuitry domain. Regarding the transistor biasing scheme, their gate-voltage should never rise above the source-voltage on any of the unaddressed memristors. Moreover, for every single input signal change, e.g. for an input sequence from AB = “01” to AB = “11”, the circuit output is updated in only a single step, whereas for multiple signal changes the output response delay is multiplied by the number of the changed input signals. Pulsing details provided in Figs. 4.8b and 4.9 infer that it is not possible to have more than one signal applied simultaneously.

The development of such hybrid crossbar architecture adds no significant complexity to similar architectures found in the literature and it is feasible with today’s fabrication technologies. As mentioned before, there is the option of integrating the driving CMOS components on the same substrate with the memristive crossbar. On the other hand, memristors could be fabricated directly above the CMOS plane with an array of vias providing electrical connectivity between the CMOS and the memristor layers [29]. The visualization of such implementation concepts is shown in Fig. 4.10.

4.4.2 Verification Using SPICE

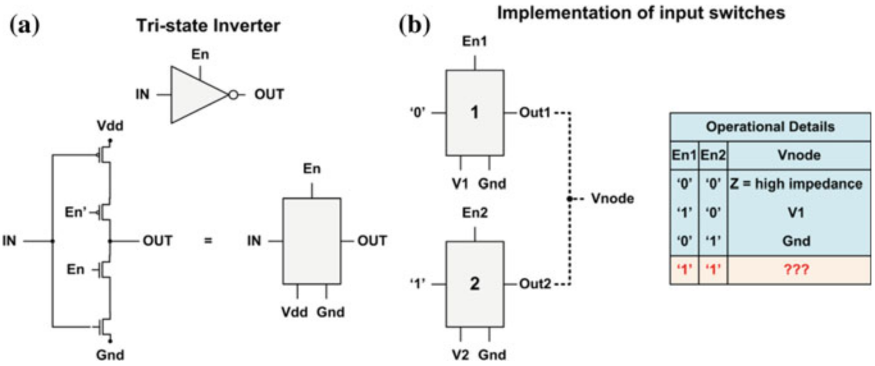
This section presents a SPICE-level simulation-based validation of the CMOS-*like* circuit design methodology using the Cadence PSPICE simulation environment.

Simulations are based on the threshold-type memristor model which was presented in Chap. 2 [51, 52].

4.4.2.1 Driving Circuitry

As mentioned previously, both positive and negative input voltages are required to program the memristors in CMOS-*like* logic circuits. Thus, the access transistors should permit control of power flow in both directions. It is worthwhile mentioning that, although FETs will conduct equally in both directions when they are turned "on," when they are turned "off" they will still conduct in the reverse direction; this is an interesting consequence of the body-source connection typically attributed to the pn junction formed between the body (p) and the drain (n) (for n-channel). So if we apply a positive voltage to the drain and a negative voltage to the source, when the FET is "on" we see current flowing with very little voltage. However, when the FET is "off," current still flows and the voltage is ≈0.7 V (the diode threshold). Power FETs generally have their source connected to their body. Therefore, in our circuit designs we used two FETs connected back-to-back (sources connected together) instead of only one for every select device. The drain-source resistance exhibited by the transistors when in "off" state was set much higher than the maximum resistance of memristors (R_{OFF}) in order to achieve better isolation of the unaddressed devices during operation.

Moreover, in order to create all necessary input signals (i.e. programming voltages, read voltages, and high impedance) we implemented the input switches utilizing tri-state inverters. A tri-state inverter is a useful device that allows controlling when current flows through the device and when it does not. Figure 4.11a shows the corresponding schematic symbol. The device has two inputs: a data input IN and a control input En (enable). The control input acts like a valve; when active,



Operational Details		
En1	En2	Vnode
'0'	'0'	Z = high impedance
'1'	'0'	V1
'0'	'1'	Gnd
'1'	'1'	???

Fig. 4.11 **a** Circuit schematic symbol and CMOS implementation of a tri-state inverter, along with the corresponding equivalent block diagram defined in SPICE. **b** Multi-state input switch implementation example using a pair of tri-state inverters sharing a common output node

the output is the inversed input, otherwise the output is “Z”, i.e. no current flows through it regardless of the data input (floating node). Figure 4.11a also shows an explicit CMOS implementation of a tri-state inverter using a typical NOT logic gate enhanced with two additional transistors where the control signal applies. In SPICE we represent such circuit using the equivalent block diagram of Fig. 4.11a for simplicity.

Using combinations of tri-state inverters allows the creation of multi-state switches which we use in order to apply the programming/read voltages to the memristors. An example of this technique is shown in Fig. 4.11b. Two tri-state inverters are utilized in order to create a set of three possible outputs on a particular shared node, namely V1, GND, or Z. All inputs receive fixed values except for the control inputs En1 and En2 which may take any of the following possible logic combinations “En1 En2” = {“00”, “01”, “10”, “11”}. Operational details for this multi-state input switch are summarized in the corresponding table in Fig. 4.11b. In this example the last specified combination of the control signals will probably result in an indeterminable output (conflicting voltages), hence it is not used. In the same fashion, we create all of the required inputs just by applying fixed inputs of appropriate value to the tri-state driving components.

4.4.2.2 Circuit Simulation

As a proof of concept we show in Fig. 4.12 the SPICE-based design and simulation of a CMOS-*like* NOR logic gate.

In Fig. 4.12a, b we give the NOR gate circuit and the corresponding schematic in the SPICE environment, whereas Fig. 4.12c presents the simulation results. For readability reasons we focus on the programming/read pulses and avoid showing the selection signals. The lower graph of Fig. 4.12c shows the input signals, where the line-colors visually correspond to the colors of the four SPICE voltmeters shown in Fig. 4.12b.

Values of the parameters of the model are common for all devices and are set as $\{a_x, b, c, m, f_0, L_0, V_{SET}, V_{RESET}\} = \{1 \times 10^5, 0, 0.1, 82, 310, 5, 1.5\text{ V}, -1.5\text{ V}\}$, whereas the memristance ratio is $R_{OFF}/R_{ON} \approx 10^2$ with $R_{OFF} \approx 200\text{ k}\Omega$ and $R_{ON} \approx 2\text{ k}\Omega$. Such value-set corresponds to a memristor which switches steeply as soon as the applied voltage exceeds either of its thresholds; otherwise its state remains unaffected (because $b = 0$) [51]. Memristance ratio is defined to be of two orders of magnitude so as to facilitate the better distinction between the two boundary memristive states and, consequently, the distinction of the binary output voltage values of the simulated circuits. All assumptions regarding the switching thresholds and the programming voltages have been made only in the context of our simulations and do not relate to any experimentally manufactured and characterized device.

Each of the input signals is sequentially set to logic ‘1’ and logic ‘0’ to create all valid input combinations before returning to the initial combination. This is done purposely to demonstrate that the circuit always recovers successfully its state. The corresponding output is observed between consecutive input signal transitions; V_{DD}

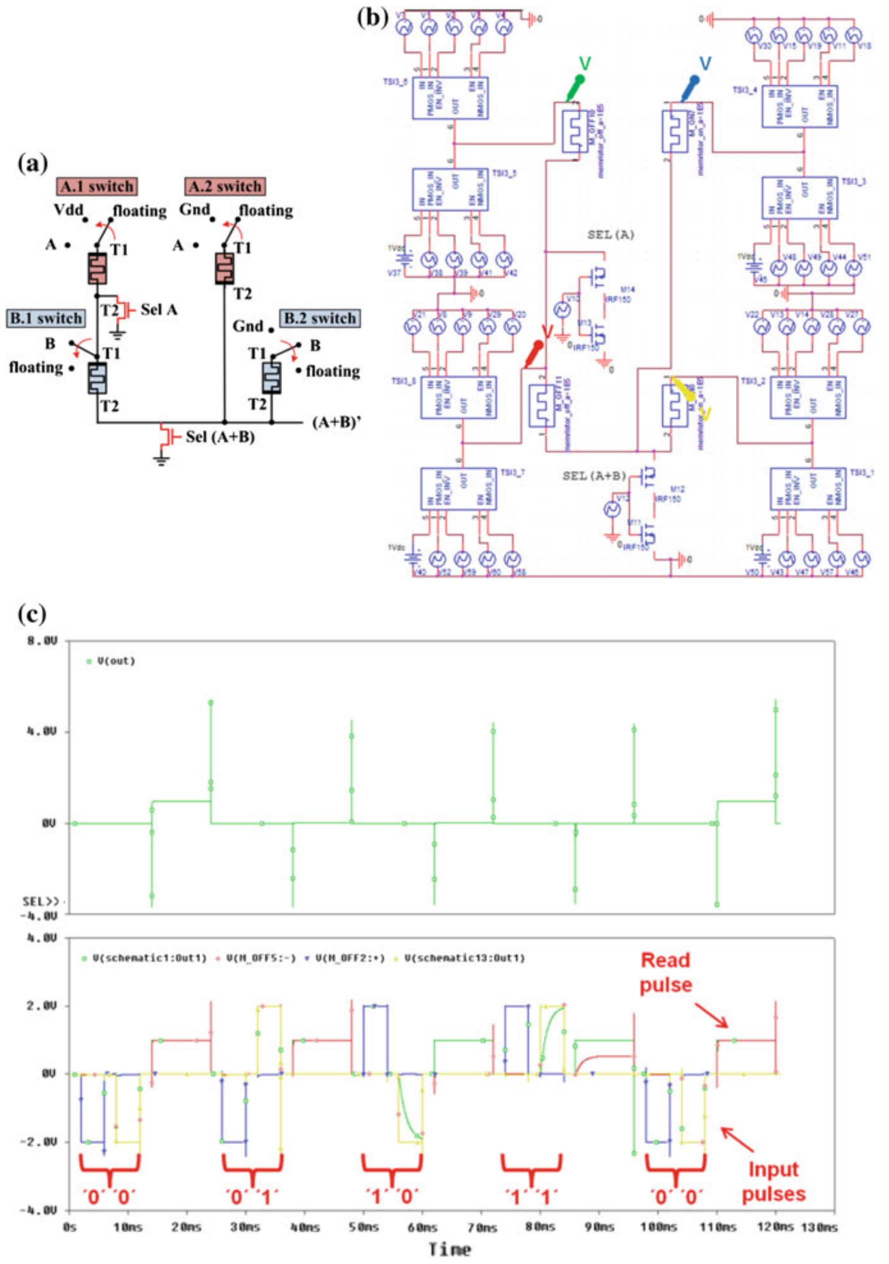


Fig. 4.12 **a** A two-input CMOS-*like* memristive NOR logic gate circuit and **b** the corresponding schematic in SPICE. **c** Simulation results: circuit output (*upper graph*) for all possible input combinations (*lower graph*)

and GND are applied only after every access operation to read the circuit output. They are, however, kept below the threshold voltages to avoid affecting the internal state of the memristors during the reading phase.

The simulation begins with the input logic combination “00”. The two corresponding negative applied pulses set RPMs and FPMs to R_{ON} and R_{OFF}, respectively. Therefore, the equivalent resistance of the upper circuit branch (RPM area) results 2 × R_{ON} (i.e. $R_{ON} + R_{ON}$) whereas for the lower branches (FPM area) it is $R_{OFF}/2$ (i.e. $R_{OFF} \parallel R_{OFF}$). Hence when V_{DD} is applied there is a much higher voltage drop on the lower resistive part of the circuit (high output value) compared to that of the higher part, attributed to the high R_{OFF}/R_{ON} ratio. However, when a positive pulse is applied to either of the inputs, the involved RPM changes its state from R_{ON} to R_{OFF}. Thus, due to the series connection of the RPMs, the equivalent resistance of the upper branch results higher and, as a consequence, the corresponding voltage-drop on the lower part of the voltage divider results significantly lower (low output value).

As far as the pulsing characteristics are concerned, we use 4 ms-wide programming pulses of ±2 V and 10 ms-wide read pulses of 1 V. Before every subsequent voltage pulse we include a 1 ms-wide idle period for clarity, when all input switches are set floating. The aforementioned programming protocol was figured out after experimentation with the memristor model, taking into account the minimum switching time of the devices, due to the given values to its parameters. Regarding the frequency of the input and output signals, there is no particular relation between them; the output signal duration depends only on the time when V_{DD} and GND are applied. The upper graph of Fig. 4.12c shows the circuit output which changes as expected for all input combinations.

4.4.3 Application in Larger Combinational Circuits

4.4.3.1 Memristive CMOS-like Circuit Simulator

In order to facilitate the effective simulation and study of complex CMOS-*like* memristive logic circuits, we developed a compact simulator which comprises the practical graphical user interface (GUI) shown in Fig. 4.13. The simulator was built using the Easy Java Simulations (EJS) code generation environment [53] and incorporates the threshold-type memristor device model presented in Chap. 2 [43, 51].

Its user-friendly interface comprises a configuration panel (left) and an output panel (right). The left panel facilitates the definition of the location and the orientation of the memristors which are invoked in the CMOS-*like* circuits through a crossbar-based logic tile. Such tile consists of interconnected two-dimensional arrays which either comprise memristive nodes or configurable routing junctions; the latter facilitate the propagation of internal signals to the outside. The horizontal and vertical lines represent nanowires which are distributed into different quadrants of adjustable

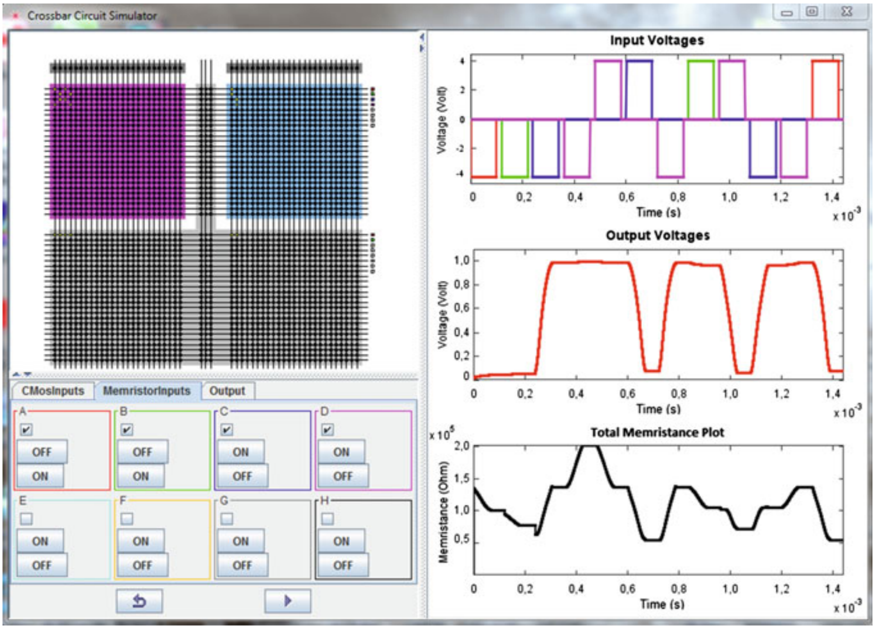


Fig. 4.13 GUI-based simulation environment for CMOS-*like* memristive logic circuits

dimensions to facilitate the design and simulation of rather complex circuits. Each quadrant is considered to possess different electrical properties due to the chemical properties of the interlayer used in that region. The overall concept of the logic tile is based on the work of Snider et al. [27] who proposed an approach to build nanoscale computing elements on mosaics of complementary (n-FET and p-FET) crossbar arrays and configurable switches. The simulator uses the same sharp array-based geometry while using appropriately polarized memristors instead of FETs.

The logic tile is illustrated in more detail in Fig. 4.14. It consists of quadrants with configurable routing switches (the lower gray colored ones) and others where a junction can be configured to be either an RPM (upper left pink quadrant) or an FPM (upper right blue quadrant). A particular input signal may be brought in on any of the upper horizontal nanowires and it applies to all the configured memristors located in the same horizontal line, whereas an output signal may be driven out on any of the horizontal nanowires of the bottom quadrants, on either side of the array. The simulator assumes the compact three-terminal representation for memristors, thus omitting the access transistors without loss of generality. According to the concept for the physical implementation of such representation, as discussed before, the third terminal roughly corresponds to the gate of a supplementary access transistor. V_{DD} and ground (GND) signals are taken from the dark gray quadrants located at the top and are used to read the circuit output. The small circles generally represent available connections between the two different crossbar planes and can be either configured (yellow) or left unconfigured (black).

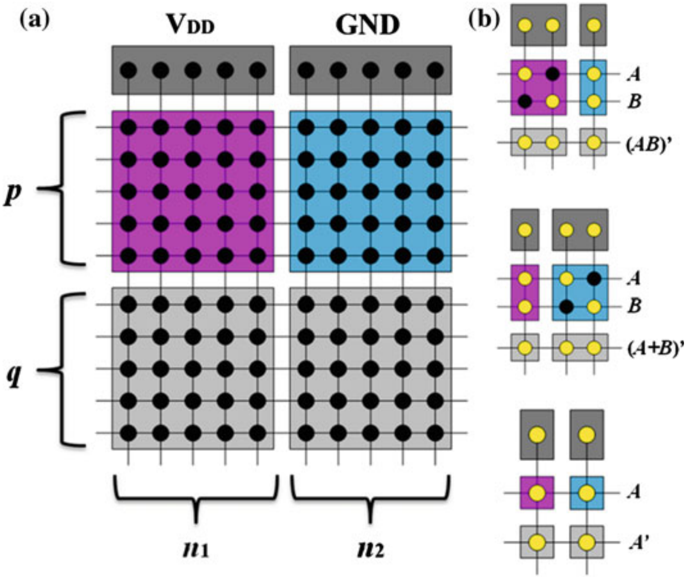


Fig. 4.14 **a** The basic logic tile which comprises V_{DD} and GND supply, $p(q)$ horizontal nanowires for inputs (outputs), and $n = n1 + n2$ vertical nanowires. **b** Tile configuration for the set of universal digital logic gates; *yellow dots* denote currently configured memristive and routing junctions

For any Boolean logic function, we first decompose it in two sets of minterms (one for the RPMs and another for the FPMs) and then we implement it on the logic tile by selectively configuring the junctions in each quadrant. For sum-of-products function representations, each product term is implemented with a single vertical chain of memristors and the final sum is created by connecting the created products to an output nanowire. Whenever necessary, a circuit may be presented with two wires for every input variable with the additional wires representing the inputs' complements; the complemented signals are considered readily produced by external circuitry.

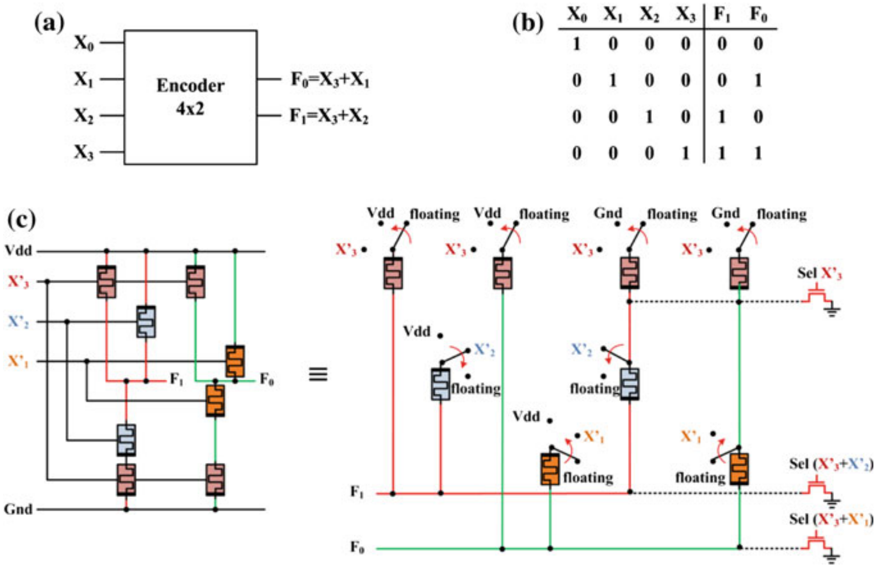
Overall, the simulator facilitates studying qualitatively the behavior of CMOS-*like* memristive logic circuits and observing the trends while experimenting with different values for the parameters of the model and for the set of applied programming/reading voltages. Programming voltage types may be sinusoidal, triangular, or rectangular. As shown in Fig. 4.13, on the right output panel the simulator provides three particular graphs representing the input/output voltages on the selected horizontal nanowires, and the time-evolution of the total equivalent memristance of the designed circuit; the latter could be used for the rough estimation of power consumption. Using the developed environment we have successfully applied the proposed methodology to design and simulate several memristive combinational circuits which are presented in the following sections.

4.4.3.2 CMOS-like Memristive Digital Encoders/Decoders

Encoders are commonly used multi-input combinational circuits which produce an encoded output according to the signals in their input lines. Generally, an n-bit digital encoder has 2^n input lines and n output lines. The output lines generate the binary equivalent of the input line whose value is at the moment set to logic ‘1’. In order to describe the circuit, every output is first represented by a logic function of the input variables. Then, each logic function can be implemented following the CMOS-*like* paradigm.

In Fig. 4.15 we show the design of a 4 × 2 digital encoder. More specifically, Fig. 4.15a shows the general block diagram where inputs and outputs are assigned a variable name and the outputs are described by logic functions. The input variables are defined as X_3, X_2, X_1, and X_0, whereas the output variables are F_1 and F_0. The corresponding truth table is given in Fig. 4.15b. It includes the four cases which guarantee a correct input to the circuit, excluding the case when all inputs are set to logic ‘0’. The latter means that both outputs are logic ‘0’, which is equivalent to having the input X_0 set to logic ‘1’.

The truth table is used to extract the logic functions for the outputs of the circuit, whose corresponding logic expressions $F'_i(x)$ and $F_i(x')$ are both functions of only the complements of the input variables. Thus, only the complement inputs appear in the circuit schematics in Fig. 4.15c. Both the compact (i.e. more abstract) and the analytical schematic versions are presented. The aforementioned logic expressions used to implement the circuit do not include variable X_0 since the latter does not



X₀	X₁	X₂	X₃	F₁	F₀
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Fig. 4.15 **a** Block diagram of a 4 × 2 encoder, **b** its truth table, and **c** the corresponding compact and analytical CMOS-*like* circuit design. Memristors associated with the same input signals are colored correspondingly

affect the circuit output. This is why this input is omitted in the circuit schematic. Memristors receiving the same input signal are colored correspondingly. Also, the interconnection lines are similarly colored in order to facilitate visual correspondence with the circuit definition in the logic tile of the simulator, demonstrated in Fig. 4.16a; the crossbar junctions are selectively configured (yellow dots) in order to appropriately map the circuit on the array-like architecture.

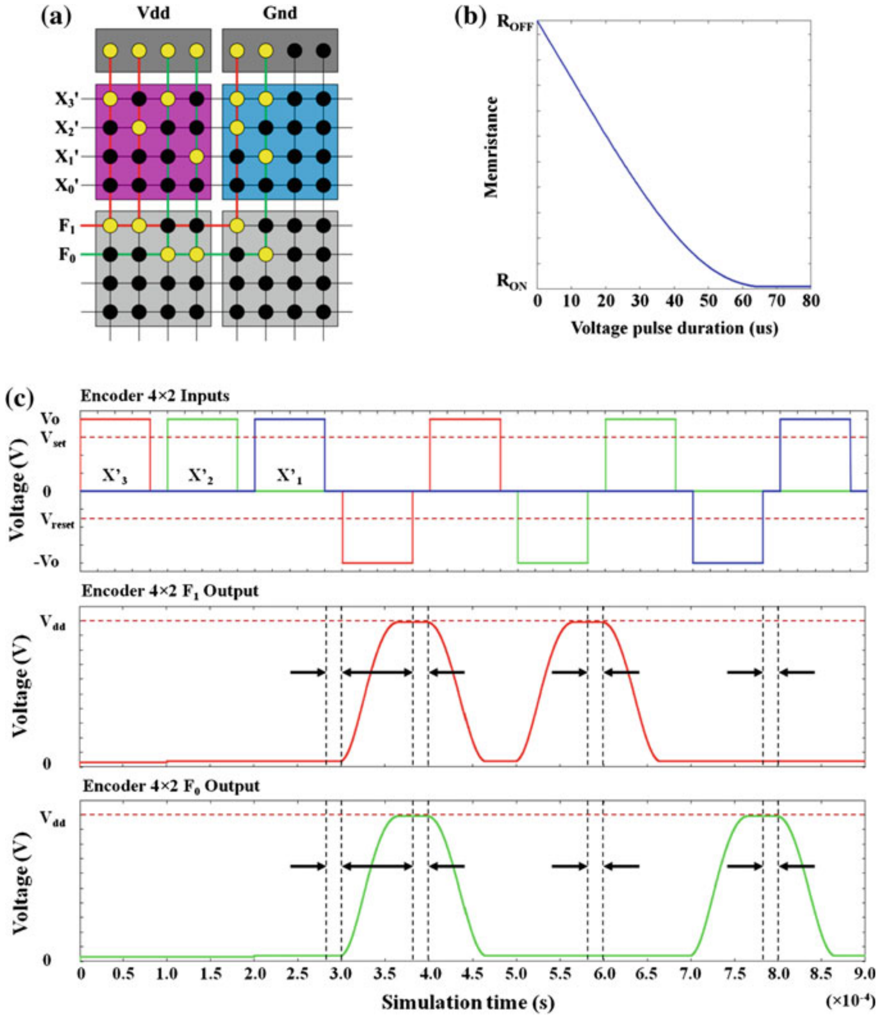


Fig. 4.16 Simulation results for a 4 × 2 digital encoder. **a** The corresponding configuration of the simulator's logic tile. **b** The memristance-change over time for a memristor under a fixed-value programming pulse. **c** The output response of F_1 (*red*) and F_0 (*green*), for all valid input variations of the signals X'_3 (*red*), X'_2 (*green*) and X'_1 (*blue*). The applied voltages are appropriately selected to exceed the threshold values V_{RESET} and V_{SET}. The time gap between the input signal transitions, where the circuit outputs should be read, is highlighted using the *vertical dotted-lines*

In simulation we use the following set of values for the parameters of the model $\{a_x, b, c, m, f_o, L_o, V_{SET}, V_{RESET}\} = \{5 \times 10^6, 0, 0.1, 82, 310, 5, 3\text{ V}, -1.5\text{ V}\}$, and we consider a R_{OFF}/R_{ON} ratio of two orders of magnitude with $R_{OFF} \approx 200\text{ k}\Omega$ and $R_{ON} \approx 2\text{ k}\Omega$. Based on the *i–v* characteristic of individual memristors and on the demonstrated threshold values, we apply read pulses of $V_{DD} = 1$ V and input (write) pulses of $V_o = \pm 4$ V. V_{DD} is kept below the threshold voltages to avoid affecting the state of the memristors during reading the circuit output. The memristance change of a single memristor induced by an input pulse with time is shown in Fig. 4.16b. It is observed that the memristance change is completed after almost 60 µs. Therefore, in simulation we apply 80 µs-wide input pulses to guarantee complete state-transition and hence facilitate the better distinction of the binary values during the reading phase. The simulator applies reading pulses at each simulation step in order to better monitor the changes of the memristor states and the behavior of the circuits over time.

In Fig. 4.16c we show the simulation results of the 4 × 2 digital encoder of Fig. 4.15. The simulation begins with the logic combination $X_3X_2X_1$ = “000” (only the inverted signals are shown). The three initial input pulses maintain RPMs and FPMs unaffected at R_{OFF} and R_{ON} states, respectively. Therefore, according to Fig. 4.15c, the equivalent resistance of the upper circuit branches (RPM area) results $R_{OFF}/2$ (i.e. $R_{OFF} \parallel R_{OFF}$) whereas for the lower branch (FPM area) it is $2 \times R_{ON}$ (i.e. $R_{ON} + R_{ON}$). Hence when V_{DD} applies there is a much higher voltage drop on the upper resistive part of the circuit compared to that of the lower part. This is why both F_1 and F_0 output voltage levels are very low (logic ‘0’). However, when a negative pulse is applied to X_3', the involved memristors change their states; the equivalent resistance of the upper branches is $\approx R_{ON}$ (i.e. $R_{OFF} \parallel R_{ON}$) and for the lower branch it is $\approx R_{OFF}$ (i.e. $R_{ON} + R_{OFF}$). As a consequence, the corresponding voltage drop on the lower part of the voltage divider is now higher. Since this input signal is involved in the equations of both F_1 and F_0, the corresponding outputs are both close to V_{DD} (logic ‘1’) as expected. Similarly, since signal X_2' affects only F_1 and signal X_1' is involved only in F_0, a particular change in either of them induces a corresponding change only to the affected output.

During simulation, every input signal is sequentially set to logic ‘1’ and ‘0’ to finally create all valid input combinations. The corresponding binary output code can be observed in the deliberately left time gap between consecutive input signal transitions (marked between the vertical black dotted-lines). Both outputs are kept at logic ‘0’ when all input signals are set at logic ‘0’.

Digital decoders are the encoders’ counterparts which activate only one of their outputs by presenting logic ‘1’ to the corresponding wire according to the applied encoded input combination. A circuit with n inputs normally has 2^n outputs, i.e. one for every possible input set. Every output is described by a logic function of the input variables. Figure 4.17 summarizes the design methodology for a 2 × 4 digital decoder. It includes the block diagram along with the truth table and the corresponding compact circuit schematic; the more analytic version of the circuit follows by analogy from the previously presented example(s). Both the devices and the

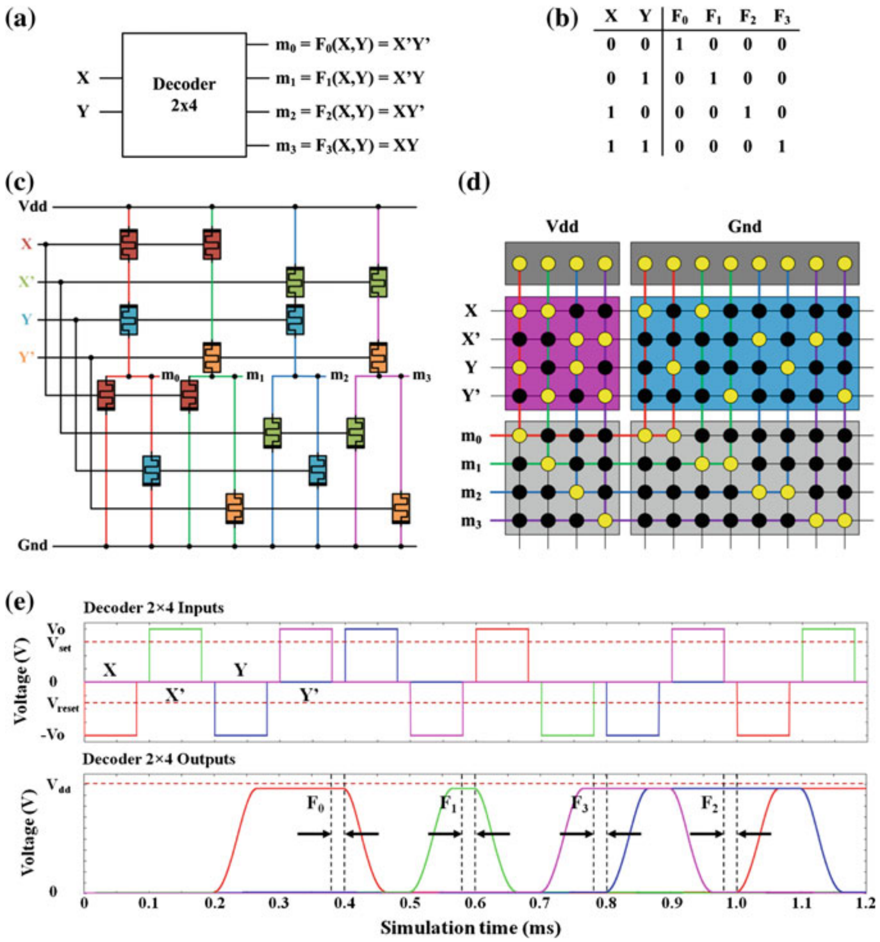


Fig. 4.17 **a** Block diagram of a 2 × 4 decoder, **b** its truth table, and **c** the corresponding compact memristive CMOS-*like* circuit schematic. Devices associated with the same input signals are colored correspondingly. **d** The configuration of the simulator's logic tile. **e** The output response of F_0 (*red*), F_1 (*green*), F_2 (*blue*) and F_3 (*magenta*) for all possible input variations of the signals X (*red*) and Y (*blue*). The time gap between the input signal transitions, where the circuit outputs should be read, is highlighted using the *vertical dashed lines*

interconnection lines in the circuit are colored accordingly to comply with the logic tile configuration shown in Fig. 4.17d. The decoder comprises two inputs, defined as X and Y, and four output logic functions representing the four minterms denoted as m_0, m_1, m_2 and m_3, respectively. For each function $F_i(X, Y)$ we find its complement $F'_i(X, Y)$, and the expression $F_i(X', Y')$ to be used for the CMOS-*like* design and configuration of the simulator. In the logic tile the FPM and RPM areas are not of equal size; the FPMs part is defined larger in order to accommodate the definition of the circuit on the tile according to the aforementioned logic functions.

Figure 4.17e presents the simulation results. The notation and the employed colors facilitate visual correspondence of the output signals with the circuit schematic and the tile configuration in Fig. 4.17c, d, respectively. Each signal transition is always followed by the corresponding change of its complement. In the output response graphs, attention should be paid again only after the complement of a particular signal completes its transition.

The simulation begins with the logic combination XY = “00” and the consecutive input signal transitions follow a two-bit width Gray-like code sequence, where only one input signal changes each time, to finally return to the initial combination. It should be noticed that there are periods where more than one output signals are found at logic ‘1’. Nevertheless, this takes place only before the complement signals complete their transition, because transitions of every input signal and its complement are not simultaneously applied each time to the respective inputs of the circuit. It can be observed that in every denoted gap only one of the outputs of the decoder is found at logic ‘1’ (i.e. high voltage level) whereas the rest of the outputs remain low, as expected.

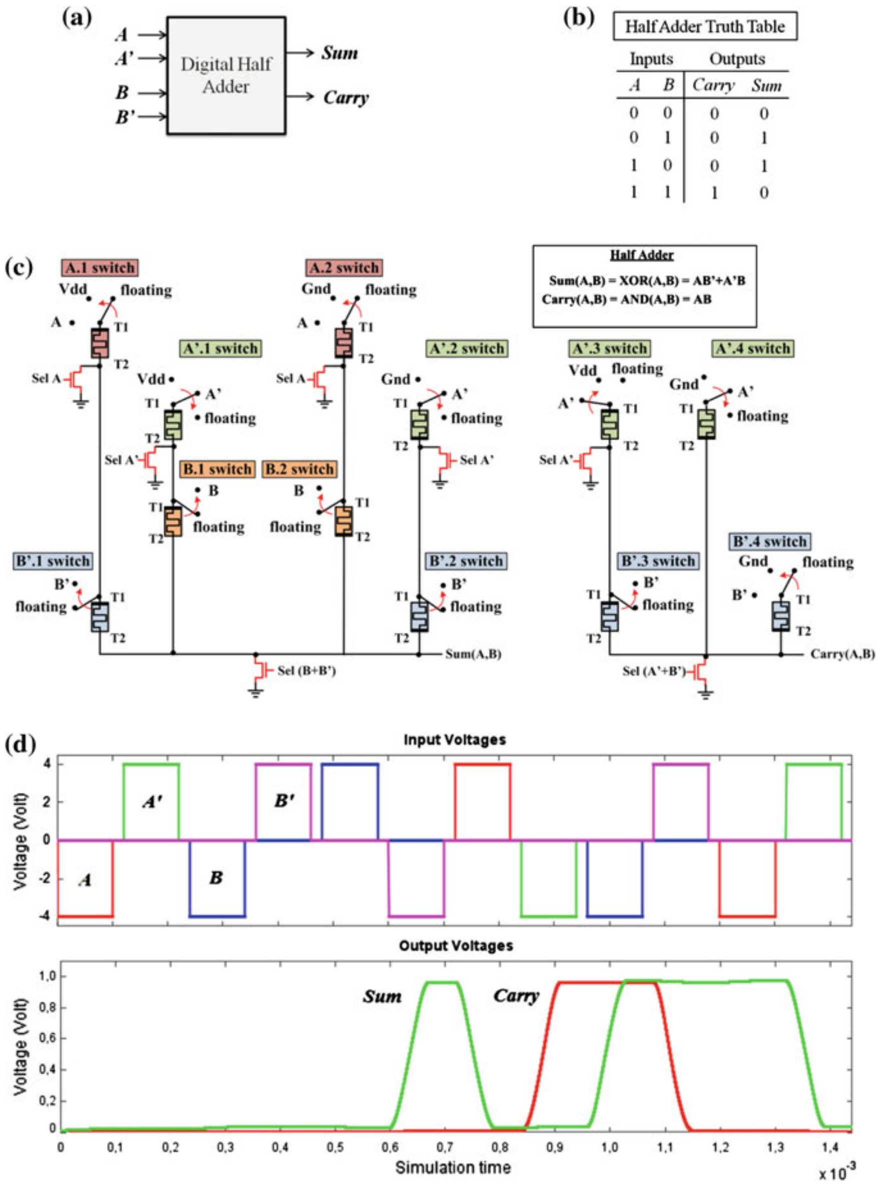
In the same manner, any $2^n \times n$ ($n \times 2^n$) encoder (decoder) circuit can be designed and implemented following the CMOS-*like* paradigm.

4.4.3.3 Half Adder

In the context of large combinational logic circuits, Fig. 4.18 summarizes the design methodology and simulation of a CMOS-*like* memristive digital half adder (HA). The HA circuit comprises two outputs, indicating the binary result of the addition, and four inputs which include the complements of the two main input signals; the inverted input signals are driven from the outside. The general block diagram and the corresponding truth table are shown in Fig. 4.18a, b, whereas the schematic of the circuit which implements the output logic functions of the HA is illustrated in Fig. 4.18c. The memristors that are associated with common input signals are found in the same horizontal line and are colored correspondingly.

Two different circuits implement separately the two output logic functions for the *Sum*(A, B) and the *Carry*(A, B), given in the inset of Fig. 4.18c. We simulated the HA circuit using the developed simulation environment. In simulation we use the following set of values for the parameters of the model $\{a_x, b, c, m, f_o, L_o, V_{SET}, V_{RESET}\} = \{5 \times 10^6, 0, 0.1, 82, 310, 5, 3\text{ V}, -1.5\text{ V}\}$, and we consider a R_{OFF}/R_{ON} ratio of two orders of magnitude with $R_{OFF} \approx 200\text{ k}\Omega$ and $R_{ON} \approx 2\text{ k}\Omega$. Also, we set $V_{DD} = 1$ V and apply 100 μs-wide input (write) pulses of $V_o = \pm 4$ V. The simulation of the output response for *Sum* and *Carry* is shown in Fig. 4.18d. Simulation begins with the input combination AB = “00” and finishes with the same input values. By comparing the output values (when the signals’ complements A' and B' complete their transition as well) with the corresponding truth table, we observe that memristive binary addition was successfully performed.

The presented circuit examples altogether prove that this straightforward design methodology for memristor-based complex combinational circuits enables the



Half Adder Truth Table

Inputs		Outputs	
A	*B*	*Carry*	*Sum*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Fig. 4.18 Memristive CMOS-*like* digital half adder. **a** The general block diagram. **b** The truth table. **c** The circuit schematic following the CMOS-*like* design paradigm. Memristors associated with a specific input signal are colored correspondingly: *pink* for *A*, *green* for *A′*, *orange* for *B* and *blue* for *B′*, respectively. **d** Simulation result for *Sum* (*green*) and *Carry* (*red*) for all possible input variations of signals *A* (*red*) and *B* (*blue*)

design and implementation of any digital logic circuit using novel nanoelectronic architectures.

4.4.4 Overview and Comparison

Here we provide a general overview of the application potential of this emerging sequential logic circuit architecture which, owing to the collective memristive dynamics, it has been shown that it is capable of universal computation. The following performance comparison with other techniques quantifies and highlights the importance of its contribution. However, the "doomed" straight comparison with nowadays ultra high-performance silicon integrated logic circuits is avoided since memristive technology is still at an early stage and further research is required at the device, circuit, and architecture levels to determine the practical utility of such novel computing approaches.

Similarly to other memristive logic paradigms, in CMOS-*like* circuits the delay of logic operations involves the time that the memristors require to fully switch their states. This time depends on the level of the applied voltage since higher voltage pulses will switch the devices more quickly, thus reducing switching energy. Possibly a memristor will not switch completely if the amplitude of the programming (write) pulse is not high enough or if the pulse is not applied for sufficient time. In such a case unacceptable output voltage levels might occur. According to [54], the time required to change the state of a linear TiO_2-based memristor (T_w), which is directly connected to a voltage source, is given by the following equation:

$$T_W = \frac{L_0^2 \cdot \beta}{2 \cdot \mu_V \cdot V_m} \tag{4.2}$$

where β is the $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ ratio, L_0 is the thickness of the device, V_{m} is the magnitude of the applied voltage, and μ_v is the mobility of oxygen vacancy dopants. We observe that T_w is a function of the physical parameters of the device and increases with increase in L_0 and β, whereas it is inversely proportional to the applied voltage.

The power consumed during logic operations depends on the resistance of all memristive devices involved in the computational process. The instantaneous current flowing through a memristor $i(t)$ during state-switching depends on its current memristance and on the applied voltage V_{m}. Thus, the energy dissipated during any access operation is calculated as:

$$E_W = \int_{o}^{Tw} V_m i(t) dt. \tag{4.3}$$

Overall, the performance of any memristive logic design paradigm, in terms of processing speed and energy consumption, will strongly depend on device material selection and operation schemes. The memristors themselves are capable of fast (nanoseconds) and low-energy (picojoules) switching [21]; a sub-nanosecond switching of tantalum oxide-based memristive devices [23] provides at the moment that this text is written the best estimate for the duration of logic computations. Thus, in terms of speed and energy consumption, memristor-based circuits seem very promising for future nanoelectronics. Circuit area, though, will depend on the type of integration with the silicon-based driving circuitry and/or on the utilization of the ultra high device density of the memristive nanowire crossbar.

Compared to imply logic, which up to now is the most recognized sequential approach for logic implementation with memristors, the CMOS-*like* paradigm: (i) *reduces significantly the number of sequential steps needed to perform logic operations*. More specifically, the necessary computation steps equal the number of the circuit's input signals, plus (if necessary) the number of their complements; thus they reach a maximum of $2n$ for an n-input arbitrary Boolean function, given that inputs' complements are considered readily available. Moreover: (ii) *it significantly simplifies the circuit design procedure* because the same well-known design principles with conventional CMOS are considered. In terms of circuit area, the exact number of needed memristors cannot be formerly defined because a specific logic function has many equivalent CMOS-*like* circuit implementations. However, for the set of universal logic gates {NAND, NOR, NOT} the number of used memristors is {4, 4, 2}, thus the CMOS-*like* design scheme proves slightly more costly than imply logic where e.g. 3 memristors suffice for a 2-input NAND operation.

Nevertheless, cascading CMOS-*like* logic circuits is not supported because the circuit output varies between $\text{GND} - V_{\text{DD}}$ whereas the necessary input signals applied to the bipolar memristors include negative programming voltages; hence input and output are incompatible. A possible solution to this handicap, though, could be the use of unipolar memristive devices.

4.5 A Memristive Logic Family for Parallel Processing of Applied Input Signals

Practical application of the emerging memristive technology in logic circuit design will most likely require substantial parallel operations in order to countervail the impact from the silicon-based driving circuitry. Both CMOS-*like* and imply-based design approaches consider serial processing of input signals; a design allowing parallel processing of inputs would be certainly more promising and could accelerate practical implementation of new generation of logic chips based on memristive devices. The previously summarized MRL paradigm was an early approach in this direction, also enabling significant physical integration savings. However, MRL provides only a restricted set of possible Boolean logic operations, while it uses linear memristors which respond slower than threshold-type devices. In the following section we exploit the threshold-dependent resistance switching behavior of memristors and their compositions to form a novel memristive logic family, which enables parallel execution of Boolean operations.

4.5.1 Boolean Logic Operations Based on Threshold-Type Resistance Switching

Here we present a memristive logic family which uses the total memory conductance (memductance) of the employed devices for the computation of parallel Boolean AND, OR, NAND, NOR, XOR, and XNOR operations.

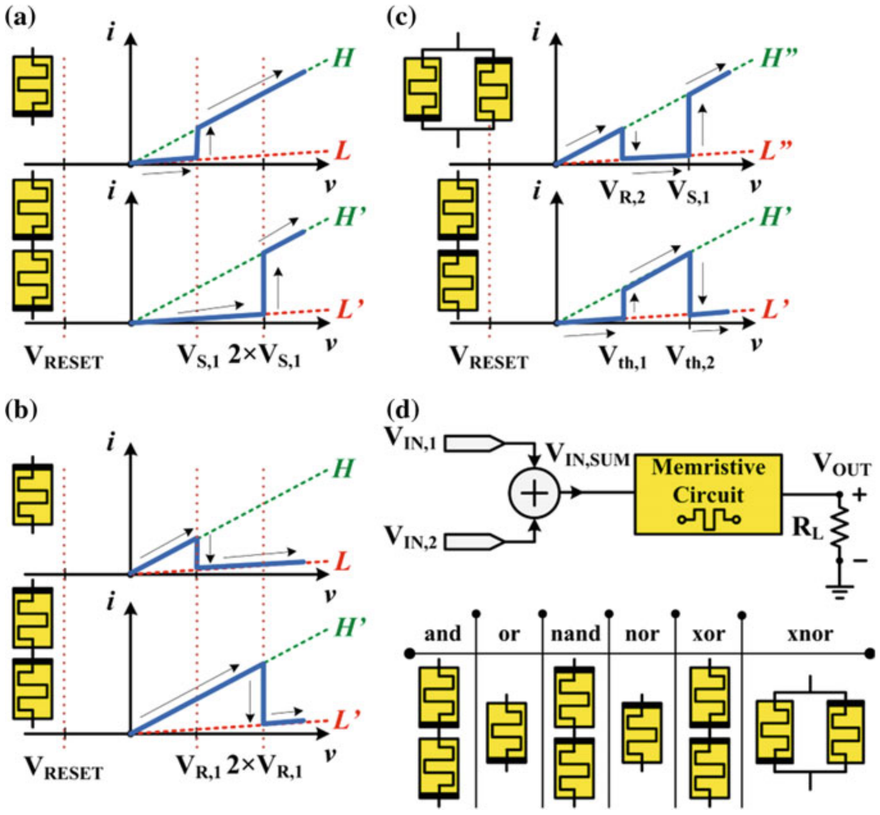


Fig. 4.19 General concept for the construction of 2-input memristive logic gates. **a**-**c** Qualitative *i*–*v* characteristics corresponding to FPMs or RPMs and their series/parallel combination. **d** Circuit implementing any of the six provided logic operations with two variables

Figure 4.19 summarizes the general concept for the construction of two-input memristive logic gates; the switching characteristics of memristors assume that the devices are kept in the ohmic regime in all dynamic ranges (i.e. have linear ON and OFF states) and are found in agreement with experimentally observed switching dynamics in [15]. According to Fig. 4.19d the aggregate input voltage is applied to a memristive ensemble which can be any of the six provided options, each implementing a particular logic operation. All necessary input combinations of the aggregated input signals could be generated by either using appropriate switches or via a summing amplifier. The qualitative *i*–*v* graphs of Fig. 4.19a-c show why memductance is inherently suitable to be the state variable for Boolean logic operations. In all cases the input voltages consist in 0 V for logic '0', whereas logic '1' corresponds to a voltage value which must lie between the first (lower) and the second (higher) of the defined switching thresholds.

As shown in Fig. 4.19a, a forward polarized memristor (FPM) will switch from a low conductance (*L*) to a high conductance (*H*) if any of the applied inputs is logic '1' (or both), i.e. it exceeds the set threshold $V_{\mathrm{S},1}$. Likewise, when employing two FPMs in series, the composite memductance will rise from a low value (L') to a high value (H') only when both inputs are logic '1' so that the aggregate input voltage exceeds the cumulative set threshold 2 × $V_{\mathrm{S},1}$. Therefore, memductance in these two cases is defined by the following equations which describe OR and AND logic operations as functions of the aggregate applied voltage:

$$OR : G\left(V_{IN,SUM} = V_{IN,1} + V_{IN,2}\right) = \begin{cases} H, & V_{IN,SUM} > V_{S,1} \\ L, & otherwise \end{cases} \tag{4.4}$$

$$AND : G\left(V_{IN,SUM} = V_{IN,1} + V_{IN,2}\right) = \begin{cases} H', & V_{IN,SUM} > 2 \times V_{S,1} \\ L', & otherwise \end{cases} \tag{4.5}$$

When employing a reversely polarized memristor (RPM) or two RPMs in series, as shown in Fig. 4.19b, under similar working principles the circuit implements a NOR and a NAND logic gate, respectively. Unlike in Fig. 4.19a, it is the reset thresholds $V_{\mathrm{R},1}$ and 2 × $V_{\mathrm{R},1}$ which now define memristance switching. OR and NOR gates with more inputs can be implemented by adding more than two signals in the applied sum; this number is practically limited by the maximum voltage a device can tolerate without being damaged. For AND and NAND operations an additional memristor has to be included for each additional input signal so as to accordingly increase the cumulative threshold.

As shown in Fig. 4.19c, when using two reciprocal devices in series or in parallel, the circuit implements XOR and XNOR logic operations, respectively. In the series connection the composite memductance rises from a low level (L') to a high level (H') if any of the applied inputs is logic '1', i.e. exceeds the set threshold $V_{\mathrm{th},1}$. However, if both inputs are logic '1' then together they exceed the reset threshold $V_{\mathrm{th},2}$ and the composite memductance collapses to level L'. In the same fashion, the equivalent memductance of the parallel configuration collapses from level H'' to L'' if only one input is logic ‘1’, i.e. > $V_{R,2}$, but rises again to H'' if both inputs are logic ‘1’, thus together exceed $V_{S,1}$. The memductance in these two cases is defined by the equations:

$$XOR : G\left(V_{IN,SUM} = V_{IN,1} + V_{IN,2}\right) = \begin{cases} H', & V_{th,1} < V_{IN,SUM} < V_{th,2} \\ L', & otherwise \end{cases} \tag{4.6}$$

$$XNOR : G\left(V_{IN,SUM} = V_{IN,1} + V_{IN,2}\right) = \begin{cases} L'', & V_{R,2} < V_{IN,SUM} < V_{S,1} \\ H'', & otherwise \end{cases} \tag{4.7}$$

Regarding the composite conductance levels of the memristive ensembles, assuming $\{L, H\} = \{G_{ON}, G_{OFF}\}$, then for the serial and the parallel compositions it is $\{L', H'\} = \{1/2 \times G_{ON}, 1/2 \times G_{OFF}\}$ and $\{L'', H''\} = \{2 \times G_{ON}, 2 \times G_{OFF}\}$.

Any potential problems caused by variation among the switching thresholds of individual devices, especially concerning memristive circuits where all the devices have the same polarity, will be overcome by optimally selected programming pulses. However, attention should be paid when memristors with different polarities are used. A simulation-based validation of the preferable relation between the V_{SET} and V_{RESET} thresholds showed that having $|V_{RESET}| > V_{SET}$ or $|V_{RESET}| = V_{SET}$ ($|V_{RESET}| < V_{SET}$) is preferable for the anti-series (for the anti-parallel) memristors; when $|V_{RESET}| = V_{SET}$ the parallel memristors initiate switching simultaneously ($R_{OFF} \| R_{ON} \leftrightarrow R_{ON} \| R_{OFF}$) and hence there is almost no intermediate switching stage with both of them in R_{OFF}. Overall, the more expanded the thresholds for each configuration are, the easier it becomes to avoid potential overlap due to variation [5]. More expanded thresholds could be achieved by either following the concept of composite memristive structures discussed in Chap. 3 [36], or by incorporating pairs of anti-parallel diodes in series with the memristors [15].

Circuit output (V_{OUT}) of either single or cascaded logic gates is read using a series load resistor. Such resistor should have a small value (better smaller than R_{ON}) so as to draw a small voltage on its terminals and hence not to impede the complete (or almost complete) switching of the employed memristor(s). This is even more crucial when logic gates are cascaded, as shown in Fig. 4.20a, where a typical sum-of-products digital circuit is presented, with the corresponding memristive implementation in Fig. 4.20b. After conducting a series of simulations for load resistor values ranging between [1, 40] kΩ while assuming a memristance range of [2, 200] kΩ, it was observed that loads of up to 10 kΩ guarantee the expected behavior, whereas for larger loads the memristor(s) fail to completely switch their states. As a consequence, the output voltage levels corresponding to logic ‘1’ and logic ‘0’ tend to approximate each other, thus ruining the circuit behavior.

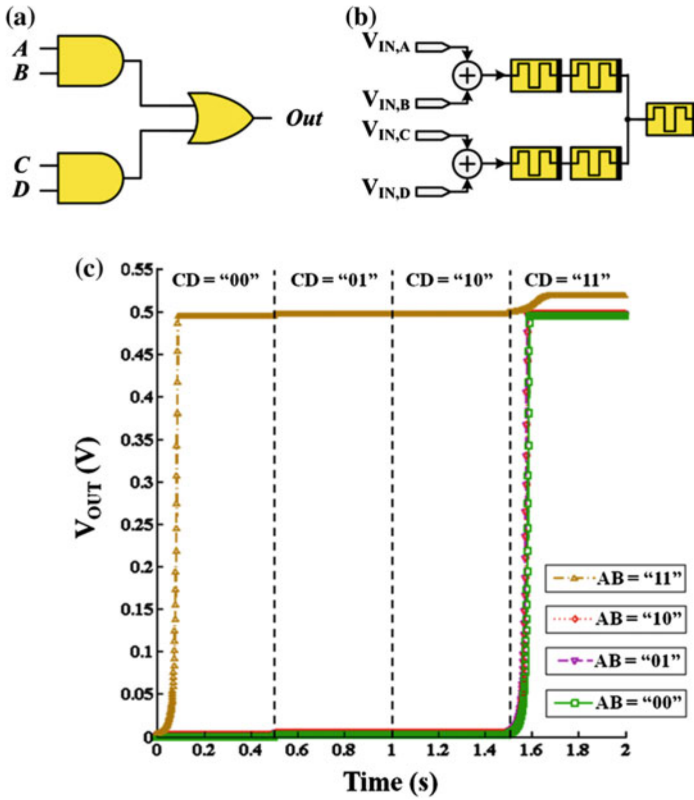


Fig. 4.20 Cascaded memristive logic gates. **a** Schematic of a typical sum-of-products logic function, **b** its memristive implementation, and **c** simulation results using SPICE

4.5.2 Verification Using SPICE

We simulated the circuit of Fig. 4.20 using the Cadence PSPICE simulation environment by employing the threshold-type model of voltage-controlled memristors presented in Chap. 2 [51]. Input signals $V_{\mathrm{IN,A\text{-}D}}$ were set to 0 and 1.8 V to represent logic '0' and logic '1', respectively. The parameters of the model were set as: $\{a_x, b, c, m, f_o, L_o, V_{\mathrm{SET}}, V_{\mathrm{RESET}}\} = \{10^3, 0, 0.1, 82, 310, 5, 1\ \mathrm{V}, -1\ \mathrm{V}\}$, and the resistance ratio was set to $R_{\mathrm{OFF}}/R_{\mathrm{ON}} \approx 10^2$ with $R_{\mathrm{OFF}} \approx 200\ \mathrm{k\Omega}$ and $R_{\mathrm{ON}} \approx 2\ \mathrm{k\Omega}$. In Fig. 4.20c we used a resistor $R_L = 1\ \mathrm{k\Omega}$.

In such memristive circuits we normally assume that the devices are initially programmed to either of the boundary resistive states because the computational result depends on the initial state of the memristors; hence occasionally resetting the gates is required. However, there is no need to particularly access each memristive device so as to adjust its memristance. In fact, regardless of the current state of the devices after any logic computation, resetting the logic gates/compositions can be done by applying a negative programming voltage pulse which exceeds a specific threshold (generally shown in Fig. 4.19 as V_{RESET} for all cases). Such pulse will eventually reset all memristors to the "desired" initial boundary resistive states (being either R_{ON} or R_{OFF}) depending on their polarity. Nevertheless, in Fig. 4.20c we avoid applying resetting pulses and, in turn, present separately the V_{OUT} response for all combinations of the input voltages $V_{\mathrm{IN,C\text{-}D}}$ while holding $V_{\mathrm{IN,A\text{-}B}}$ to particular values. The simulation result confirms the correct behavior of the circuit for all input logic combinations. The memory property of the memristor ensures that the results of previous computations are maintained in the states of the employed devices.

Similarly to the MRL design approach, all presented gates lack signal restoration since they are built out of passive elements only. Output voltage levels degrade and thus these logic gates cannot be cascaded for many stages without any signal amplification. Therefore, CMOS inverter/buffer is necessary to make this logic family computationally complete and to provide signal inversion/amplification as well.

4.5.3 Overview and Comparison

Compared to the sequential nature of CMOS-*like* and imply-based logic, this approach assumes parallel processing of input signals. Moreover, it utilizes the threshold-type switching of memristors for the computation of Boolean logic operations, unlike "threshold logic" which implements ratioed logic by continuously controlling the memory state of memristors.

The MRL has a number of characteristics in common with the proposed threshold-based approach, which are summarized below:

- It assumes bipolar memristive devices;
- The logic state-variable is voltage;
- The computational process is composed of only a single step;
- The topology of the circuit determines the logical function;
- It is non-inverting and non-restoring.

However, the main differences between the two logic families are summarized below:

- Proposed: Memristors may have any polarity and be connected either in series or in parallel;
  MRL: Memristors are connected only in series with opposite polarities.
- Proposed: The sum of the input signals is applied to a common input node;
  MRL: Each input signal is applied to a different terminal of the memristors.
- Proposed: Before each subsequent logic operation the memristors are reset to their initial states;

MRL: The computational result is independent of the initial state of the memristors.

- Proposed: Threshold-type switching is preferable for the memristors;
  MRL: A linear memristive device with no threshold is preferred.

Overall, although both logic families share almost the same disadvantages, the proposed approach provides a much richer set of possible Boolean operations, it is polarity-independent and compatible with threshold-type switching which characterizes the majority of the experimental memristive devices.

In all SPICE-based simulations, whose results were illustrated throughout this chapter, we tried to take into consideration as many circuit parameters as possible and used practical models of existing devices (e.g. transistors, op-amps, etc.), so as to achieve more realistic results. The fact that the assumed switching characteristics of memristors are in agreement with experimentally observed memristive dynamics, along with the continuous improvement of the memristance switching behavior, thanks to the incessant accumulation of knowledge on resistive switching materials and the underlying phenomena by academia and industry, are encouraging for the future implementation and the establishment of unconventional computing paradigms and sophisticated memristive circuits and systems such as those presented here.

References

1. Y.V. Pershin, M. Di Ventra, Neuromorphic, digital and quantum computation with memory circuit elements. Proc. IEEE **100**(6), 2071–2080 (2012)
2. E. Lehtonen, M. Laiho, CNN using memristors for neighborhood connections, in *12th International Workshop Cellular Nanoscale Network Application (CNNA)*, Berkeley, CA (2010)
3. K.H. Kim, S. Gaba, D. Wheeler, J.M. Cruz-Albrecht, T. Hussain, N. Srinivasa, W. Lu, A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. Nano Lett. **12**(1), 389–395 (2012)
4. G. Howard, E. Gale, L. Bull, B. de Lacy Costello, A. Adamatzky, Evolution of plastic learning in spiking networks via memristive connections. IEEE Trans. Evol. Comput. **16**(5), 711–729 (2012)
5. I. Vourkas, G.C. Sirakoulis, Nano-crossbar memories comprising parallel/serial complementary memristive switches. BioNanoScience **4**(2), 166–179 (2014)
6. H. Kim, M.P. Sah, C. Yang, T. Roska, L.O. Chua, Neural synaptic weighting with a pulse-based memristor circuit. IEEE Trans. Circ. Syst. I, Reg. Papers **59**(1), 148–158 (2012)
7. S. Kim, H.Y. Jeong, S.K. Kim, S.Y. Choi, K.J. Lee, Flexible memristive memory array on plastic substrates. Nano Lett. **11**(12), 5438–5442 (2011)
8. E. Linn, R. Rosezin, S. Tappertzhofen, U. Bottger, R. Waser, Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations. Nanotechnology **23** (305205) (2012)
9. E. Lehtonen and M. Laiho, Stateful implication logic with memristors, in *IEEE/ACM Int. Symp. Nanoscale Architectures (NANOARCH)*, San Francisco, CA (2009)
10. J. Borghetti, Z. Li, J. Straznicky, X. Li, D.A.A. Ohlberg, W. Wu, D.R. Stewart, R.S. Williams, A hybrid nanomemristor/transistor logic circuit capable of self-programming. Proc. Nat. Acad. Sci. (PNAS) USA **106**(6), 1699–1703 (2009)

11. J. Borghetti, G.S. Snider, P.J. Kuekes, J.J. Yang, D.R. Stewart, R.S. Williams, Memristive switches enable 'stateful' logic operations via material implication. Nature **464**(7290), 873–876 (2010)
12. M. Di Ventra, Y.V. Pershin, The parallel approach. Nat. Phys. **9**, 200–202 (2013)
13. S. Kvatinsky, G. Satat, N. Wald, E.G. Friedman, A. Kolodny, U.C. Weiser, Memristor-Based Material Implication (IMPLY) logic: design principles and methodologies. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **22**(10), 2054–2066 (2014)
14. E. Lehtonen, J.H. Poikonen, M. Laiho, Implication logic synthesis methods for memristors, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seoul, South Korea, (2012)
15. G. Ligang, F. Alibart, D.B. Strukov, Programmable CMOS/memristor threshold logic. IEEE Trans. Nanotechnol. **12**(2), 115–119 (2013)
16. I. Vourkas, G.C. Sirakoulis, Memristor-based combinational circuits: a design methodology for encoders/decoders. Microelectron. J. **45**(1), 59–70 (2014)
17. J.J. Yang, D.B. Strukov, D.R. Stewart, Memristive devices for computing. Nat. Nano. **8**, 13–24 (2013)
18. Vourkas, G.C. Sirakoulis, Recent progress and patents on computational structures and methods with memristive devices. Recent Pat. Electr. Electron. Eng. **6**(2), 101–116 (2013)
19. E. Lehtonen, J.H. Poikonen, M. Laiho, Two memristors suffice to compute all Boolean functions. Electron. Lett. **46**(3), 239–240 (2010)
20. W. Zhao, D. Querlioz, J.O. Klein, D. Chabi, C. Chappert, Nanodevice-based novel computing paradigms and the neuromorphic approach, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seoul, South Korea (2012)
21. International Technology Roadmap for Semiconductors (ITRS) (2013). Available: http://www.itrs.net/. Accessed June 2014
22. Y.V. Pershin, M. Di Ventra, Memory effects in complex materials and nanoscale systems. Adv. Phys. **60**(2), 145–227 (2011)
23. C. Torrezan, J.P. Strachan, G. Medeiros-Ribeiro, R.S. Williams, Sub-nanosecond switching of a tantalum oxide memristor. Nanotechnology **22**(48), 485203 (2011)
24. S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U.C. Weiser, E.G. Friedman, MRL—Memristor Ratioed Logic, in *13th International Workshop on Cellular Nanoscale Network and Application (CNNA)*, Turin, Italy (2012)
25. J. Rajendran, H. Manem, R. Karri, G.S. Rose, Memristor based programmable threshold logic array, in *IEEE/ACM International Symposium on Nanoscale Architecture (NANOARCH)*, Anaheim, CA (2010)
26. S. Paul, S. Bhunia, A scalable memory-based reconfigurable computing framework for nanoscale crossbar. IEEE Trans. Nanotechnol. **11**(3), 451–462 (2012)
27. G.S. Snider, P.J. Kuekes, R.S. Williams, CMOS-like logic in defective, nanoscale crossbars. Nanotechnology **15**, 881–891 (2004)
28. M.M. Ziegler, M.R. Stan, CMOS/nano co-design for crossbar-based molecular electronic systems. IEEE Trans. Nanotechnol. **2**(4), 217–230 (2003)
29. D.B. Strukov, K.K. Likharev, CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. Nanotechnology **16**(6), 888–900 (2005)
30. Y.V. Pershin, M. Di Ventra, Solving mazes with memristors: a massively parallel approach. Phys. Rev. E **84**, 046703 (2011)
31. Y.V. Pershin, M. Di Ventra, Self-organization and solution of shortest-path optimization problems with memristive networks. Phys. Rev. E **88**, 013305 (2013)
32. I. Vourkas, G.C. Sirakoulis, Study of memristive elements networks. J. Nano Res. **27**, 5–14 (2014)
33. F. Jiang, B.E. Shi, The memristive grid outperforms the resistive grid for edge preserving smoothing, in *European Conference on Circuits Theory and Design (ECCTD)*, Antalya, Turkey (2009)
34. Z. Ye, S.H.M. Wu, T. Prodromakis, Computing shortest paths in 2D and 3D memristive networks. 15 Mar 2013. Available: http://arXiv:1303.3927

35. E. Linn, R. Rosezin, C. Kugeler, R. Waser, Complementary resistive switches for passive nanocrossbar memories. Nat. Mater. **9**(5), 403–406 (2010)
36. I Vourkas, G.C. Sirakoulis, On the analog computational characteristics of memristive networks, in *20th IEEE International Conference on Electronics, Circuits, Systems (ICECS),* Abu Dhabi (2013)
37. A.N. Whitehead, B. Russell, *Principia Mathematica*, vol. I(7) (Cambridge University Press, Cambridge, 1910)
38. R.H. Wilkinson, A method of generating functions of several variables using analog diode logic. IEEE Trans. Electron. Comput. **EC-12**(2), 112–129 (1963)
39. S. Muroga, *Threshold Logic and its Applications, Hoboken, NJ* (Wiley, USA, 1972)
40. R. Zhang, P. Gupta, L. Zhong, N.K. Jha, Synthesis and optimization of threshold logic networks with application to nanotechnologies, in *Design Automation and Test in Europe Conference (DATE)*, Paris, France, 2004
41. V. Beiu, J.M. Quintana, M.J. Avedillo, VLSI implementations of threshold logic: a comprehensive survey. IEEE Trans. Neural Netw. **14**(5), 1217–1243 (2003)
42. Y. Leblebici, H. Ozdemir, A. Kepkep, U. Cilingiroglu, A compact high-speed (31, 5) parallel counter circuit based on capacitive threshold logic gates. IEEE J. Solid-State Circuits **31**(8), 1177–1183 (1996)
43. I. Vourkas, G.C. Sirakoulis, A novel design and modeling paradigm for memristor-based crossbar circuits. IEEE Trans. Nanotechnol. **11**(6), 1151–1159 (2012)
44. J.R. Heath, P.J. Kuekes, G.S. Snider, R.S. Williams, A defect-tolerant computer architecture: opportunities for nanotechnology. Science **280**(5370), 1716–1721 (1998)
45. I. Vourkas, G.C. Sirakoulis, Nano-crossbar memories comprising parallel/serial complementary memristive switches. BioNanoScience **4**(2), 166–179 (2014)
46. A. Chen, Accessibility of nano-crossbar arrays of resistive switching devices, in *11th IEEE Conference on Nanotechnology (IEEE-NANO)*, Portland, OR (2011)
47. S. Shin, K. Kim, S.M. Kang, Analysis of passive memristive devices array: data-dependent statistical model and self-adaptable sense resistance for RRAMs. IEEE Proc. **100**(6), 2021–2032 (2012)
48. J. Liang, H.-S.P. Wong, Cross-point memory array without cell selectors—Device characteristics and data storage pattern dependencies. IEEE Trans. Electron. Devices **57** (10), 2531–2538 (2010)
49. M.A. Zidan, H.A.H. Fahmy, M.M. Hussain, K.N. Salama, Memristor-based memory: the sneak paths problem and solutions. Microelectronics J. **44**(2), 176–183 (2013)
50. I. Vourkas, D. Stathis and G.C. Sirakoulis, Improved read voltage margins with alternative topologies for memristor-based crossbar memories, in *21st IFIP/IEEE International Conference on on Very Large Scale Integrated (VLSI-SoC)*, Istanbul (2013)
51. I. Vourkas, A. Batsos, G.C. Sirakoulis, SPICE modeling of nonlinear memristive behavior. Int. J. Circ. Theor. Appl. **43**(5), 553–565 (2015)
52. I. Vourkas, G.C. Sirakoulis, Employing threshold-based behavior and network dynamics for the creation of memristive logic circuits and architectures. Physica Status Solidi (c), in *Proceedings of E-MRS 2014 Spring Meeting Symposium S: Memristor materials, mechanisms and devices for unconventional computing,* vol. 12, no. 1-2, pp. 168–174 (2015)
53. Easy Java Simulations (EJS). Available: http://fem.um.es/Ejs/. Accessed 2014
54. Y. Ho, G.M. Huang, P. Li, Dynamical properties and design analysis for nonvolatile memristor memories. IEEE Trans. Circuits Syst. I, Reg. Papers **58**(4), 724–736 (2011)

Chapter 5
Memristive Crossbar-Based Nonvolatile Memory

5.1 Introduction

The primary purpose of many memory systems is to store massive amounts of data, hence making storage capacity (or density) one of the most important system parameters. A functional memory cell in an array usually comprises two components: the “storage node” and the “select device”; the latter allows a given memory cell to be addressed for read or write. Both components impact scaling limits for different memory technologies.

At the highest abstraction level, memory technologies are separated in volatile and nonvolatile, according to their ability to retain data without power. Today, nonvolatile memory (NVM) is essentially ubiquitous, offers key advantages, and the degree of nonvolatility is measured in terms of the length of time that data can be retained. Endurance and retention are requirements unique to NVM technologies and determine whether the device has adequate utility to be of interest to the end-user. NVM technologies are further categorized by their maturity. Flash memory is nowadays considered the baseline NVM because it is highly mature, well optimized, and has a significant commercial presence [1]. Flash memories are based on simple one-transistor cells, where the storage node (floating gate) and the select/access device (transistor) are combined in one device. Rapid progress in NAND Flash technology in recent years resulted in tighter half-pitch (i.e. minimum feature size of the process technology) than that of DRAM [2]. Current memory technology roadmap forecasts NAND Flash memory will continue to dominate high-density storage in the short and intermediate terms. In longer term years, however, a significant slowing down of scaling is expected because NAND Flash is facing scaling limitations, including the number of electrons per logic level and breakdown voltage between neighboring word lines.

Several non-conventional NVMs that are not based on charge storage, e.g. Spin Transfer Torque Magnetostatic RAM (STT-MRAM), phase-change RAM (PCRAM), and resistive RAM (ReRAM), form the category of so-called “emerging”

I. Vourkas and G.Ch. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Emergence, Complexity and Computation 19,
DOI 10.1007/978-3-319-22647-7_5

memories, which constitute the focus of this chapter. These are the least mature memory technologies, but have been shown to offer significant potential benefits if various scientific and technological hurdles can be overcome. Such technologies are nowadays being investigated as promising candidates to replace the popular Flash memories in the future, and potentially even the other conventional memories such as SRAM and DRAM. Their corresponding memory elements usually have a two-terminal structure (the storage node) which cannot serve as a cell selection device. As far as ReRAM is concerned, the type of storage device is normally based on memristors.

Memristor-based ReRAM provides many advantages such as scalability, energy efficiency, density, CMOS compatibility, etc. Table 5.1, presenting data taken from 2013 International Technology Roadmap for Semiconductors (ITRS) [2], illustrates a brief comparison between conventional and emerging memories; it clearly shows that ReRAM with scalability down to sub-10 nm, comparable read/write times with today's memories and good retention time, is promising to advance the state-of-the art. Flash memory is the benchmark against which prototypical and emerging NVM technologies are measured.

At the architectural level, crossbar cell array structure is considered one of the best ways to implement ReRAMs [3]. Crossbar architecture offers several benefits including pattern regularity, manufacturing flexibility, defect-tolerance, CMOS compatibility, and the highest possible device density [4]. Passive crossbar arrays comprising bipolar memristors at their junctions, have been proposed as convenient geometries to achieve higher density and performance [5–7]; they even provide the possibility of having multiple array-layers stacked on top of each other to further augment density and bandwidth [8]. Recently, 20 nm 1 Gb 2-layer 3D ReRAM was implemented [9], thus under optimistic scenarios, 3D ReRAM may continue the density scaling beyond 2D and 3D NAND Flash capabilities [10, 11]. However, still there is not enough understanding of the atomic details at device level to be able to project when this will limit the scaling of ReRAM.

For high-density applications one difficult challenge is how to achieve small cell size and what access device to use. For bipolar operation (which is the most common mode) the lack of a compact select device makes it hard to achieve cross-point cell-size of $4F^2$. The most commonly used memory select devices are

Table 5.1 Key features of traditional and emerging memory technologies

	Current baseline technologies			Emerging technologies		
	DRAM	SRAM	Flash NAND	PCM	STT MRAM	ReRAM
Feature size	36–65 nm	45 nm	16 nm	45 nm	65 nm	5 nm
Cell area	6–$30F^2$	$140F^2$	$4F^2$	$4F^2$	$20F^2$	$4F^2$
Read time	2–10 ns	0.2 ns	0.1 ms	12 ns	35 ns	<10 ns
Write time	2–10 ns	0.2 ns	0.1–1.0 ms	100 ns	35 ns	<1.0 ns
Retention	4–64 ms	N/A	10 years	>10 years	>10 years	>10 years

transistors which, however, easily expand the cell size to $8F^2$ (DRAM) or $10F^2$ (NOR Flash) [2]. On the other hand, a typical passive crossbar memory where no rectifying devices are used to isolate the cells being written or read [12, 13], suffers from large amount of leakage current flowing through unselected cells called current sneak paths; this reduces both the size and the reliability (noise margin) of the memory [14–16]. Many solutions have been proposed to overcome or diminish this drawback. They can be classified into three classes:

- *Select devices*, which are separate devices such as a diodes or transistors which are connected with the ReRAM cells [15, 17–19];
- *Bias schemes*, where the voltages applied to non-accessed wordlines and bitlines are set to values different than those applied to accessed wordlines and bitlines; examples are multi-stage reading [15] and using AC signal instead of DC for sensing the stored data in the desired cells [20].
- *Switching-device modifications*, where the resistive devices are modified; examples are serially connecting two bipolar memristors with opposite polarities, resulting into a "complementary resistive switch-CRS" being able to block the current at low voltage irrespective of the state of the devices [21, 22], and the employment of a highly nonlinear memristor (due to current-controlled negative differential resistance) to overcome sneak path [23].

All the above solutions contribute to the reduction/removal of the current sneak-paths, though they still suffer from certain limitations. For instance, using a *select* transistor reduces the integration-density, whereas diode threshold voltages will decrease the output swing (diodes with sufficiently high forward current density are still under investigation); *bias schemes* and *device modifications* normally require complex reading schemes, thus impact both hardware-area and performance. Although at storage device level high-density ReRAM still must overcome several challenges to be cost-competitive to NAND Flash (examples are scalability below 10 nm and high $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ ratio), at the architecture level reliability is the major bottleneck; it is challenging to achieve high density and reliability without innovative, compact, and high-endurance cell selection devices. However, innovative approaches to memory cell structure and/or memory architecture could instead efficiently address the current sneak-path problem and pave the way towards the practical realization of passive high-density crossbar-based ReRAMs.

In the rest of this chapter we first present an overview of emerging ReRAM technologies, their potential benefits, and the key research challenges, with a focus on reduction/oxidation (Redox)-based RAM [24]. Afterwards, we briefly describe the basic operation principles of memristive memory cells, and we present the memristor-based crossbar memory architecture to finally focus on the serious negative impact of the current sneak-paths. Then, we explore two possible methodologies as means to deal with the sneak-path problem, concerning (i) novel storage cell structures [25], or (ii) modifications in the memory architecture [26, 27]. More specifically, (i) we study anti-parallel memristive switches (APMs) as potential cross-point elements in ReRAM arrays, in comparison with anti-serial memristive switches (ASMs), i.e. CRS. We provide a comprehensive and comparative presentation between them, while commenting on their overall performance and the most appropriate switching characteristics that the structural memristors should have in order to better fit to memory applications. Moreover, (ii) we present five alternative architectures (topologies) for passive crossbar ReRAM which are based on the introduction of a certain percentage of insulating nodes spread out inside the array according to specific distribution patterns. Both approaches enable crossbar memory arrays without external select devices, thus they simplify the array fabrication process and could be well-suited for future data storage applications. Finally, we present "XbarSim" [28], a GUI-based simulation tool developed using the JAVA programming language [29], aiming to serve students/researchers who wish to explore and study the memristive crossbar circuit architecture.

5.2 Overview of Redox-Based RAM Device Technology

A typical memristive device consists of two metallic electrodes that sandwich a thin dielectric insulating layer (I-layer) serving as permanent storage medium while making its leakage current almost zero. The exact mechanism differs significantly among the different materials being used, but the common link among all devices is an electric field which causes ionic movements and local structural changes in the storage medium, which in turn causes a measurable change in the resistance. In ReRAM the data is stored in the form of two (or multiple) resistance states of the memristor device, which in its simplest form relies on a metal-insulator-metal (MIM) stack.

In Redox-based ReRAM the physical mechanism for state-switching is based on reduction/oxidation (Redox)-related chemical effects [30]. The category of Redox-based ReRAM encompasses a wide variety of MIM structures; operation is based on a change in resistance caused by ion (cation or anion) migration combined with Redox processes involving either the electrode material, or the insulator material, or both. In many cases the conduction is of filamentary nature, and hence a one-time formation process (electroforming) is required before the bi-stable switching can be started. Electroforming stage corresponds to a voltage-induced resistance switching from an initial very high resistance state to a conductive state. If this effect can be controlled, memories based on this bi-stable switching process can be scaled to very small feature sizes. The switching speed is limited by the ion transport. If the active distance over which the anions or cations move is small (in the <10 nm regime), the switching time can be as low as a few nanoseconds [31, 32]. Although deeper understanding of the physical mechanisms governing the switching of the Redox-based ReRAM is a key challenge, nevertheless, recent experimental demonstrations of scalability, retention, and endurance are very encouraging [2].

In this section we briefly describe the three varieties of Metal-Oxide ReRAM, where switching is due to anion reconfiguration. These are: *bipolar-filamentary*, *unipolar-filamentary*, and *bipolar-nonfilamentary*. Bipolar versus unipolar behavior is distinguished by the requirement of opposite polarities for the SET and RESET operations (bipolar requires both polarities). Filamentary versus non-filamentary is characterized by the area through which electrical conduction and resistance switching take place. In the more common filamentary ReRAM, conduction occurs through a filament, which is typically small and of fixed size for a given material and forming conditions. Hence, the current through the device is not strongly dependent on device area. Conversely, in non-filamentary ReRAM, conduction takes place over a significant portion of the device area. In this case, the device current is directly proportional to the device area. The operating principles, current status, and challenges of the aforementioned three Metal-Oxide ReRAM categories, are given below.

5.2.1 Metal Oxide-Bipolar Filamentary ReRAM

Metal oxide-bipolar filamentary (MO-BF) ReRAM is an emerging bipolar resistance switching memory, often referred to as "valence change memory (VCM)" [24, 30]. The structure of MO-BF ReRAM cell is an asymmetric electrode/insulator/electrode stack. One electrode serves to create the interface where switching occurs, which is sometimes referred to as the active electrode. The other electrode serves as an ohmic contact and as a reservoir for the oxygen anions during the switching process. The most common switching metal oxides are TaO_x [33] and HfO_x [34] due to their excellent performance and CMOS compatibility. However, bipolar filamentary switching has been reported in numerous transition metal oxides including TiO_x [35]. These oxides are typically oxygen deficient (sub-stoichiometric). Additionally, it is common to form a MO-BF ReRAM cell from a bi-layer combination of oxides, where one contains a significantly higher oxygen stoichiometry than the other (e.g. Ta_2O_{5-x}/TaO_{2-x} [36]). Prior to switching, a MO-BF typically requires a one-time electroforming pulse which creates a switching filament with a high concentration of oxygen vacancies (OV). Switching is thought to occur due to the regulation of OVs in this switching channel.

Rapid improvement at the device level has been made in the past several years for MO-BF ReRAM. Demonstrated parameters include: dimensions of <10 nm [37], cell endurance of 10^{12} cycles [38], extrapolated ten-year retention at 85 °C [33], sub-ns switching times, and SET/RESET switching energy of 115 fJ/13 pJ [31, 32]. However, an improved understanding of the bipolar filamentary switching mechanism may solve one of the most significant technological issues of MO-BF ReRAM: *variability*. It will be important in the next years to have a functional ReRAM array that demonstrates excellent speed, fast readout, high endurance, scalability, low switching energy, high reliability, and low variability characteristics, altogether simultaneously.

5.2.2 Metal Oxide-Unipolar Filamentary ReRAM

Metal Oxide-Unipolar Filamentary (MO-UF) ReRAM is another resistive switching device, also referred to in the literature as "thermo-chemical memory (TCM)" due to the primary physical switching mechanism [30]. The device consists of a MIM structure with typical metal-oxides (e.g. NiO_x, HfO_x) as insulator materials and common metal electrodes such as Pt and Ni. The device can be asymmetric (i.e. have different top and bottom electrode materials), but asymmetry is generally not necessary. Unipolar switching allows using the same voltage polarity for changing the resistance from high to low (SET) and vice versa (RESET). In the general case, however, polarity is still important; repeatable SET/RESET switching only occurs for one voltage polarity with respect to one of the electrodes [39]. Only in symmetric structures (e.g. $Pt/HfO_2/Pt$) SET and RESET may occur irrespective of voltage polarity [40].

The switching process is generally understood as being filamentary, where conduction is caused by a filamentary arrangement of OVs throughout the I-layer. As with other filamentary ReRAM, an initial high voltage electroforming step is required to form the conductive filament. The unipolar character of the switching indicates that drift of charged defects does not play a role as it does in bipolar switching, but that thermal effects probably dominate.

Unipolar memristor is seen as advantageous since it allows the use of simple select devices (e.g. a diode) that can be stacked vertically with the storage element in a dense crossbar array. Also, the use of a single program voltage polarity greatly simplifies the circuitry. On the other hand, there are important tradeoffs between unipolar and bipolar switching modes. Unipolar switching typically shows a higher R_{OFF}/R_{ON} ratio. However, the key drawback is that resistance switching is typically obtained at higher currents (i.e. at higher power) than in bipolar mode, and also endurance is lower. Further study of the stability and control of the large resistance window at lower current levels is required to determine if unipolar ReRAM variability can be improved. As a result, major research and development work on ReRAM has up to now shifted towards bipolar switching devices.

5.2.3 Metal Oxide-Bipolar Non-filamentary ReRAM

Metal Oxide Bipolar Non-Filamentary (MO-BN) ReRAM is a nonvolatile bipolar resistive switching device composed of oxide layers, also referred to as interfacial switching, or non-filamentary. The memory effect has been shown to occur uniformly at (or near) the interface of at least two layers, typically within 2 or 3 nm. One layer is a conductive metal oxide (CMO) which is usually a perovskite [41]. In contrast to filamentary ReRAM devices, the resistance change effect of MO-BN ReRAM is uniform. Depending on materials choice and structure, the current is conducted across the entire or the majority of the electrode area. There is no need for a prior electroforming step. NVM functionality is achieved by the field-driven redistribution of OVs close to the contact resulting in a change of the electronic transport properties of the interface. Oxygen can be exchanged between layers due to the exponential increase in ion mobility with the applied field.

One class of the MO-BN ReRAM includes a deposited ion conductive tunnel layer where a redistribution of oxygen vacancies causes a change of the electronic transport properties of the tunnel barrier. SET, RESET, and read currents scale with device area. In addition, write current is controlled by the tunnel oxide and hence it can be adjusted by changing the tunnel barrier thickness. Both SET and RESET *i-v* characteristics are highly nonlinear, enabling cross-point architectures without the need for an additional select device.

However, MO-BN ReRAM technology is the less mature among all three metal-oxide ReRAM categories. Depending on the material system and structure, demonstrated characteristics include: cycling endurance up to a billion cycles, data retention from days to months at 70 °C, dimensions down to 30 nm, $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ ratios on the order of 10, and sub μA switching currents with read currents in the order of a few nA [42, 43]. The major challenges to be resolved towards the commercialization of MO-BN ReRAM are (i) the improvement of data retention, and (ii) the replacement of Pt electrodes by non-reactive and CMOS compatible materials. More theoretical work is needed to understand the kinetics of programming and retention mechanisms. Once understood, materials have to be chosen to maximize the ratio between SET/RESET and retention times.

The rest of this chapter focuses on MO-BF (VCM) ReRAM where resistance switching corresponds to an abrupt change between a high resistance state (HRS or R_{OFF} state) and a low resistance state (LRS or R_{ON} state), achieved by applying specific voltage to the cell structure.

5.3 Memristive Memory Cell Operation Principles

As mentioned before, in memristive ReRAM the binary data is stored in the form of two resistance states of the memristor, namely R_{OFF} and R_{ON}. Based on the fundamental switching properties of single memristors, this section studies the case of replacing individual memristors with composite memristive switches to be used as cross-point storage structures in crossbar memory arrays. These switches comprise two memristors with opposite polarities connected in a serial (ASM) or a parallel (APM) manner. Depending on their internal state, their polarity, and the device-specific properties (represented in simulation by the values of the parameters of the used model), we provide the basic operation principles for both types of a memristive switch whose properties are exploited to mitigate the sneak-path current impact.

Simulations are conducted using the memristor device model presented in Chap. 2. Figure 5.1 illustrates the response of a single memristor to a AC applied voltage according to the used model. Based on the memory cell hysteresis presented in Fig. 5.1, next we study the composite behavior of APMs and ASMs (a similar description was briefly given in Sect. 4.2 of Chap. 4 as background for the operation of logic circuits). Model parameter values are used as given in $\{\alpha_x, b, c, m, f_o, L_o, V_{RESET}, V_{SET}\} = \{3 \times 10^4, 5, 0.1, 83, 180, 8, -1\text{ V}, 2\text{ V}\}$ and the resulting resistance ratio is set to $R_{OFF}/R_{ON} \approx 10^3$ with $R_{OFF} \approx 2$ MΩ and $R_{ON} \approx 2$ KΩ. We remind here that, according to the mathematical formulation of the model, when $\{a, b\} > 0$ then a positive (negative) voltage applied to the top terminal with respect to the bottom terminal (denoted by the black thick line), tends to decrease (increase) the memristance. Likewise in previous chapters, hereinafter we will again refer to forward (reversely) polarized memristors as FPMs (RPMs).

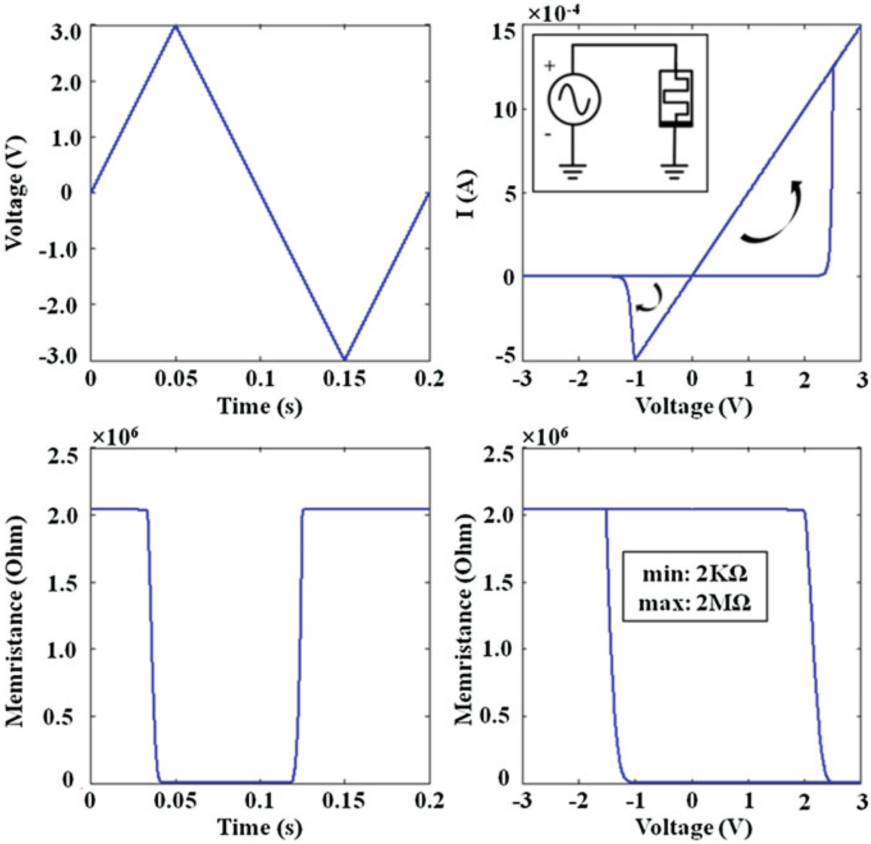


Fig. 5.1 Simulation results from the response of the memristor model to a triangular AC applied voltage

5.3.1 *Anti-serial Memristive Switch (ASM)*

In the anti-serial memristive switch (ASM) concept (also referred in the literature as complementary resistive switch—CRS [21, 44]), a memory cell is formed by two memristors vertically stacked in an anti-serial manner. Compared with individual memristors where binary logic values ‘0’ and ‘1’ can be represented with R_{OFF} and R_{ON} resistances, the unique aspect of ASMs is in using a combination of low and high resistances to represent the same values. Throughout this chapter we will be using the following notation to denote the placement of the in-series devices as top/bottom; hence the ON/OFF combination could represent logic ‘1’ and the OFF/ON could respectively represent logic ‘0’.

Figure 5.2 shows a simulation-based validation of the preferable relation between the V_{SET} and V_{RESET} thresholds for the anti-series and the anti-parallel (discussed later) connections of memristors. The switches are subjected to a triangular AC voltage sweep of appropriate amplitude to make sure that the corresponding voltage drop will cause both memristors to switch states. Three different cases are examined, considering voltage thresholds for both memristive elements set as: (i) $|V_{\mathrm{RESET}}| > |V_{\mathrm{SET}}|$, (ii) $|V_{\mathrm{RESET}}| = |V_{\mathrm{SET}}|$, or (iii) $|V_{\mathrm{RESET}}| < |V_{\mathrm{SET}}|$.

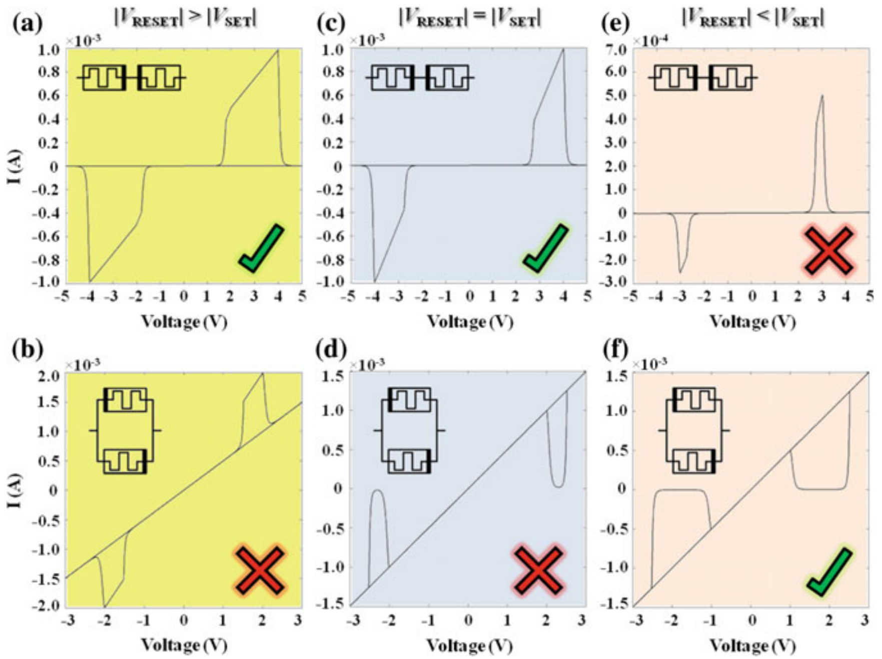


Fig. 5.2 *i-v* characteristics from simulation of ASM and APM configurations under a triangular voltage sweep for three different voltage threshold-sets: **a**, **b** $|V_{\mathrm{RESET}}| > |V_{\mathrm{SET}}|$ with $V_{\mathrm{RESET}} = -2$ V and $V_{\mathrm{SET}} = 1$ V, **c**, **d** $|V_{\mathrm{RESET}}| = |V_{\mathrm{SET}}|$ with $V_{\mathrm{RESET}} = -2$ V and $V_{\mathrm{SET}} = 2$ V, and **e**, **f** $|V_{\mathrm{RESET}}| < |V_{\mathrm{SET}}|$ with $V_{\mathrm{RESET}} = -1$ V and $V_{\mathrm{SET}} = 2$ V

A positive applied voltage to the ASM creates the necessary conditions to either change the state of the RPM (i.e. the lower placed device) from ON to OFF or to switch the FPM state from OFF to ON. Indeed, when voltage reaches a particular point then the FPM switches first (the current rises) until when the RPM switches to the OFF state (the current falls). At this point, the initial state configuration FPM/RPM = OFF/ON has been flipped to ON/OFF. Next, the circuit exhibits an ohmic behavior until the applied voltage exceeds a specific negative threshold where the composite state of the switch is reset to the initial combination.

In a series memristor configuration the switching thresholds cannot be formerly known exactly. The reason is that the devices form a voltage divider; therefore the voltage drop over each element depends on the total external applied voltage, on the current states of the devices, and on their particular switching characteristics. In order to utilize an ASM as a memory cell, starting from its corresponding *i-v* graph of Fig. 5.2, programming and reading voltages need to be optimized. The first must exceed the voltage limits where the state transitions are completed, whereas the latter must be selected within the specific region where presence of high (low) current will determine reading an OFF/ON (ON/OFF) binary state. Here we program the device using ±5 V pulses, whereas reading is done by using pulses whose amplitude falls within the voltage window specified by the limits of the high conduction lobes. However, the width of the reading window depends on the switching thresholds of the memristors. According to Fig. 5.2, the $|V_{\mathrm{RESET}}| > |V_{\mathrm{SET}}|$ or $|V_{\mathrm{RESET}}| = |V_{\mathrm{SET}}|$ cases are preferable for ASM because the reading windows are better defined. Even though in simulation we consider memristors with identical properties (i.e. voltage thresholds, memristance ratio, model parameters), proper selection of devices will be useful to overcome inevitable threshold variability.

Having the ASM preprogrammed to the OFF/ON state, a read pulse will give a high read current because the state of the ASM changes to the intermediate state ON/ON (which is why this has been identified in the literature as a destructive read [45]). Therefore, after reading this state it is necessary to restore the ASM immediately afterwards by rewriting it with a negative programming pulse (here −5 V). Whenever the less resistive series combination ON/ON occurs, there is an instant read current peak which is characteristic of this transition. In order to program the state of the ASM to ON/OFF we apply a positive programming pulse (here +5 V). Then, a reading pulse identifies the state of the cell with a low measured read current since no change is induced to the cell during the read process.

5.3.2 *Anti-parallel Memristive Switch (APM)*

Here we analyze the behavior of two parallel memristors with opposing polarities forming an APM. Likewise with ASM, a combination of low and high resistances is used to represent the stored state. More specifically, the combination {FPM, RPM} = {OFF, ON} could denote logic ‘1’, whereas the opposite combination could denote logic ‘0’. Since the same voltage is simultaneously applied to both memristors, there is no shift in the switching thresholds that dominate their composite behavior; in fact, the switching thresholds are the same with those of the individual memristors (see also Fig. 4.1). However, unlike the series connection, here it is the low resistance (R_{ON}) which dominates the overall resistance of each APM switch.

According to Fig. 5.2b, d, f, in the resulting *i-v* characteristic the current is linear with the applied voltage exclusive of two finite intervals where both devices are found at the OFF state. The APM switch behaves opposite to the ASM switch; the composite memristance is kept low except for two certain windows. The {FPM, RPM} = {OFF, OFF} combination is the intermediate state occurring during the state transitions, likewise happened with the ON/ON combination in ASMs. Proper selection of devices which demonstrate appropriate switching thresholds will correspondingly affect the duration of the reading window. Having $|V_{RESET}| < |V_{SET}|$ seems to be the only viable option in anti-parallel memristors; when $|V_{RESET}| = |V_{SET}|$ both memristors initiate switching simultaneously ($R_{OFF}||R_{ON} \leftrightarrow R_{ON}||R_{OFF}$) and hence there is almost no intermediate switching stage with both of them being OFF. Furthermore, if $|\mathrm{V}_{RESET}| > |\mathrm{V}_{SET}|$ in Fig. 5.2b it can be observed that the APM switch actually behaves like an ASM. This particular choice however, compared to really anti-serially connected memristors, it is certainly well-defined with the composite thresholds being equal to those of the individual memristors. However, it delivers a much smaller resistance ratio between the two distinct stored states, and thus will be less useful when used as a storage cell element.

5.3.3 *Pulse Properties of ASMs and APMs*

Figure 5.3 presents the simulation results of a pulse driven memory cell when comprising either an ASM (a, b) or an APM (c, d) switch. Starting with the ASM pre-programmed as FPM/RPM = OFF/ON, a positive read pulse results in high read current because the internal state of the ASM changes to the intermediate ON/ON state; i.e. a higher portion of the applied voltage drops over the high resistive element as a result of the voltage divider, hence it changes first its state. Afterwards, both memristors remain unaffected since the corresponding voltage drop on each of them does not surpass their switching thresholds. On the contrary, in APMs the {FPM, RPM} = {OFF, OFF} state combination is the intermediate state during the state transitions; the less resistive device here changes first its state towards the high resistive state.

During simulation, first a read pulse is applied to check the state of each memristive switch. In general, such a read pulse must be of appropriate amplitude and duration so as to switch the ASM (APM) to the intermediate ON/ON (OFF/OFF) state, as discussed previously. In our simulations we assume the most convenient relation for the voltage thresholds of each switch, i.e. $|\mathrm{V}_{RESET}| > |\mathrm{V}_{SET}|$ with $\mathrm{V}_{RESET} = -2$ V and $\mathrm{V}_{SET} = 1$ V for ASMs, and $|\mathrm{V}_{RESET}| < |\mathrm{V}_{SET}|$ with $\mathrm{V}_{RESET} = -1$ V and $\mathrm{V}_{SET} = 2$ V for APMs, respectively. We apply read pulses of 3 V to ASMs and 2 V to APMs so as to approximate the centre of the reading

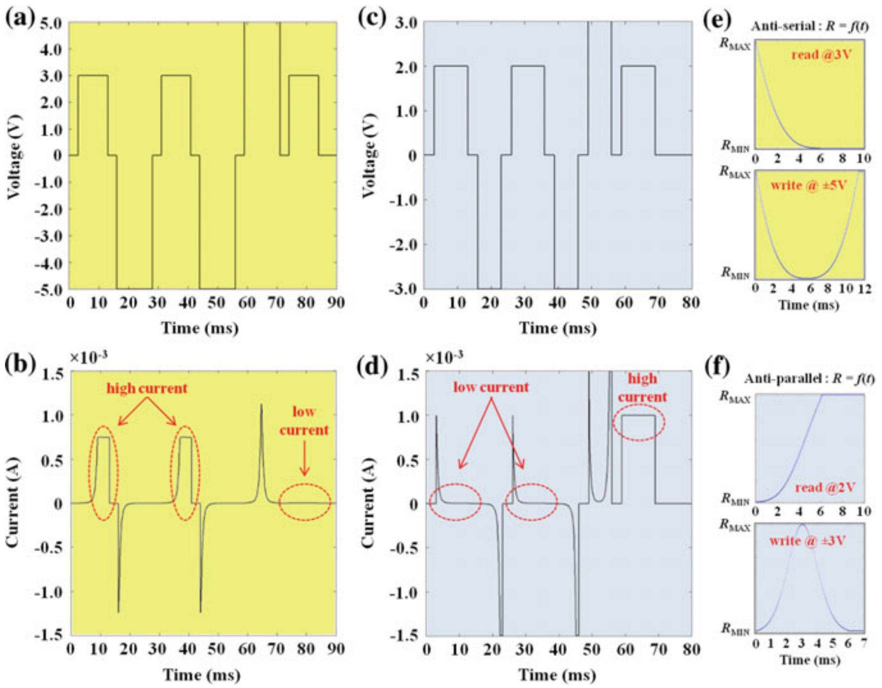


Fig. 5.3 Pulse properties of ASM (*yellow background*) and APM (*light blue background*) switches. The applied voltage pulses are shown in **a** and **c**, whereas the resulting currents are shown in **b** and **d**. After each one of the first two read pulses, since the stored information is destroyed, a write back of the initial state is performed. The change of the composite resistance of ASM and APM switches with time, induced by the applied read and write pulses, is shown in **e** and **f**, respectively

voltage windows and ensure a secure read operation. As shown in Fig. 5.3e, f, the resistance switching for both ASMs and APMs is completed within less than 7 ms. Here the duration of the applied pulses is chosen 10 ms so as to facilitate better distinction of the stored states by observing the read currents in the graphs in Fig. 5.3b, d. As high (low) current is detected, the ASM (APM) is initially found in the FPM/RPM = OFF/ON state. Next, a negative write pulse restores the “destroyed” state of the switches. The amplitude of the programming pulses is ±5 V for ASMs and ±3 V for APMs. Their duration was set to the minimum value which still guarantees a complete transition of the composite memristance, according to Fig. 5.3e, f; in fact, it is 12 ms for ASMs and 7 ms for APMs. Afterwards, the same procedure is repeated, i.e. a read pulse is applied which results in the same current measurements and a successive negative write pulse restores the initially stored state. Finally, a positive write pulse is applied which sets the switches to the {FPM, RPM} = {ON, OFF} combination. The last read pulse results in high current for the APM and in low current for the ASM, which are indicative of reading the aforementioned stored state.

5.4 Sneak-Path Challenge in Memristive Crossbar-Based Memory

This section starts with a brief description of crossbar based memory architectures. Thereafter, the impact of the sneak path on the read margin as function of the memory size is estimated. Some simulation results are given for different cases in order to get more insight in the correlation between the read margins and the memory size, before extending the analysis to include the cases of ASM and APM used as storage elements instead of single memristors.

5.4.1 Fundamentals of Memristive Crossbar Based Memory

Crossbar is probably the most well-known and well-documented memristive architecture in the literature and is among the most promising candidate geometries to implement nonvolatile resistive (memristive) RAM. Owing to their two-terminal structure, memristors can be integrated into crossbar networks, composed of two sets of parallel nanowire-electrodes crossing each other perpendicularly, with a memristive element at each cross-point, as it was also explained in Sect. 4.4.1 of Chap. 4. A schematic representation of such nanoelectronic architecture is shown in Fig. 5.4a. The types of memristive elements that may be used as cross-point storage cells include either a single memristor, or two memristors forming an ASM or APM switch.

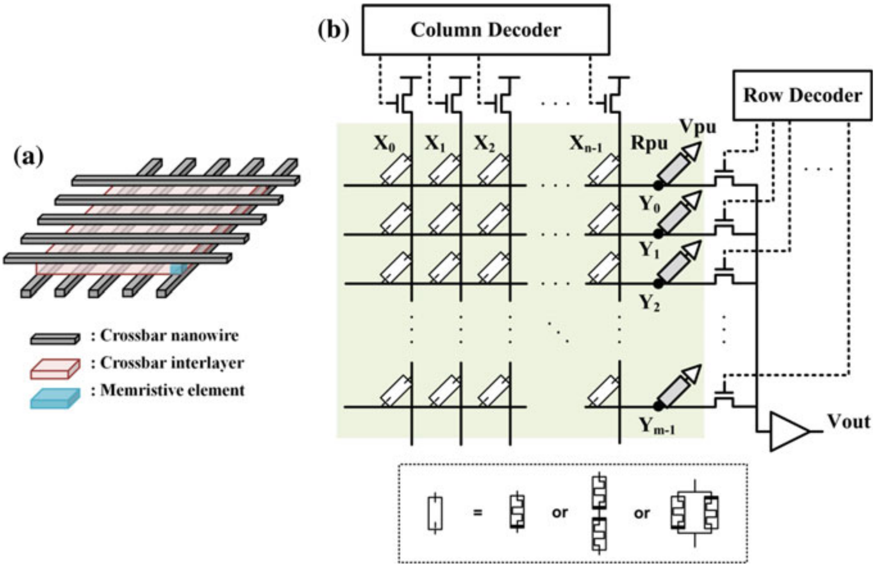


Fig. 5.4 **a** A schematic representation of the crossbar architecture. **b** Basic setup of a nano/CMOS crossbar-based memristive memory system. The *inset* shows all considered options for the cross-point storage cells

In this chapter, the crossbar memory architectures of interest use memristors with highly nonlinear *i*-*v* characteristics and do not rely on any kind of devices (e.g. diodes or transistors) that are normally used to isolate the cell being accessed. The simplest circuit approach for reading information from the memristor-based crossbar, whether it is based on single memristors, ASMs, or APMs, is by applying a certain read voltage across a junction and transforming the current flow into a voltage. Figure 5.4b shows the basic setup of nano/CMOS crossbar memory architecture; here a crossbar with *n* word-lines and *m* bit-lines is assumed. Column and row decoders drive the necessary selection switches in order to form a voltage divider circuit with the corresponding pull-up (sense) resistor and the accessed cross-point (hereinafter also called as "crossbar node"). Typically, the pull-up resistors are implemented in a CMOS layer or in a form of nano-wire resistors [46]. The output of the voltage divider is then driven to a CMOS sense amplifier and the state of the device is distinguished by comparing this voltage to a reference value.

5.4.2 Estimation of Read Margins

In order to perform correct read operations, the voltage swing at the output of the crossbar read circuit, between reading distinct binary stored data in form of different impedance states, should be large enough for the two states to be easily distinguishable. Two different approaches of accessing the crossbar memory are considered: (i) single cell: select one word-line, pull up one bit-line and leave the other bit-lines floating, or (ii) entire word: select one word-line and pull up all bit-lines simultaneously. The circuit-setup corresponding to the read operation of a single cell or an entire word-line, regardless of the cross-point cell type, is shown in Fig. 5.5a, c, respectively. Based on the equivalent circuits of Fig. 5.5b, d the corresponding parasitic worst-case resistance can be computed.

In the ideal reading case, where no current sneak paths are present, the equivalent circuit for the read operation with only one bit-line pulled up is a simple voltage divider formed by a pull-up resistor $\mathrm{R_{PU}}$ and the accessed element, as shown in Fig. 5.6a. Considering a single memristor at each cross-point, then for a given $\beta = R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ ratio the achieved voltage swing ΔV for a certain applied pull-up voltage V_{PU} is calculated as follows:

$$\frac{\Delta V}{V_{PU}} = \frac{V_{OFF} - V_{ON}}{V_{PU}} = \frac{R_{OFF}}{R_{OFF} + R_{PU}} - \frac{R_{ON}}{R_{ON} + R_{PU}} \qquad (5.1)$$

This normalized detection margin of the two possible states of a memory cell is maximized if the pull-up resistor $\mathrm{R_{PU}}$ is optimally chosen to be the geometric mean of the two bi-stable resistances of the memristors. However, for large β the optimal $\mathrm{R_{PU}}$ is close to the less resistive state R_{ON}. In reality, though, when a read operation is performed in the presence of parasitic current paths parallel to the accessed memristor, the effective β substantially reduces.

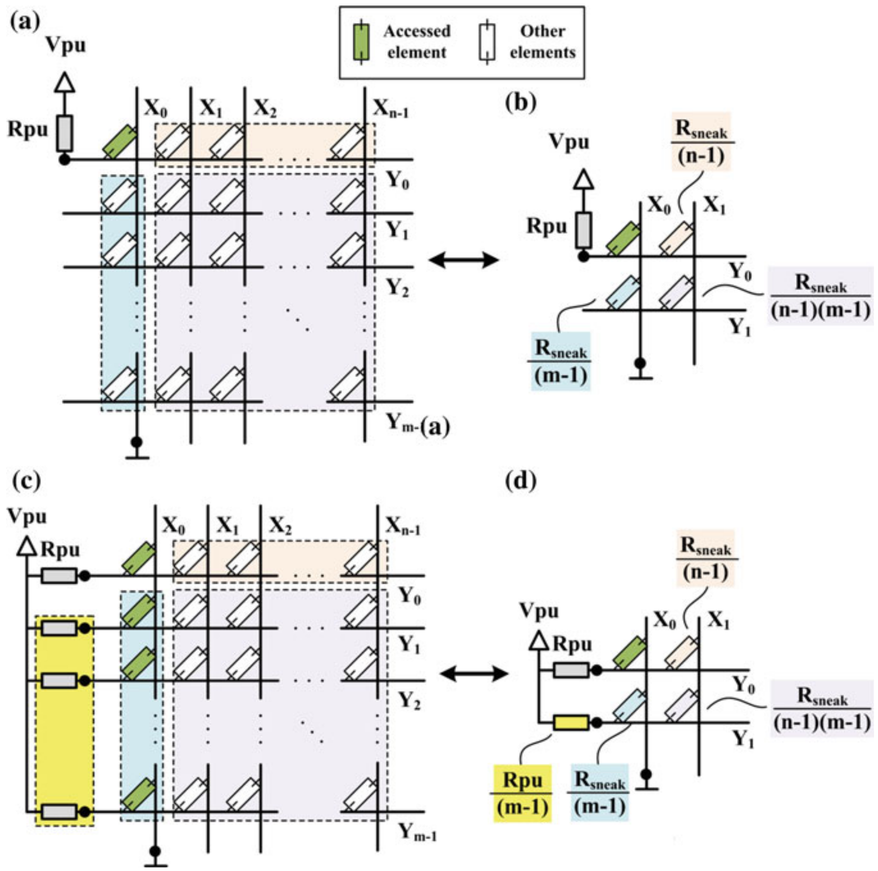


Fig. 5.5 Read operation setup and equivalent circuits in passive crossbar with **a**, **b** one bit-line or **c**, **d** all bit-lines pulled up, when all non-accessed cells are set to the same resistive state given as $\mathrm{R_{sneak}}$

The worst-case reading scenario in case the accessed memristor is in R_{OFF} occurs when the parasitic resistance is as small as possible. This is when all non-accessed memristors are set to R_{ON}, i.e. $\mathrm{R_{sneak}} = R_{\mathrm{ON}}$ (see Fig. 5.5a, b); thus the crossbar output voltage suffers maximum degradation. One could similarly consider the worst case scenario for reading a memristor in the R_{ON} state to be when all non-accessed nodes are in the R_{OFF} state (i.e. $\mathrm{R_{sneak}} = R_{\mathrm{OFF}}$), thus resulting in smaller measured current, although the impact is less severe in this case. According to Fig. 5.6b, the parasitic worst-case resistance is given by the following equation:

$$R_{P,OFF(ON)} = \frac{R_{ON(OFF)}}{(m-1)} + \frac{R_{ON(OFF)}}{(n-1)} + \frac{R_{ON(OFF)}}{(m-1)(n-1)} = R_{ON(OFF)} \frac{m+n-1}{(m-1)(n-1)} \tag{5.2}$$

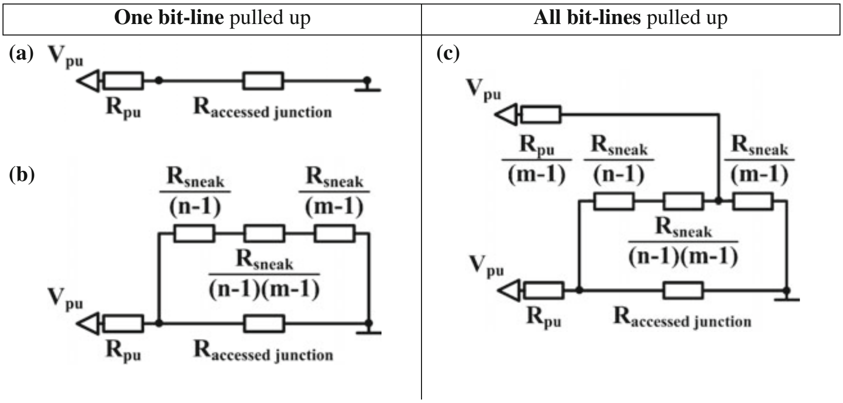


Fig. 5.6 Read operation equivalent circuits in passive crossbar for both memory accessing schemes. **a** The ideal reading case with the accessed memristor forming a simple voltage divider with the pull-up resistor. **b** The realistic case with the inevitable parasitic resistance and **c** the realistic case when accessing entire word-lines instead of only one cell per time

This resistance is connected in parallel with the accessed memristor and, as a result, the maximum achievable read voltage margin gets significantly smaller with increasing crossbar size, as well as it strongly depends on the distribution of the stored information in the array (discussed later).

However, the impact of parasitic current paths depends on the way the crossbar is accessed; i.e. different accessing approaches result in different parasitic resistances. When having all bit-lines pulled up, the parasitic worst-case resistance for any of the accessed elements (of the same word-line) can be computed based on the equivalent circuit depicted in Fig. 5.6c. After applying typical Y-Δ transform, it is found that the equivalent read circuit corresponds to a voltage divider between the *effective* pull-up resistance, given by:

$$R_{PU,eff} = \frac{R_{sneak\_b} \cdot R_{PU}}{R_{sneak\_b} + R_{PU}} \tag{5.3}$$

and the effective *sensed* resistance for either stored binary values, given by the following equation:

$$R_{OFF(ON),sensed} = \frac{R_{sneak\_a} \cdot R_{OFF(ON)}}{R_{sneak\_a} + R_{OFF(ON)}} \tag{5.4}$$

where the auxiliary variables $R_{\mathrm{sneak\_a}}$ and $R_{\mathrm{sneak\_b}}$ are calculated as follows:

$$R_{sneak\_a} = \frac{\left(\frac{R_{ON}}{n-1} + \frac{R_{ON}}{(n-1)(m-1)}\right) \cdot \frac{R_{ON}}{m-1} + \frac{R_{PU}}{m-1} \cdot \frac{R_{ON}}{m-1} + \left(\frac{R_{ON}}{n-1} + \frac{R_{ON}}{(n-1)(m-1)}\right) \cdot \frac{R_{PU}}{m-1}}{\left(\frac{R_{PU}}{m-1}\right)} \tag{5.5}$$

$$R_{sneak\_b} = \frac{\left(\frac{R_{ON}}{n-1} + \frac{R_{ON}}{(n-1)(m-1)}\right) \cdot \frac{R_{ON}}{m-1} + \frac{R_{PU}}{m-1} \cdot \frac{R_{ON}}{m-1} + \left(\frac{R_{ON}}{n-1} + \frac{R_{ON}}{(n-1)(m-1)}\right) \cdot \frac{R_{PU}}{m-1}}{\left(\frac{R_{ON}}{m-1}\right)}. \quad (5.6)$$

Finally, the measurable normalized read voltage margin in this case is:

$$\frac{\Delta V}{V_{PU}} = \frac{R_{OFF,sensed}}{R_{OFF,sensed} + R_{PU,eff}} - \frac{R_{ON,sensed}}{R_{ON,sensed} + R_{PU,eff}}. \quad (5.7)$$

5.4.3 Sneak Path Negative Impact in Readout Performance

For evaluation and comparison purposes, readout performances of several sets of crossbar memory designs are compared in this section. Simulations are based on the device model for memristors presented in Chap. 2 [47, 48]. Nodal analysis is performed and all differential equations are numerically solved using a 4th order Runge-Kutta integration method, as it is implemented in [29]. It is assumed that all memristors inside the crossbar are identical with equal resistance ratio $\beta = R_{OFF}/R_{ON}$. The resistance of interconnects, sensing elements, and voltage source/s is not taken into consideration for two reasons: (i) to reduce the total complexity of the system and minimize simulation run-time, and (ii) because our intension is to perform qualitative analysis and check the trends. For all the design sets, the detection margins are normalized with respect to the applied readout voltage V_{PU} since their values will be always proportional to it.

Figure 5.7a shows the simulation results for a floating memristor array considering different sizes and different β when accessing the leftmost cell of the first row; we choose this particular cell to perform our study following the fundamental analysis in [49]. The maximum achievable read voltage margin gets significantly smaller with increasing array size. Evidently, the noise margins almost vanish quickly as the array size gets larger, regardless of β. The effect of β is more intense when pull-up resistors of optimum values are used. However, the measured voltages strongly depend on the distribution of the stored information in the memory; the overall resistance of the array depends on the number of the stored logic ‘0’ and logic ‘1’ (here R_{OFF} and R_{ON}, respectively), with the power consumption increasing if the resistive elements are mainly found in the low resistive state. Simulation results in Fig. 5.7b indicate how the normalized voltage margins decay fast with increased probability of low resistive junctions when $\beta = 200$. Here the highest of the considered β values was used and the sensing resistors for each array were chosen with their optimum value to maximize the detection margin for the worst-case scenario, i.e. when probability of data ‘1’ approaches 100 %.

Furthermore, we investigate the effect of random data distribution patterns on the measured voltages when reading entire word-lines (array columns). In this context, in order to select and read an m-bit word (in our case a column of the array), we apply m read voltage sources, in series with pull-up resistors, to m rows and connect the

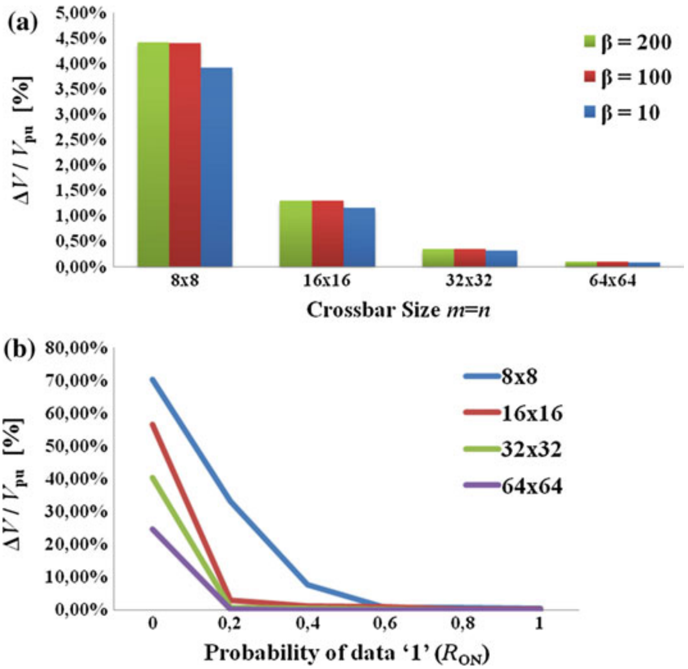


Fig. 5.7 **a** Normalized read margin $\Delta V/V_{PU}$ versus size of an $n \times m$ memristor-based quadratic crossbar ($n = m$) with a stored worst-case pattern for different ratios $\beta = R_{OFF}/R_{ON}$ and $R_{PU} = R_{ON}$. **b** Dependence of read margin on the stored data distribution with $R_{PU} = R_{OPTIMUM}$ and $\beta = 200$

corresponding column to the ground, leaving the rest of the columns floating as shown previously in Fig. 5.5c. The simulation results for a read operation from a column located close to the middle of the grid, both for 32 × 32 and 64 × 64 array sizes, is shown in Fig. 5.8. In each case, the minimum and maximum values from the measured voltages across the selected word-line are given, with $V_{PU} = 1$ V. Apparently, the larger the number of low resistive nodes, the higher the impact on the read voltage margins, regardless of the array-size. We also observe that as the probability of the low resistive nodes approaches 100 %, when moving from smaller to larger arrays, the minimum output voltages almost remain unaffected, whereas the maximum measured voltages continue to decay, thus shortening the resulting ΔV.

In our simulations, given that the resistance of the interconnections is not considered, there is no significant fluctuation among voltages that characterize the same stored state across the word-lines. However, as it was demonstrated in [50], these voltages are expected to slightly decay when moving away from the sources (i.e. towards the middle of the array if the voltage sources are applied on both terminals of each row) depending on the interconnect resistance between adjacent word/bit-lines. The very small variation observed in each graph, though, has as a consequence the resulting ΔV to take a minimum value, i.e. $\Delta V_{MIN} = V_{HIGH,MIN} - V_{LOW,MAX}$.

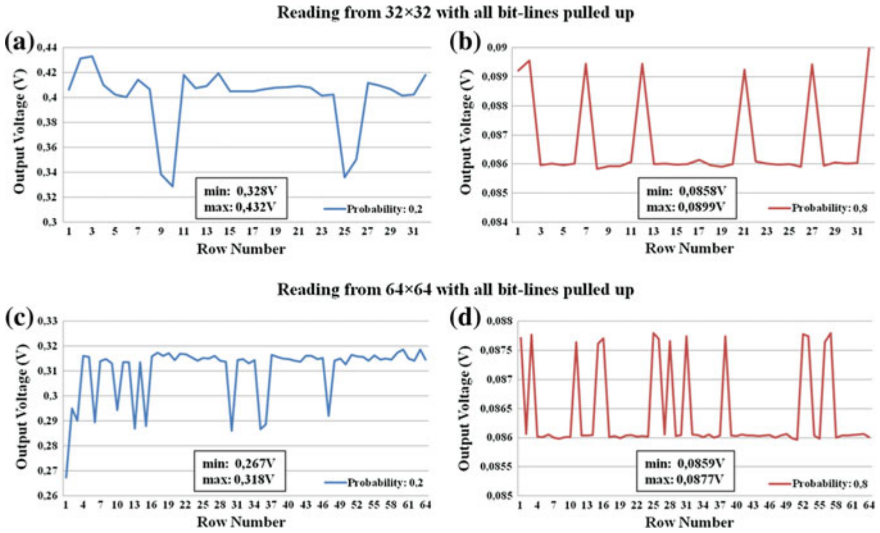


Fig. 5.8 Simulation results for a word-line read operation from **a**, **b** 32 × 32 and **c**, **d** 64 × 64 crossbar arrays considering 20 and 80 % probability for the low resistive nodes across the grid with $R_{\mathrm{PU}} = R_{\mathrm{OPTIMUM}}$ and $\beta = 200$

This ΔV_{MIN} is critical for the correct operation of the memory and will be given much of attention in the rest of this study.

All of the provided simulation results underline that innovative techniques, which will provide us with the opportunity to enlarge importantly the measured voltage margins, thus resulting in more effective read-out memory operations, constitute a key factor towards the practical realization of high-density passive crossbar-based memory systems.

5.4.4 ASM/APM-Based Crossbar Array

In this section we analyze the impact that ASM and APM cross-point devices have on the crossbar-based memory performance. Based on the previously presented equivalent circuits for the read operations in the presence of parasitic current, we perform mathematical analysis and also simulate the ASM/APM-based crossbar to study the current sneak path mitigation.

5.4.4.1 Mathematical Analysis

When having ASMs as cross-point storage cells, the equivalent circuit of a read operation is the same with that of Fig. 5.5, only that now the composite resistance at each node is equal to the sum of the FPM and the RPM resistances, i.e. $R_{\mathrm{OFF}} + R_{\mathrm{ON}}$, regardless of the specific binary stored data. Assuming a high memristance ratio $\beta \gg 1$, then the composite resistance at each node is approximated as $R_{\mathrm{OFF}} + R_{\mathrm{ON}} \approx R_{\mathrm{OFF}}$, hence it is $R_{\mathrm{sneak}} \approx R_{\mathrm{OFF}}$. Therefore, based on Fig. 5.6b and assuming a quadratic crossbar grid ($m = n$) in order to simplify our calculations (without loss of generality), the parasitic resistance found in parallel with the accessed ASM is calculated as follows:

$$R_P = 2\frac{R_{OFF}}{m-1} + \frac{R_{OFF}}{(m-1)^2} = \frac{2m-1}{(m-1)^2}R_{OFF} = \lambda R_{OFF}. \tag{5.8}$$

Parameter λ in Eq. 5.8 only depends on the crossbar size of the quadratic array (m) and its approximate value for the sizes of interest in this study, namely for $m = n = \{8, 16, 32, 64\}$, is respectively calculated as $\lambda \approx \{0.31, 0.14, 0.07, 0.03\}$. The equivalent measured resistance which results by combining the resistance of the accessed node with the parasitic resistance, when the stored state of the ASM during the “destructive” read-out becomes FPM/RPM = ON/ON, i.e. when the composite resistance becomes 2 × R_{ON}, is:

$$\begin{aligned} R_{EQ,1} &= 2R_{ON}||R_P \\ &= \frac{2R_{ON}\lambda R_{OFF}}{2R_{ON}+\lambda R_{OFF}} \xrightarrow{R_{OFF}=\beta R_{ON}} \frac{2\lambda\beta R_{ON}^2}{2R_{ON}+\lambda\beta R_{ON}} = \frac{2\lambda\beta}{2+\lambda\beta}R_{\mathrm{ON}} = \frac{2\lambda}{\frac{2}{\beta}+\lambda}R_{\mathrm{ON}}. \end{aligned} \tag{5.9}$$

Assuming a high ratio β, from Eq. 5.9 we have $R_{\mathrm{EQ,1}} \approx 2 \times R_{\mathrm{ON}}$. This means that the result of reading this state only involves the less resistive state of the memristors and is almost independent of the crossbar size and the stored data distribution within the memory array. Similarly, when the stored state of the ASM is the opposite, i.e. the one which is unaffected during read-out, the corresponding equivalent resistance is:

$$R_{EQ,2} = (R_{ON} + R_{OFF} \approx R_{OFF})||R_P = \frac{R_{OFF}\lambda R_{OFF}}{R_{OFF}+\lambda R_{OFF}} = \frac{\lambda}{1+\lambda}R_{OFF}. \tag{5.10}$$

According to Eq. 5.10, the result of reading this stored state appears to exclusively depend on the crossbar size. In particular, for large crossbar arrays where $\lambda \ll 1$, it is $R_{\mathrm{EQ,2}} \approx \lambda \times R_{\mathrm{OFF}}$. Hence, in this case the measured resistance involves the most resistive state of the memristors. In overall, the resulting ratio between the two equivalent resistances becomes:

$$\frac{R_{EQ,2}}{R_{EQ,1}} = \frac{\lambda R_{OFF}}{2R_{ON}} \xrightarrow{R_{OFF}=\beta R_{ON}} \frac{\lambda\beta}{2}. \tag{5.11}$$

According to Eq. 5.11, since each of the binary states involves different resistive states, then a high ratio β is required since it plays a crucial role in the effective distinction between them during read-out, compensating the effect of the larger crossbar sizes represented by the small values of parameter λ.

Relative to having APMs as cross-point devices, we use again the same equivalent circuit of Fig. 5.6b, only that the composite resistance at each node is now equal to the resulting resistance of the two parallel connected memristors, i.e. $R_{\mathrm{OFF}} \parallel R_{\mathrm{ON}}$, regardless of the particular stored information:

$$R_{ON}||R_{OFF} = \frac{\beta}{\beta+1}R_{ON} \xrightarrow{\beta\gg1} \approx R_{ON} \tag{5.12}$$

As a consequence, for the entire grid it is $R_{\mathrm{sneak}} \approx R_{\mathrm{ON}}$. Therefore, assuming again a high memristance ratio $\beta \gg 1$ and a quadratic crossbar array with $m = n$, then the parasitic resistance connected in parallel with the accessed APM is calculated as follows:

$$R_P = 2\frac{R_{ON}}{m-1} + \frac{R_{ON}}{(m-1)^2} = \frac{2m-1}{(m-1)^2}R_{ON} = \lambda R_{ON}. \tag{5.13}$$

This parasitic resistance, compared to that of Eq. 5.8 which was derived for ASM-based crossbar, it is significantly smaller and involves the less resistive state of the memristors, i.e. R_{ON}. Combined with parameter λ, which gets smaller for larger array sizes, this resistance will quickly reach very small values and thus it is expected to affect dramatically the overall memory function. The equivalent measured resistance, which combines the resistance of the accessed node with the parasitic resistance, when the stored state of the APM during read-out becomes FPM/RPM = OFF/OFF, i.e. when the composite resistance becomes $R_{\mathrm{OFF}}/2$, is:

$$R_{EQ,1} = \left(\frac{R_{OFF}}{2}\right)||R_P = \frac{\left(\frac{R_{OFF}}{2}\right)\lambda R_{ON}}{\left(\frac{R_{OFF}}{2}\right)+\lambda R_{ON}} \xrightarrow{R_{OFF}=\beta R_{ON}} \frac{\lambda\beta}{2\lambda+\beta}R_{ON} = \frac{\lambda}{\frac{2\lambda}{\beta}+1}R_{ON}. \tag{5.14}$$

Assuming a high resistance ratio β, from Eq. 5.14 we have $R_{\mathrm{EQ},1} \approx \lambda \times R_{\mathrm{ON}}$, i.e. the measured resistance results the same with the parasitic resistance, affected only by the crossbar size (parameter λ) but not by the stored data distribution within the array. Similarly, when the stored state of the APM is the opposite, i.e. that which remains unaffected during read-out, the corresponding equivalent resistance is:

$$R_{EQ,2} = (R_{ON}||R_{OFF} \approx R_{ON})||R_P = \frac{R_{ON}\lambda R_{ON}}{R_{ON}+\lambda R_{ON}} = \frac{\lambda}{1+\lambda}R_{ON}. \tag{5.15}$$

Equation 5.15 shows that the result of reading this state, likewise in the case of ASMs, depends only on parameter λ and for large crossbar arrays, where $\lambda \ll 1$, it will be $R_{\mathrm{EQ},2} \approx \lambda \times R_{\mathrm{ON}}$. In this case the measured resistance involves again the less resistive state of the memristors, hence the resulting general ratio between the two equivalent resistances becomes:

$$\frac{R_{EQ,1}}{R_{EQ,2}} = \frac{\frac{\lambda}{\frac{2\lambda}{\beta}+1} R_{ON}}{\frac{\lambda}{1+\lambda} R_{ON}} \xrightarrow{\beta \gg 1 \; \lambda \ll 1} \approx 1. \tag{5.16}$$

According to Eq. 5.16, regardless of the high memristance ratio that the memristors may exhibit, since both binary states involve only R_{ON}, then for large crossbar arrays (i.e. when $\lambda \ll 1$) the effective distinction between the different stored states during read-out becomes impractical.

5.4.4.2 Simulation-Based Verification of Sneak-Path Mitigation

For evaluation and comparison purposes, readout performances of several sets of ASM/APM-based crossbar memory designs are compared in this section. For all the design sets, detection margins are normalized with respect to the applied readout voltage V_{PU}.

Figure 5.9 shows the simulation results for the floating memristor array while considering either of the studied cross-point memristive solutions. Model parameter values are as given in $\{\alpha_{\text{x}}, b, c, m, f_{\text{o}}, L_{\text{o}}\} = \{5 \times 10^4, 0, 0.1, 82, 310, 5\}$, whereas the limiting memristance values are selected such that $R_{\text{ON}} \approx 2$ KΩ and $R_{\text{OFF}} \approx \{20, 200, 400, 600\}$ KΩ, respectively. The latter result in four different memristance ratios $\beta = \{10, 100, 200, 300\}$. Moreover, the threshold voltages for ASMs and APMs are set in such a way so as to result in a wider (hence clearer) reading window, as previously shown in Fig. 5.2. In all read-out measurements we assume common values for the pull-up resistors, namely we set $R_{\text{PU}} = 2 \times R_{\text{ON}}$ for ASMs (i.e. equal to the less resistive state of an ASM) and $R_{\text{PU}} = R_{\text{ON}}/2$ for APMs (i.e. equal to the less resistive state of an APM), respectively. Also, different array sizes and/or different β affect the measured equivalent resistance. Therefore, taking into consideration the in-series sense resistor R_{PU} as well as the composite switching behavior that ASM and APM devices exhibit when studied inside the crossbar array (likewise we did for the standalone devices and presented in Fig. 5.3), we selectively adjust the amplitude and the duration of the applied reading pulse V_{PU} in order to achieve the largest possible voltage margins. Table 5.2 summarizes the used V_{PU} pulse characteristics for each simulation scenario. We observe that, although the necessary pulse duration seems comparable, there is a huge difference between the voltage amplitudes; unlike ASMs, APMs require much higher operation voltages which increase with the array size regardless of the β ratio. Of course, the absolute values of the voltages presented in Table 5.2 are used only in the context of this survey; it is not good practice (if not unacceptable) to use such high voltages for any concrete memory application or nonvolatile embedded memories.

In specific, the voltage margins for different distributions of the stored information in the memory array are calculated, for different grid sizes and for different β.

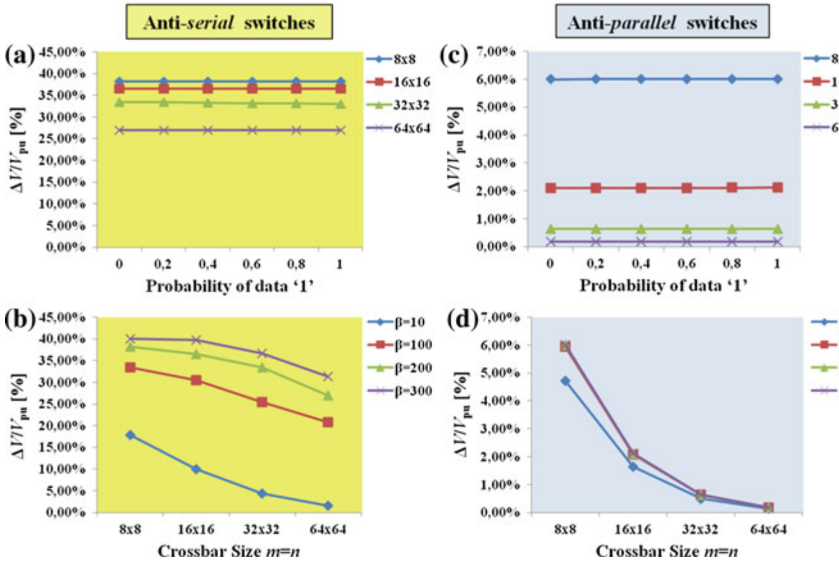


Fig. 5.9 Normalized read margin $\Delta V/V_{PU}$ for ASMs and APMs. **a**, **c** $\Delta V/V_{PU}$ versus the stored data distribution in the grid for four different array sizes with up to 64 × 64 elements. **b**, **d** $\Delta V/V_{PU}$ versus crossbar size (quadratic array where columns = rows) for four different $\beta = R_{OFF}/R_{ON}$ ratios. We consider reading one cross-point element/time and assume $R_{PU} = 2 \times R_{ON}$ for ASMs and $R_{PU} = R_{ON}/2$ for APMs, respectively. Graphs presented in subfigures **a** and **c** correspond to $\beta = 200$

With the used voltage scheme we make sure that the corresponding voltage drop on the target cell approximates the middle of the reading window. Voltage-drops on all non-accessed cells are limited to values below the threshold voltages of the reading window for both serial and parallel memristive cross-point configurations; thus these storage cells are not affected during read-out. Our calculations, without loss of generality, neglect the word and bit line resistance R_{LINE}, which in general should be small compared to R_{ON} in order to be able to operate large crossbar arrays (for functional arrays, optimization between the array size and the R_{ON}/R_{LINE} ratio has to be worked out). As shown before in Fig. 5.7, in passive memristive crossbar arrays the noise margin almost vanishes very quickly as the array size gets larger regardless of the memristance ratio, whereas the measured voltages strongly depend on the distribution of the stored information in the memory. In this context, as shown in Fig. 5.9a, c, ASMs and APMs could efficiently address the sneak path problem since they exhibit measured resistances which are independent of the stored data distribution within the memory array. However, although ASM-based arrays exhibit high enough noise margins, this is not true for APM-based arrays where normalized read margins not only are lower but also decay with increased array sizes faster than in ASM-based architectures. Furthermore, in Fig. 5.9b, d we notice the minor effect that high β values have on the APM-based array performance. In fact, examining

Table 5.2 V_{PU} pulse characteristics for ASM/APM based crossbar memory

Ratio β	Anti-*serial* memristors (ASMs)					Anti-*parallel* memristors (APMs)				
	Amplitude (V)				Duration (ms)	Amplitude (V)				Duration (ms)
	8 × 8	16 × 16	32 × 32	64 × 64		8 × 8	16 × 16	32 × 32	64 × 64	
10	3.7	5	8	14	2.4	5	8	14	25	2.4
100	2.2	2.4	2.7	3.4	7.2	4	6.5	12	22	8.4
200	2.1	2.2	2.4	2.7	12	4	6.5	12	22	13.2
300	2.1	2.2	2.3	2.5	18.6	4	6.5	12	22	20.4

each array size separately, in ASMs seemingly there is some improvement in the voltage margin when moving from lower to higher memristance ratios, whereas in the APMs case there is no significant change; indeed for larger arrays there is no evident difference at all.

5.4.5 Alternative Crossbar Topologies

This section discusses alternative topologies for passive crossbar ReRAM as means to deal with the sneak-path problem [26]. In such alternative topologies, a certain percentage of insulating nodes are spread out inside the array according to specific distribution patterns. The motivation is to restrain current sneak-paths and thus improve the voltage margins by replacing some memory cells. Such a practice is considered a viable solution given the huge device density that the crossbar geometry offers compared to other circuit architectures. Figure 5.10 shows five possible alternative topological patterns where insulators are placed between mutually perpendicular wires, together with the full memristive crossbar (X-bar). The details for each pattern are listed below:

- *Columns pattern*: the insulating junctions are located in columns which are uniformly distributed across the array.
- *Rows pattern*: the insulating junctions are placed in uniformly distributed rows.
- *Columns and Rows pattern*: this combines the previous two mentioned patterns.

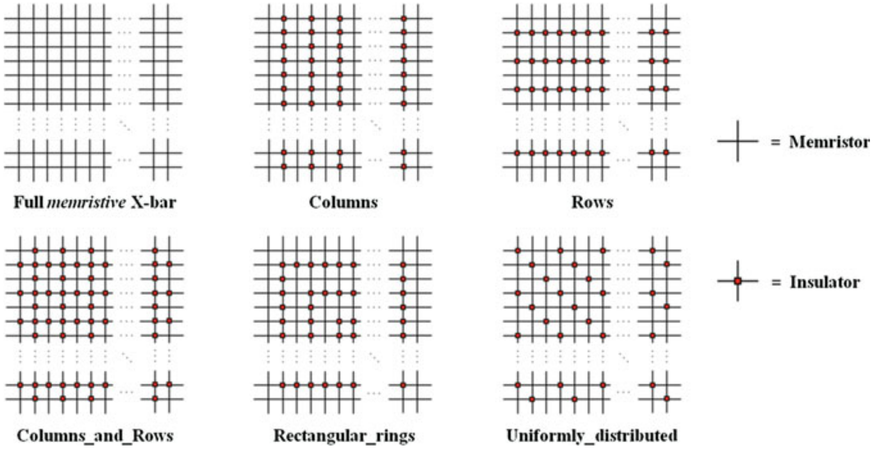


Fig. 5.10 Alternative crossbar architectures (topologies) compared to the full memristive crossbar. *Red dots* denote insulating cross-points whereas simple wire crossings denote memristive memory cells

- *Rectangular Rings pattern*: the inserted insulating nodes are placed in rectangular rings which are distributed across the grid starting from the central rectangle which is formed by the four innermost nodes.
- *Uniformly Distributed pattern*: the insulating nodes are uniformly distributed both horizontally and vertically inside the grid. For each insulator, the closest neighboring insulating nodes are always found at equal horizontal and vertical distances.

Each of the above patterns aims to uniformly cover as much as possible the entire grid area; the grid is considered a “torus” to facilitate the distribution of the insulating junctions.

5.4.6 *Simulation-Based Evaluation of Alternative Topologies*

In order to investigate the impact of such topologies, several simulations were performed for different grid sizes and by considering different populations for the total introduced insulators. Simulation of these new architectures was based on the memristor device model which was presented in Chap. 2 with the model parameter values as given previously in Sect. 5.4.4.2 and ratio $\beta = 200$. The sense resistor R_{PU} was set to the optimum value to yield the highest possible output ratio. The presented patterns were tested for both 32 × 32 and 64 × 64 arrays and their performance was compared to that of the full memristive crossbar, while considering three different distributions characterizing the total number of insulators in the grid; these are 10, 25, and 50 % of the total number of nodes. Once all data were programmed in the array, the read operation was performed by applying a 1 V pulse across the target cells for different binary stored values. Then, the resulting voltage margin was calculated.

5.4.6.1 Reading One Memory Cell: Worst-Case Scenario

Figure 5.11 presents the read voltage margin for the worst-case reading scenario; in each sub-figure the voltage margin ($\Delta V/V_{\mathrm{PU}}$) is normalized to the reference value which is always the performance of the full memristive crossbar (i.e. the array with 100 % memristive nodes). In addition, the voltage margins of smaller arrays, resulting by removing the insulating junctions from the simulated grid, are also included; e.g. Figure 5.11a shows that the read margin of the simulated full memristive 32 × 32 grid is improved by 14 % when its size is reduced by 10 %. However, maintaining the same grid size while replacing 10 % of the nodes with insulators increases the voltage margin up to 31 % depending on the architecture. All architectures yield improved read margin which is never equal to the margin

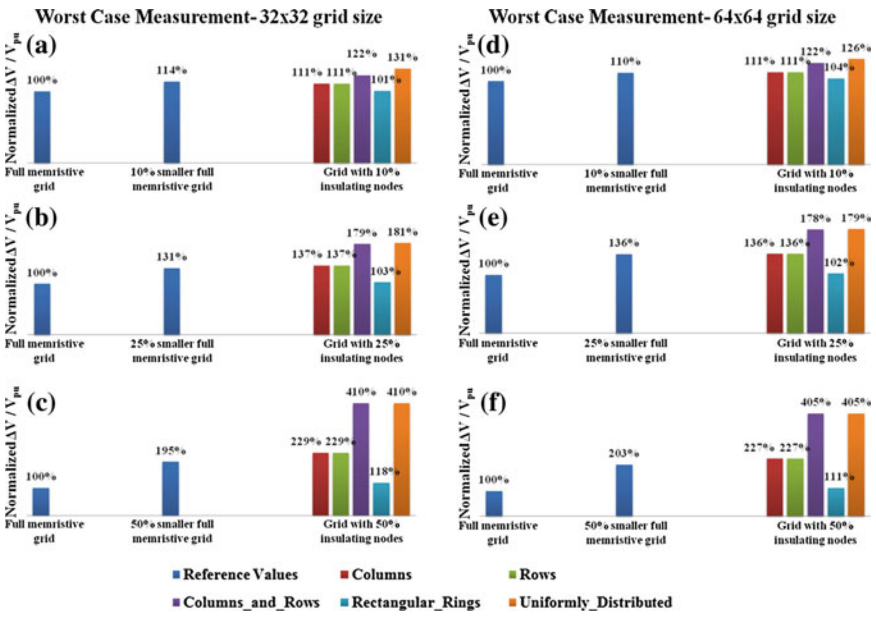


Fig. 5.11 Normalized worst-case read voltage margin $\Delta V/V_{PU}$ for **a**–**c** 32 × 32 and **d**–**f** 64 × 64 crossbar arrays. The $\Delta V/V_{PU}$ is normalized to the voltage ratio of the full memristive grid. The achieved margins of the applied patterns are also compared to the margins of smaller arrays, resulting by removing the insulating junctions from the simulated grid. The performance of the patterns with 10, 25 and 50 % insulators is compared to the performance of memristive grids with 10, 25 or 50 % less nodes. The five patterns are designated with *red* (Columns), *green* (Rows), *purple* (Columns & Rows), *cyan* (Rectangular Rings), and *orange* (Uniformly Distributed)

corresponding to the smaller memristive grid which comprises the same total number of memory cells. Depending on the applied architecture, the percentage of insulating nodes, as well as the percentage of low-resistive cross-point cells, the exact values of the improved read margins for the examined array-sizes fall within the range [2, 440] mV. Overall, inspecting the simulation results of Fig. 5.11 reveals the following:

- All of the five architectures improve the read margins, although the strength of the improvement is strongly architecture-dependent. The improvement trends for the architectures are the same irrespective of the simulated grid-sizes; improvement of the voltage swing is kept when moving to larger quadratic memory crossbar arrays.
- Columns and Rows pattern-based architectures result in similar improvements for all simulated cases. Combining these two patterns results in better improvement; the higher the percentage of inserted insulators, the higher the improvement. However, this improvement difference seems to be grid-size independent. Moreover, it implies the introduction of a much larger number of

insulating nodes which particularly approximates the percentages 17, 43, and 75 % for each one of the considered categories, respectively.

- Rectangular Rings-based architecture performs the worst, while the Uniformly Distributed-based architecture scores the best, irrespective of the simulated case. This can be easily explained. Having insulators is essentially like having R_{OFF} cells (with larger R_{OFF} though); hence, the more uniformly distributed these are, the less the sneak current, which in turn results in better read margins.

In addition to the above analysis, we studied the sneak current distribution during random worst-case read operations performed to a 32 × 32 array; a total of 2000 read operations were performed. Each time we measured the sneak current at all non-accessed nodes. The total accumulated currents were finally computed and normalized to the maximum noticed value. Figure 5.12a presents the normalized sneak current distribution whereas Fig. 5.12b shows the read-out event distribution in the array. It can be observed that the majority of the sneak current is uniformly distributed in the grid (nodes marked with purple color). Consequently, the architectures which spread as much as possible the insulating nodes rightfully suppress more effectively the sneak currents. The pattern which less complies with this policy is the Rectangular Rings pattern.

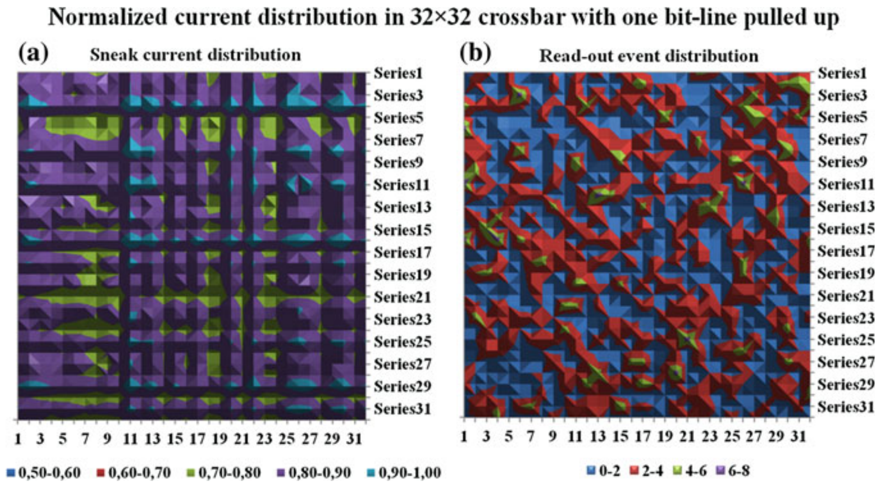


Fig. 5.12 Simulation results for the sneak-current distribution analysis of the worst-case scenario when accessing one bit per memory operation in a crossbar with all of its nodes initialized at R_{ON} except for the node being accessed. **a** Shows the accumulated sneak current at each node normalized to the maximum measured value. The five distribution classes are given in *blue* (0.5–0.6), *red* (0.6–0.7), *green* (0.7–0.8), *purple* (0.8–0.9), and *cyan* (0.9–1.0) color, respectively. **b** Shows the distribution of the read-out events in the grid which were performed randomly based on the uniform distribution. The four distribution classes are given in *blue* (0–2), *red* (2–4), *green* (4–6), and *purple* (6–8) color, respectively. In order to maximize the output we set $R_{\mathrm{PU}} = R_{\mathrm{OPTIMUM}}$ with $\beta = 200$

5.4.6.2 Reading Entire Word-Lines: Random Stored Data Distribution

To evaluate the performance of the five architectures when reading the entire word-line (we remind that here a word-line corresponds to a column of the array), we performed statistical analysis of their behavior for a series of read operations over different random initializations of the array.

Our intention was to observe how the read margins changed depending on the array size and the stored data. We created a referencing database by collecting information from 30 read-out events of the same word-line (located close to the middle of the grid), each for a different random memory initialization. The same procedure took place for each one of the two probabilities for the low resistive nodes. For example, in a grid with 20 % probability for R_{ON} nodes, in every word-reading most likely there were 80 % R_{OFF} nodes and 20 % R_{ON} nodes (i.e. approximately 26 R_{OFF} and 6 R_{ON} nodes in a 32-bit word-line). After each access operation, the maximum (greatest) and the minimum (smallest) of the measured read voltage differences between (i) the voltage of R_{OFF}-read (V_{OFF}), and (ii) the voltage of R_{ON}-read (V_{ON}), were calculated as follows: $\Delta V_{MIN} = V_{OFF,MIN} - V_{ON,MAX}$, and $\Delta V_{MAX} = V_{OFF,MAX} - V_{ON,MIN}$. Finally, the average of the 30 values of ΔV_{MIN} and ΔV_{MAX} was determined. ΔV_{MIN} is the most critical, though, denoting whether the distinction of different stored states is still possible.

Once we created the referencing values, we incorporated the proposed patterns in the memory arrays and repeated the process. Reading was performed on the same column as before and, depending on the applied pattern, the specific insulating junctions which were included in the target word-line were omitted. However, the higher the % of insulators, the fewer the available memristive nodes to read in each word-line (e.g. for the Rows pattern with 50 % insulators there are only 16 remaining memristive nodes in each word-line, thus an estimated 3 R_{ON} and 13 R_{OFF} nodes to read from). Within such a small sample, depending on the stored data probability we sometimes accidentally had only R_{OFF} or only R_{ON} nodes in the target columns after omitting the insulating junctions. Therefore, whenever the target word-line happened to include mostly insulators (e.g. in the Columns or Rectangular Rings pattern), we instead performed the read operation to an adjacent column in order to improve the sample of the measurements.

Figure 5.13 shows the simulation results for 32 × 32 and 64 × 64 memory arrays, respectively; the change of the voltage margin is normalized to the reference value, which represents the case where no insulating junctions are inserted in the array. Since the worst-case array pattern is different when different reading schemes are utilized, we notice that some patterns demonstrate different behavior compared to what they did for the bit-reading approach. For example, the Columns pattern (in which the insulating nodes are arranged in accordance with the way the word-lines are read) performs better. Overall, we observe the following:

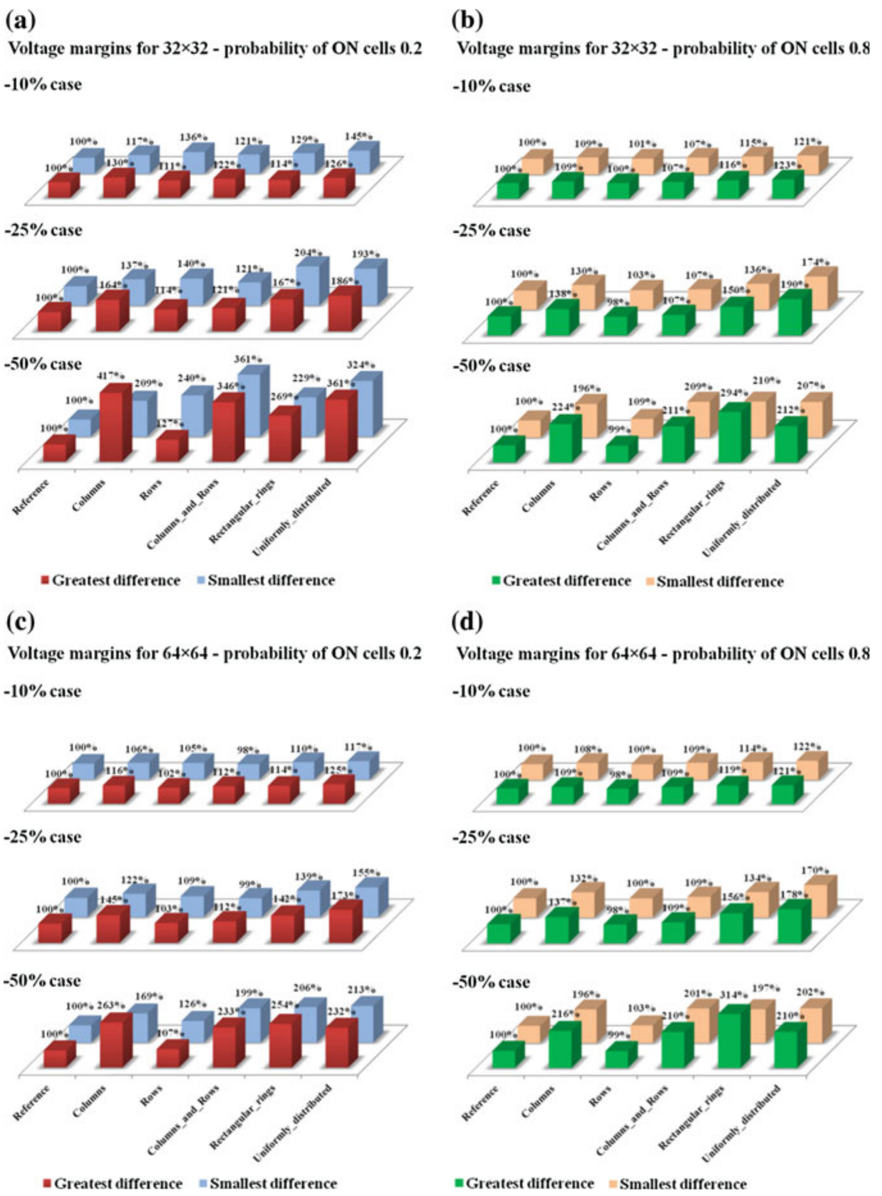


Fig. 5.13 Evaluation of the average patterns' performance on the greatest and smallest measured ΔV over a series of 30 different read operations in **a**, **b** 32 × 32 and **c**, **d** 64 × 64 grids. The considered probabilities for the stored low resistive states across the grid are **a**, **c** 20 % and **b**, **d** 80 %. In order to maximize the output voltages, we assume $R_{\mathrm{PU}} = R_{\mathrm{OPTIMUM}}$ and set $\beta = 200$

- All of the five architectures improve the read margins irrespective of the array size and the occurrence probability of low resistance nodes in the array, except for Row-based architecture in some cases; e.g. when occurrence probability of low resistance nodes is 80 % and percentage of inserted insulators in the grid is 25 or 50 %. However, such results are attributed to the sample of measurements because for high % of inserted insulators there were only few remaining memristive nodes to read.
- The strength of the improvement is not only strongly architecture dependent, but also strongly dependent on the metrics (greatest versus smallest), and the percentage of inserted insulators. However, the dependence on the array size seems very marginal.
- The worst architecture when considering ΔV_{MAX} metric (greatest difference) is the Row-based for all simulated cases. However, when considering ΔV_{MIN} metric (smallest difference), it becomes case-dependent; it is the Row-based architecture for large occurrence probability of low resistance nodes, and can be Row-, or combined Row-Column based architecture for low occurrence probability of low resistance nodes.
- The best architecture is Uniform-based for most of simulated cases. However, in some cases Column and Rectangular-ring based perform (slightly) better.

An interesting remark has to do again with the performance of the patterns when moving from smaller to larger arrays. We notice that improvements over the voltage margins attenuate in an indefinite manner when the probability of R_{ON} junctions is low. On the contrary, the induced improvements are maintained when the probability of R_{ON} junctions approximates 100 %. Hence the impact of the alternative architectures is more evident when approaching the worst-case reading scenario.

Additionally, we performed similar experiments as in the previous section in order to analyze the sneak current distribution during random read operations. We performed read-outs of all word-lines and computed the mean value of the accumulated parasitic current flowing through each node. However, the current flowing through the less resistive nodes is always much higher; hence for each probability we took into consideration only the nodes which are in majority within the array; e.g. when having 20 % nodes in R_{ON} we omit the current flowing through them and normalize the results to the minimum of the mean values. Figure 5.14 shows the simulation results for a 32 × 32 array; the most congested nodes of the grid are found along specific rows located close to the center and the borders of the array. Such distribution is more evident when the probability of the low resistive nodes increases. Therefore, the architectures which position the insulators along the columns of the array are more likely to replace critical (congested) nodes and, consequently, contribute more to the lowering of sneak currents. This is very much in line with the simulation results showed in Fig. 5.13, which clearly show that the Column-based architecture overall performs very well.

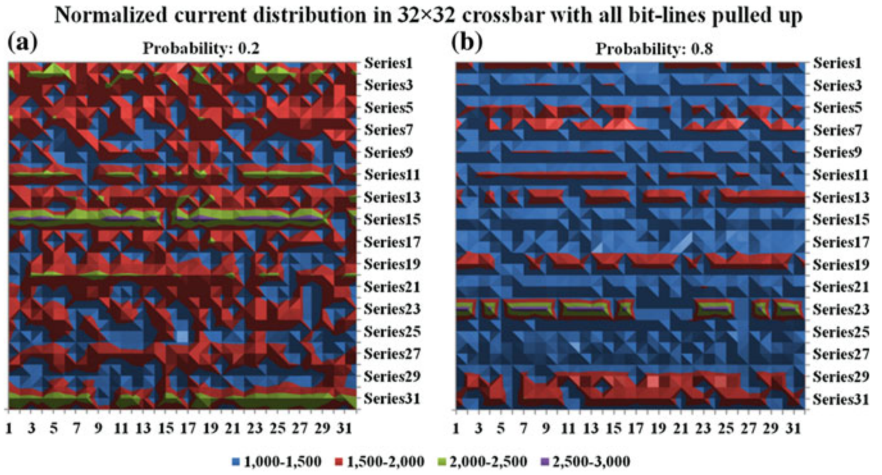
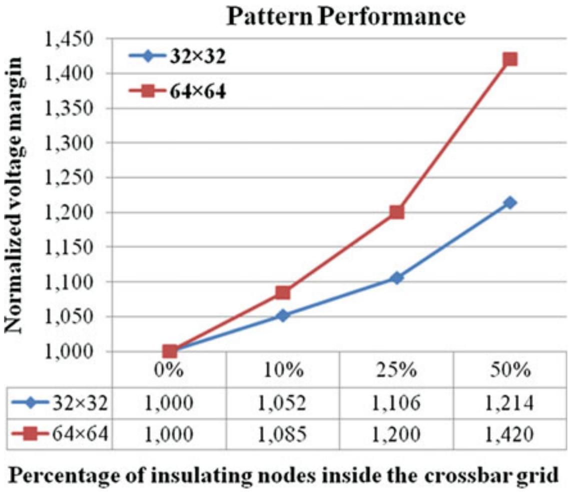


Fig. 5.14 Simulation results of the sneak-current distribution analysis when accessing one word-line per memory operation on a 32 × 32 crossbar initialized randomly with **a** 20 % or **b** 80 % of its nodes at R_{ON}. After initialization we read all columns and calculate the mean value of the amount of parasitic current corresponding to each node. In **a** we show the accumulated sneak current at each node normalized to the minimum mean value when taking into consideration only the nodes found at R_{OFF}, whereas in **b** the mean values are normalized to the minimum mean value when taking into consideration only the nodes found at R_{ON}. The four distribution classes are given in *blue* (1.0–1.5), *red* (1.5–2.0), *green* (2.0–2.5), and *purple* (2.5–3.0) color, respectively. In order to maximize the output, we assume $R_{PU} = R_{OPTIMUM}$ and set $\beta = 200$

5.4.7 Application of Alternative Topologies to ASM-Based Crossbar

We applied the most efficient alternative topology for the bit-reading approach to ASM-based crossbar arrays, since the latter proved to address better than the APM-based the sneak path problem. We evaluated the performance of the Uniformly Distributed pattern by comparing the worst-case read voltage margins with those of the pure ASM-based array. In simulation we used the model parameter values and the voltage pulsing characteristics as previously mentioned in Sect. 5.4.4.2 for a memristance ratio $\beta = 200$, whereas the resistances of interconnects, sensing elements, and voltage source/s were not taken into consideration. The read voltage margins were normalized to those of the array without insulators. Both 32 × 32 and 64 × 64 memory arrays were studied and the read-out operation was performed to the leftmost cell of the first row, likewise in the bit-reading simulation of typical memristive crossbar.

In the simulation results of Fig. 5.15 we observe that the new read margins are improved incrementally when increasing the percentage of the inserted insulating nodes and the notable improvement reaches up to 21 % (42 %) for 32 × 32 (64 × 64) arrays when half of the existing nodes are insulators. An interesting



	0%	10%	25%	50%
32×32	1,000	1,052	1,106	1,214
64×64	1,000	1,085	1,200	1,420

Fig. 5.15 Application of the Uniformly Distributed topology to an ASM-based crossbar array. The worst-case normalized read voltage margin for 32 × 32 and 64 × 64 arrays is shown. The tested pattern introduces 10, 25 or 50 % insulators and its performance is compared to the performance of the ASM-based grid with 0 % insulating nodes for a memristance ratio $\beta = 200$

observation concerns the resulting ratio for corresponding % of insulated nodes in the two studied array sizes. Particularly, for each one of the three examined %, doubling the size of the side of the square crossbar array results in (almost) two-fold rate of improvement in the read margin. Therefore, for *n* times larger quadratic crossbar arrays, for the same % of insulators, the read margins are improved almost *n* times. Of course, the larger the % of insulators, the larger the improvement in the read margins. However, in comparison with the results shown in Fig. 5.11, we observe that in ASM-based memory the gains in array-performance do not scale in a similar manner with the % of inserted insulators as they did for typical memristive crossbar arrays; though, they still contribute to the addressing of the parasitic conducting problem.

5.4.8 Overview and Discussion

Many challenges need to be solved before memristor-based memory really takes off; examples are: endurance, yield, reliability, robust design and architecture. Some important technological aspects and design constrains were highlighted in this chapter, whereas different crossbar architectures for high density storage were presented. Although some assumptions were made to facilitate and accelerate the simulation-based validation of the impact of the alternative crossbar topologies (e.g. neglecting the resistance of interconnects), the results clearly show that considering the injection of insulator-patterns in the crossbar is a valid and interesting approach to be considered for reducing the impact of the sneak path problem. Following this approach, significant improvements over the measured voltages throughout the crossbar array can be accomplished provided that a certain percentage of the crossbar memory cells is replaced by insulating junctions (or its dual form: when the given memory cells are arranged in a sparser configuration). This unfortunately comes at the price of additional area overhead. However, given the fact that (i): crossbar architecture typically will target huge data storage, (ii) nano-devices integrated in the crossbar are extremely small (size $4F^2$ where the feature size $F < 10$ nm [2]), and (iii) the regularity of the structure (easy to manufacture), it is very justifiable to trade additional array area for better reliability and robustness; keep in mind that the proposed approach eliminates the requirement for select-devices at each cross-point as well. It is worth noting that reliability is one of the major bottlenecks that technology scaling is facing, leading to lower yield, thus preventing cost per die from further scaling.

The provided simulation results indicate that the choice of the best architecture will strongly depend on the memory size, the memory addressing mode, and the impact of the access circuitry (decoders, sense amplifiers, etc.). Obviously the architectures which spread the insulating nodes uniformly across the array perform steadily well and can yield good improvements for either bit- or word-reading modes. However, a designer might choose the topology which has less impact on the access circuitry, as well as on the memory addressing mode. In this context, addressing an array with Rectangular Rings topology, from an architectural point of view, might present a challenge since, depending on which column we are reading from, we might get different number of bits read in each operation. On the contrary, there is no such complexity overhead in arrays with e.g. Columns or Uniformly Distributed topologies, where efficient row-decoding can be implemented by appropriate combinational circuitry. Moreover, it is worth noting that the proposed architectures could be useful in large hierarchical memory organizations where a memory lattice is divided into a number of smaller-sized sub-arrays in order to maintain sufficient voltage margins; the necessary sub-arrays will be less in number and larger in size, thus reducing the overhead of extra peripheral circuits.

As already explained, the provided architectures require the injection of insulator patterns in the crossbar. From manufacturing point of view, this will require additional steps in the fabrication process and modification of the masks. Nevertheless, the regularity of the insulator patterns will make it easier to be integrated both in the masks and in the fabrication process. As mentioned before, the current mainstream in nonvolatile memories is Flash memory. 2D NAND-type Flash has already scaled to 16 nm node, whereas scaling to near 10 nm seems possible [2]. Further density scaling, though, may require a different memory technology and/or a 3D architecture, and 3D NAND Flash is currently being developed [51]. Current projections for the achievable packing density (bits/cm^2) of ReRAM remain substantially lower than those for 3D NAND Flash, unless 3D ReRAM is fabricated [9]. However, compact, bipolar cell selection devices with scalability below 10 nm, high ON/OFF ratio, and high-endurance, are highly required to cut off the leakage paths in the z-plane of 3D ReRAM. Indeed, the higher the number and the size of the stacked 2D layers, the higher ON/OFF ratio is needed. Even though the presented architectures were not tested for multiple stacked 2D arrays, they are expected to somehow “relax” the aforementioned tight requirements for the 3D selection devices by mitigating the leakage paths at each 2D stacked layer separately.

5.5 XbarSim—An Educational Simulation Tool for Memristive Crossbar-Based Circuits

Given the considerable amount of attention that memristors and memristive circuits have gained in the recent years, introducing these elements and their applications to the next generation of physicists and engineers is essential. Additional theoretical discussion as part of a regular circuit theory course would be of great value. Of course, a “hands-on” experience would certainly provide an added educational benefit, but memristive systems are not yet commercially available and even when some of these will become, they may require cautious handling, thus will most likely have a limited time-span in the laboratory. On the other hand, properly developed simulation environments, which incorporate memristor device models and enable the experimentation with novel circuits and architectures, will complement the theoretical teaching and will provide young students and researchers with the necessary experience. The potential users will learn more effectively the fundamental device properties and will realize several emerging circuit design constraints for such novel technologies. Simulation has been extensively used in the study of emerging nanoelectronic circuits and architectures in the past [52–57]. Similarly, simulation is expected to become an indispensable educational and research tool for studying memristive circuits, architectures, and developing new circuit design paradigms.

This section describes a project to create a novel design and simulation tool for standard/alternative memristive crossbar architectures, targeting memory and/or logic applications. The objective of the project, namely XbarSim (short for X-bar Simulator), is to create an easy to use simulation and layout tool available freely to the research community. Today’s students/researchers and inexperienced circuit designers require a rapid and accurate simulation and design layout tool to determine the functionality of crossbar-based circuits with memristors. XbarSim is the product of an ongoing effort to create an educational simulation tool which will provide the ability to quickly layout a memristive circuit design on a crossbar topology. Besides being still in its infancy, a large amount of effort is being put forth to enhance its functionalities and create a large, useful toolbox for future students, researchers, and engineers.

XbarSim facilitates the design and simulation of nanoelectronic circuits with memristors based on the crossbar architecture. It is a self-contained, platform-independent simulation tool which was developed using the JAVA programming language through the Easy Java Simulation (EJS) environment [29], aiming to serve students/researchers who wish to explore and study memristive circuits mapped on the crossbar circuit architecture. More specifically, XbarSim models a two dimensional array which consists of two sets of wires crossing perpendicularly, as shown previously in Fig. 5.4a. In every cross-point (hereinafter also called as crossbar node) we assume having either a memristive device (a single memristor or a composite memristive structure), or a typical connection which can be adjusted as insulator (high resistance—open circuit) or as routing junction (very low resistance—short circuit). The types of memristive device that can be used as cross-point elements include a single memristor—being either forward (FPM) or reversely (RPM) polarized- and two memristors connected in anti-serial (ASM) or anti-parallel (APM) configuration.

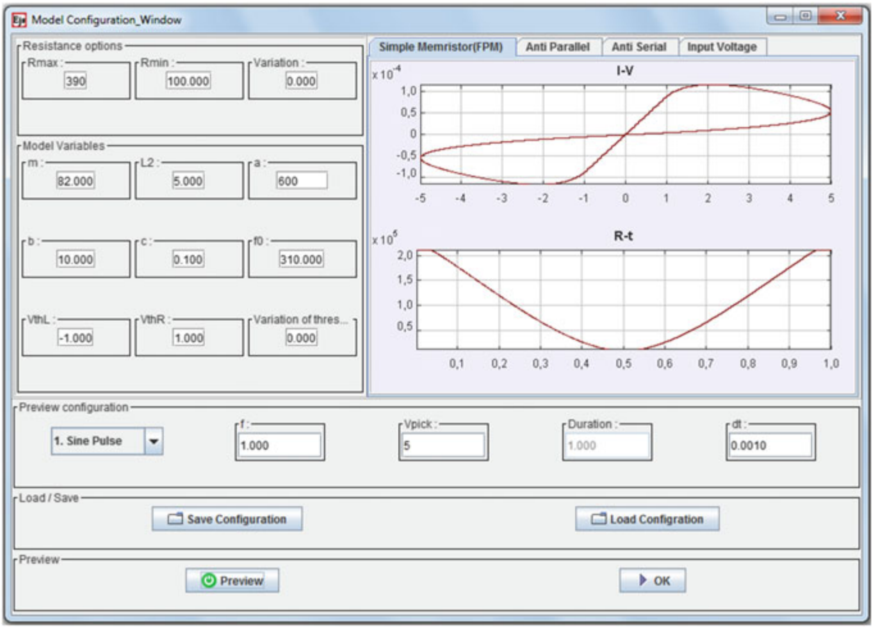
The main goals of developing XbarSim tool are listed below:

- Understand the fundamental dynamics of single memristors and composite memristive switches;
- Get familiar with the crossbar network circuit architecture;
- Explore the crossbar-based memristive circuit design space and find optimized solutions to comply with certain limitations (e.g. the leakage current through unselected cells which limits the crossbar memory array size, i.e. a critical parameter to design a memory module);
- Experiment with different bit-accessing schemes, calculate sneak current, and estimate long-term interference (state-drift), data-retention, etc., in memristive crossbar-based NVM;
- Find the optimal topology, design options, and conduct logic-in-memory computations.

Considering the fact that research on ReRAM NVM technologies is still in an early stage and there are only a limited number of NVM prototype chips available to experienced designers, we expect XbarSim to prove helpful in providing circuit performance and functionality estimations at an early design stage. Earlier versions of this tool have been used for the estimation of current-sneak paths and the evaluation of the mitigation strategies presented previously in this chapter. The complete version presented here is an engineering work and has been developed for users with no deep understanding of memristive dynamics. It comprises a graphical user interface (GUI) which facilitates the study of circuits with memristors and nonvolatile memory circuits. One of the most important design specifications concerns the portable and standardized method of representing information within the software so that all users can run their simulations easily even using input files created by third parties. Unlike other simulators [52], XbarSim is independent of the memristor device fabrication technology. The evolution of the resistance of all cross-point elements is visualized using graphical outputs showing the voltage-drop and the time-evolution of the resistance at each simulation step. The complete visualization of the current resistive state of every cross-point is difficult, since crossbar is a three-dimensional (3-D) topology. However, the novelty of this simulator is the effective visualization of the simulation result by producing 2-D graphs showing the most important aspects of the conducted circuit simulation. While SPICE-level models and circuit-level simulators for NVM energy and area estimations are already available [52, 58], an educational simulation tool similar to XbarSim is currently missing.

5.5.1 Details on the Simulated Circuit Topology

The memristive crossbar circuit simulator presented here is based on the device model of a threshold-type, voltage-controlled bipolar memristor presented in Chap. 2 [47, 48]. The user has the ability to separately interact and experiment with the device model through the model configuration graphical interface, shown in Fig. 5.16. The left part of the upper panel of this window enables: (i) the adjustment of the model fitting parameters according to the target experimental device that is being simulated, and (ii) the introduction of variation for specific device characteristics of cross-point cells (e.g. the boundary memristance values $\{R_{ON}, R_{OFF}\}$ or the switching thresholds $\{V_{RESET}, V_{SET}\}$). The user may then observe how each one of the supported memristive device combinations behaves based on the selected


Model Configuration_Window

Resistance options
Rmax : 390
Rmin : 100.000
Variation : 0.000

Model Variables
m : 82.000
L2 : 5.000
a : 600
b : 10.000
c : 0.100
f0 : 310.000
VthL : -1.000
VthR : 1.000
Variation of thres... : 0.000

Preview configuration
1. Sine Pulse
f : 1.000
Vpick : 5
Duration : 1.000
dt : 0.0010

Fig. 5.16 XbarSim: memristor device model configuration window

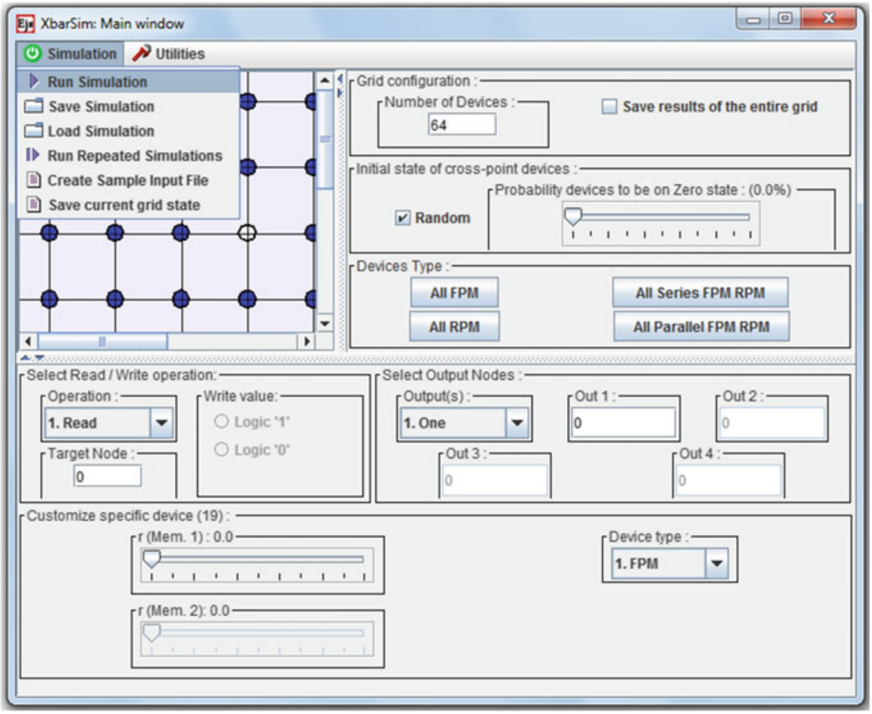
parameter value-set (the two memristors forming a particular ASM/APM cross-point configuration are always identical). The provided information includes the *I-V* and *R-t* graphs shown in the right upper part of the window for a variety of different applied pulsing options, inserted through the first row of the lower part of the interface (e.g. sinusoidal, saw-tooth waveform, or rectangular applied voltage). After the selection of the most suitable parameter value-set, the model configuration can be either saved for later use (likewise a previously saved model configuration can be loaded) or be applied to all the memristors of the array. This way the devices are all assumed identical, unless certain variation is introduced for particular model attributes.

XbarSim permits working with quadratic crossbar arrays (i.e. arrays with equal number of columns and rows) which consist of up to 4096 nodes (i.e. the dimensions can be at most 64 × 64). This limit for the total number of nodes serves to minimize memory requirements and accelerate the simulation through the GUI-based version of the tool. However, by replacing certain nodes of the quadratic array with insulating junctions, the user may indirectly define the desired dimensions of the target non-quadratic array; e.g. by defining a 32 × 32 array with all the nodes of the last 16 rows being insulated, then a 16 × 32 array is practically simulated.

The provided options for the initialization of the crossbar array are found in the right part of the upper panel of the main window, shown in Fig. 5.17, and include: (i) the total number of cross-points, (ii) the type of the cross-point devices, and (iii) the initial condition (resistive state) of the devices, with the latter being either common for the entire array or assigned randomly according to a given probability. Nevertheless, any cross-point can be also selected through the matrix-like 2-D representation of the crossbar and then be configured independently, according to the options given in the last row (at the bottom) of the user interface. For ASM/APM switches the two connected memristors can be separately adjusted.

During simulation, the available operations include: (i) reading from, and (ii) writing to (programming) the target cross-point devices. Such operations concern a single cross-point device per time. However, a series of access operations can be assigned to XbarSim to be executed sequentially through a simulation configuration input file. Such file includes the number and type of operations to be executed, as well as the array-initialization settings which are primarily defined inside the file. The array is initialized only once, so every subsequent access operation uses the output of the last simulation, hence making possible the study of the array performance in time.

As far as the available accessing schemes are concerned, XbarSim models two different options for the crossbar wires that are not immediately connected to the accessed cross-point where voltage (V_M) and ground (GND) are applied: (i) the floating array scheme, where all the wires are left floating; (ii) the $V_M/2$ scheme, where additional sources (hereinafter also called as “protecting sources”) of amplitude equal to $V_M/2$ are connected to all the wires. The latter scheme assures that the voltage drop on the rest of the cross-point devices will be at most equal to $V_M/2$,



Number of Devices : 64

☐ Save results of the entire grid

☑ Random

Probability devices to be on Zero state : (0.0%)

Operation : 1. Read

Target Node : 0

Write value: ○ Logic '1' ○ Logic '0'

Output(s) : 1. One

Out 1 : 0

Out 2 : 0

Out 3 : 0

Out 4 : 0

Customize specific device (19) :

r (Mem. 1) : 0.0

r (Mem. 2): 0.0

Device type : 1. FPM

Fig. 5.17 XbarSim: main window of the simulation tool

hence it addresses better the interference issue concerning the devices that are not being accessed during each operation. This is of course more important during programming operations because of the high voltages that are applied in order to change the state of particular cross-point elements. The currently used accessing scheme can be changed through the simulation options.

Moreover, XbarSim provides the opportunity to experiment with alternative crossbar patterns/topologies which combine memristive devices along with insulated nodes within the same array, where the latter are distributed either according to particular patterns or in a custom-wise manner (the available pattern options were previously presented in Sect. 5.4.5). Depending on the applied topology, a subset of the total array cross-points are set as high-resistance connections (insulators) in order to address the inherent current sneak-path problem of floating crossbar grids by significantly improving the measured read voltage margins. However, the provided option to selectively set any cross-point as typical resistor of constant value (adjustable through the utilities of the tool) permits: (i) the simulation of cross-point defects within the array (e.g. stack-at-ON/OFF cases, etc.), as well as (ii) the study of different circuit topologies which combine the variety of the provided memristive structures, mapped on the crossbar architecture.

5.5.2 GUI-Based Simulation Procedure

According to the used device model, memristors demonstrate threshold-type response; they comply with a set of adjustable voltage thresholds for the SET and RESET operations meaning that there is negligible change induced to their memristance if the magnitude of the applied voltage remains below them. During simulation, a certain voltage pulse of user-defined amplitude and duration is applied in order to write to (i.e. to program) a particular cross-point. In XbarSim logic values ‘0’ and ‘1’ correspond to R_{ON} and R_{OFF}, respectively, when typical memristive cross-points are concerned. However, depending on whether the cross-point cell is a FPM or a RPM, the polarity of the programming pulse should be different; e.g. writing ‘0’ to a FPM (RPM) requires a positive (negative) voltage. Therefore, depending on the device orientation and/or the data to be written, the correct polarity of the applied pulse is set automatically to avoid confusion.

Likewise, reading the state of a device is performed via a pull-up series resistor (R_{PU}) by applying a user-defined voltage pulse whose amplitude is normally set below the programming thresholds (here the applied voltage is by default positive). The R_{PU} value is also set by the user. In order to perform correct read operations, the optimal R_{PU} value is close to the less resistive state of the cross-point device type (i.e. $\mathrm{R}_{\mathrm{PU}} = R_{ON}$ for typical memristive cross-points, $\mathrm{R}_{\mathrm{PU}} = 2 \times \mathrm{R}_{\mathrm{ON}}$ for ASM, and $\mathrm{R}_{\mathrm{PU}} = \mathrm{R}_{\mathrm{ON}}/2$ for APM switches, respectively).

The time-evolution of the entire array during simulation can be optionally saved in a simulation log file, whereas information regarding up to four selected cross-points of interest can be demonstrated graphically through *R*-*t* and *V*-*t* graphs, as shown in Fig. 5.18. The above set of cross-points can be changed between consecutive simulations, whereas information from each subsequent access operation is appended (concatenated) to the existing output graphs unless the user resets the simulation output window. Furthermore, the composite resistive state of the array is schematically shown using a 2-D color-map representation to denote different resistance values of the memristive cross-points, as shown in Fig. 5.19. The correspondence of the equivalent node resistance to a particular color value is done in the following way: for every cross-point the assigned color reflects the ratio of the current node resistance value and the max resistance of this particular node. Min and max memristance values are common when all cross-point devices are identical. However, in case of having simultaneously both single memristors and composite memristive switches as cross-point devices in the array, the min and max memristances are device-dependent and may be different for different cross-points.

If the time-evolution of the entire array is not required, the current state of the array (i.e. the specific simulation time along with node-specific information such as device-type, resistance value, etc.) may be saved in a file. Similarly, an XbarSim simulation file, which contains the current values of the entire set of variables of the tool, can be created at any time; simulation may continue exactly from the particular stored point if a simulation file is loaded.

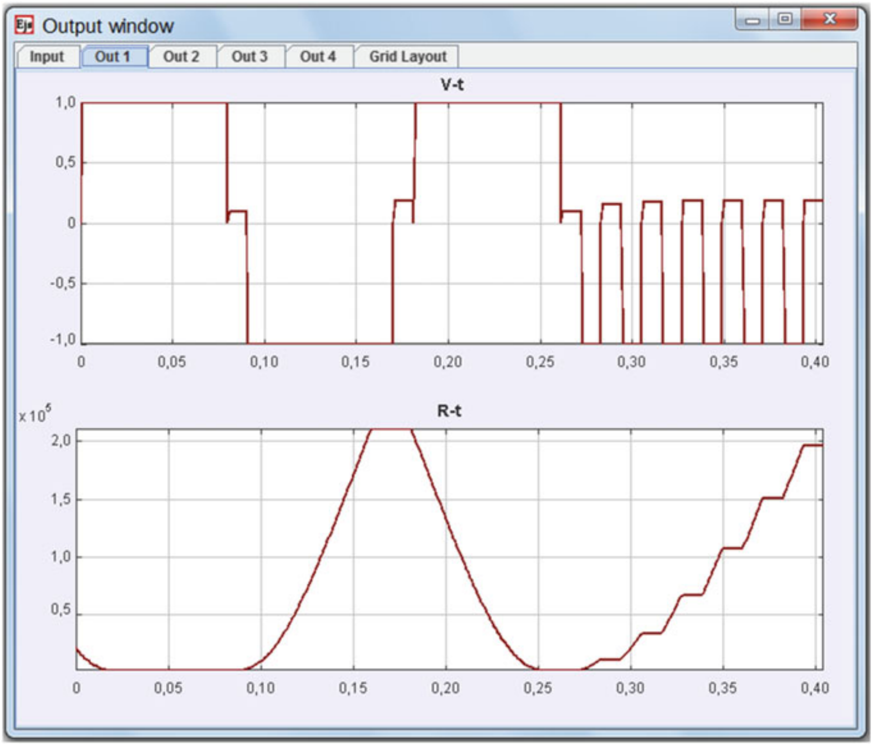


Fig. 5.18 XbarSim: *v-t* and *R-t* output graphs in output window of the simulation tool

5.5.3 Simulation Details—Crossbar Network Nodal Analysis

Generally, in each simulation step the tool takes into account the current resistive state of the array and resolves the resistive crossbar network in order to calculate the voltage drop on each memristive cross-point. Then the memristor model is employed to compute the memristance evolution by numerically solving all differential equations using a 4th order Runge-Kutta integration method as implemented in EJS [29].

Given the large size of the crossbar arrays supported by XbarSim, a systematic extension of the usual node-voltage analysis method was performed so as to convert all necessary equations to a more process-appropriate form using matrices. Matrices are used both to describe the topological properties of the resistive circuit as well as for the input/output variables of the equations. Analysis of the circuit is then performed by using linear algebra and graph theory. For a particular circuit we calculate the corresponding directed graph which demonstrates both the overall interconnection and the internal structure of the memristive network (i.e. type and

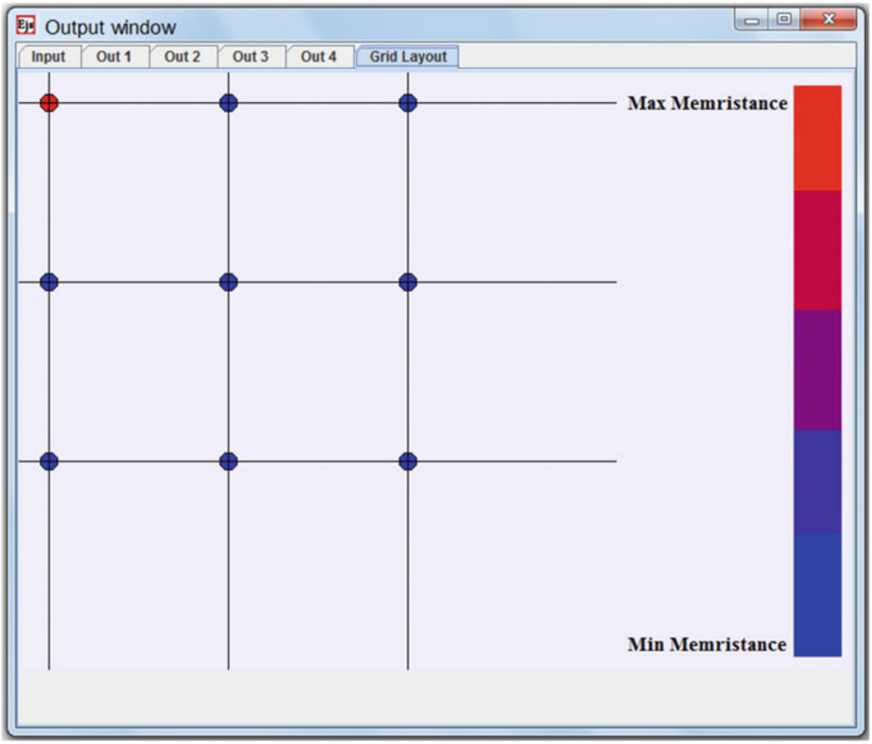


Fig. 5.19 XbarSim: 2-D colormap representation of resistive state in output window of the simulation tool

orientation of devices). More specifically, the entire resistive crossbar circuit is represented as a directed graph comprising a set of *n* nodes and *b* branches. Each branch *k* generally comprises a DC voltage source v_{sk} connected in series to a memristive element of composite memductance g_k, and a DC current source j_{sk} connected in parallel, as shown in Fig. 5.20a.

The matrices which are used in the node-voltage analysis are: (i) a $n \times b$ incidence matrix A (also called node-branch incidence matrix) whose rows and columns equal the number of nodes and branches of the graph, respectively, and whose values {0, 1, −1} denote the existence (non zero) and/or direction of the flowing current j_k which leaves from node *i* (1) and enters to node *j* (−1); (ii) a $b \times b$ memductance matrix G whose rows and columns equal the number of branches of the graph, and whose only non-zero elements (in the main diagonal) contain the memductance values of each branch; and (iii) the source vectors V_s and J_s which contain the values of the DC voltage and current sources of each branch, respectively.

In a circuit comprising *n* nodes, we assume the reference node to be node *n*, whereas the rest of the nodes are arbitrarily numbered as 1, 2, …, *n* − 1. Moreover,

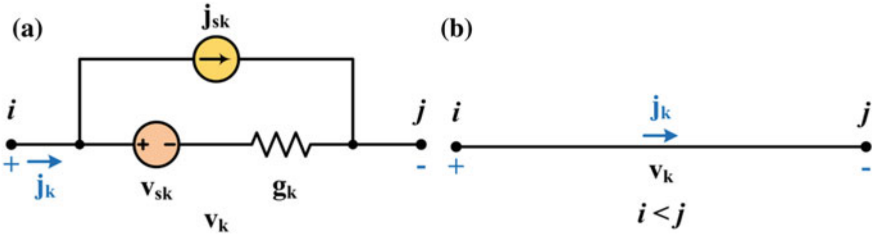


Fig. 5.20 XbarSim: **a** generic form of circuit branch *k*. **b** Conventional current flow in a circuit branch

regarding the generalized circuit branch of Fig. 5.20, we assume the number of node i to be smaller than j, thus the current j_k always flows from the lower-numbered towards the higher-numbered node; the DC sources have positive values if they are placed as shown in Fig. 5.20. Therefore, the basic equation describing the generalized branch k has the following form:

$$j_k = g_k v_k + j_{sk} - g_k v_{sk}. \tag{5.17}$$

If the above equation is applied to all the branches b using matrices, it will take the following form:

$$J = GV + J_S - GV_S \tag{5.18}$$

where vectors J, G and V contain the current, the memductance, and the voltage values of all the branches, respectively. In our case, since no voltage/current sources are necessary inside the crossbar network, the corresponding matrices are filled with zeros. However, additional circuit branches which correspond to the externally applied reading/writing voltage sources are included in the graph; a resistor $\mathrm{R_{PU}}$ is used instead of a memristive element in series to a reading voltage source, whereas no resistor is assumed for the programming or the protecting voltage sources ($r_k \approx 0$).

The application of Kirchhoff's current law to each node of the circuit (except the reference node) is resumed in the solution of the following system of equations:

$$A_S J = 0 \tag{5.19}$$

where $\mathrm{A_S}$ is a $(n - 1) \times b$ smaller version of the incidence matrix after removing the row which corresponds to the reference node where the GND is connected (this simplification imposes no loss of information since the row corresponding to the reference node can be easily derived from the rest of the values in each column of $\mathrm{A_S}$). After the calculation of the aforementioned matrices, the algorithm leading to the solution of the above system of equations is summarized below. XbarSim calculates the auxiliary matrices $\mathrm{I_s}$ and $\mathrm{Y_n}$ according to the following equations:

$$I_S = A_S G V_S - A_S J_S \tag{5.20}$$

$$Y_n = A_S G A_S^T \tag{5.21}$$

where the $\mathrm{A}_\mathrm{s}^\mathrm{T}$ matrix is the transpose A_s matrix. Then the node-voltages are calculated by solving the following system of equations:

$$Y_n E = I_S \tag{5.22}$$

where E is a vector containing the node-voltage variables. Once the node-voltage values are calculated, the voltage drop and the current flowing through any branch can be easily computed based on the following equation and Eq. 5.18, as shown below:

$$V = A_S^T E \tag{5.23}$$

Equation 5.23 simply correlates the voltage drop on the branches with the node-voltages of the circuit as: $\mathrm{v}_k = e_i - e_j$. Once the voltages on the circuit branches have been calculated, XbarSim employs the memristor model in order to compute the memristance evolution of each cross-point element.

All matrices involved in linear algebra computations within XbarSim use the Linear Algebra for Java (LA4J) open source library [59]. The computational complexity of the system of equations depends on *n* and *b*; hence the simulator suffers from significant slowdown as any of these variables increases. However, LA4 J enables computations using sparse matrices, so the computation complexity was somewhat improved using sparse matrix techniques since the majority of the matrices within XbarSim contain a large number of zeros. In the GUI-based version of XbarSim presented, the type of matrices used is user-defined and sparse matrices facilitate memory saving when large crossbar arrays are being simulated.

References

1. P. Pavan, R. Bez, P. Olivo, E. Zanoni, Flash memory cells-an overview. IEEE Proc. **85**(8), 1248–1271 (1997)
2. International Technology Roadmap for Semiconductors (ITRS). Available: http://www.itrs.net/. Accessed June 2014 (2013)
3. S. Mondal, J.-L. Her, F.-H. Chen, S.-J. Shih, T.-M. Pan, Improved resistance switching characteristics in Ti-Doped Yb_2O_3 for Resistive nonvolatile memory devices. IEEE Elec. Dev. Lett. **33**(7), 1069–1071 (2012)
4. J.R. Heath, P.J. Kuekes, G.S. Snider, R.S. Williams, A defect-tolerant computer architecture: opportunities for nanotechnology. Science **280**(5370), 1716–1721 (1998)
5. K.H. Kim, S. Gaba, D. Wheeler, J.M. Cruz-Albrecht, T. Hussain, N. Srinivasa, W. Lu, A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. Nano Lett. **12**(1), 389–395 (2012)

6. Y. Ho, G.M. Huang, P. Li, Dynamical properties and design analysis for nonvolatile memristor memories. IEEE Trans. Circuits Syst. I Reg. Papers **58**(4), 724–736 (2011)
7. C. Kügeler, M. Meier, R. Rosezin, S. Gilles, R. Waser, High density 3D memory architecture based on the resistive switching effect. Solid-State Electron. **53**(12), 1287–1292 (2009)
8. D.B. Strukov, R.S. Williams, Four-dimensional address topology for circuits with stacked multilayer crossbar arrays. Proc. Nat. Acad. Sci. **106**(48), 20155–20158 (2009)
9. H.Y. Chen, S.M. Yu, B. Gao, P. Huang, J.F. Kang and H.-S.P. Wong, HfO_x based vertical resistive random access memory for cost-effective 3d cross-point architecture without cell selector, in *IEEE Int. Electron Devices Meeting (IEDM)*, San Francisco, CA (2012)
10. E.K. Lai, H.T. Lue, Y.H. Hsiao, J.Y. Hsieh, C.P. Lu, S.Y. Wang, L.W. Yang, T.H. Yang, K.C. Chen, J. Gong, K.Y. Hsieh, R. Liu, C.Y. Lu, A multi-layer stackable thin-film transistor (TFT) NAND-type flash memory, in *Technical Digest International Electron Devices Meeting (IEDM)*, San Francisco, CA (2006)
11. W. Kim, S. Choi, J. Sung, T. Lee, C. Park, H. Ko, J. Jung, I. Yoo and Y. Park, Multi-layered vertical gate NAND flash overcoming stacking limit for terabit density storage, in *Digest of Technical Papers, Symposium on VLSI Technology*, Honolulu, HI (2009)
12. J. Liang, H.-S.P. Wong, Cross-point memory array without cell selectors—device characteristics and data storage pattern dependencies. IEEE Trans. Electron Dev. **57**(10), 2531–2538 (2010)
13. P.O. Vontobel, W. Robinett, P.J. Kuekes, D.R. Stewart, J. Straznicky, R.S. Williams, Writing to and reading from nano-scale crossbar memory based on memristors. Nanotechnology **20** (425204), 1–21 (2009)
14. S. Hamdioui, H. Aziza, G.C. Sirakoulis, Memristor based memories: technology, design and test, in *9th IEEE International Conference Design and Technology of Integrated System in Nanoscale Era (DTIS)*, Santorini island, Greece (2014)
15. M.A. Zidan, H.A.H. Fahmy, M.M. Hussain, K.N. Salama, Memristor-based memory: the sneak paths problem and solutions. Microelectron. J. **44**(2), 176–183 (2013)
16. S. Kannan, J. Rajendran, R. Karri and O. Sinanoglu, Sneak-path testing of memristor-based memories, in *26th International Conference on VLSI Design and 12th International Conference on Embedded Systems (VLSID 2013)*, Pune, India (2013)
17. S. Kim, H.Y. Jeong, S.K. Kim, S.Y. Choi, K.J. Lee, Flexible memristive memory array on plastic substrates. Nano Lett. **11**(12), 5438–5442 (2011)
18. H. Manem, J. Rajendran, G.S. Rose, Design considerations for multilevel CMOS/nano memristive memory. ACM J. Emerg. Technol. Comput. Syst. **8**(16), 1–22 (2012)
19. W. Fei, H. Yu, W. Zhang, K.S. Yeo, Design exploration of hybrid cmos and memristor circuit by new modified nodal analysis. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **20**(6), 1012–1025 (2012)
20. M. Qureshi, W. Yi, G. Medeiros-Ribeiro, R. Williams, AC sense technique for memristor crossbar. Electron. Lett. **48**(13), 757–758 (2012)
21. R. Rosezin, E. Linn, L. Nielen, C. Kügeler, R. Bruchhaus, R. Waser, Integrated complementary resistive switches for passive high-density nanocrossbar arrays. IEEE Elec. Dev. Lett. **32**(2), 191–193 (2011)
22. C. Jung, J. Choi, K. Min, Two-step write scheme for reducing sneak-path leakage in complementary memristor array. IEEE Trans. Nanotechnol. **11**(3), 611–618 (2012)
23. J.J. Yang, M.-X. Zhang, M.D. Pickett, F. Miao, J.P. Strachan, W.-D. Li, W. Yi, D.A. Ohlberg, B. Joon Choi, W. Wu, J.H. Nickel, G. Medeiros-Ribeiro, R.S. Williams, Engineering nonlinearity into memristors for passive crossbar applications. Appl. Phys. Lett. **100**(11), 113501 (2012)
24. H.-S.P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F.T. Chen, M.-J. Tsai, Metal-oxide RRAM. IEEE Proc. **100**(6), 1951–1970 (2012)
25. I. Vourkas, G.C. Sirakoulis, Nano-crossbar memories comprising parallel/serial complementary memristive switches. BioNanoScience **4**(2), 166–179 (2014)

26. I. Vourkas, D. Stathis, G.C. Sirakoulis, Improved read voltage margins with alternative topologies for memristor-based crossbar memories, in *21st IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Istanbul (2013)
27. I. Vourkas, D. Stathis, G.C. Sirakoulis, S. Hamdioui, Alternative architectures towards reliable memristive crossbar memories, *IEEE Trans. Very Large Scale Integr.* (VLSI) Syst. (2015) doi:10.1109/TVLSI.2015.2388587
28. I. Vourkas, D. Stathis and G.C. Sirakoulis, XbarSim: an educational simulation tool for memristive crossbar-based circuits, in *IEEE Int. Symp. Circuits Syst.* (*ISCAS*), Lisbon, Portugal (2015)
29. Easy Java Simulations (EJS), Available: http://fem.um.es/Ejs/. Accessed 2014
30. R. Waser, R. Dittman, G. Staikov, K. Szot, Redox-based resistive switching memories—nanoionic mechanisms, prospects, and challenges. Adv. Mat. **21**(25–26), 2632–2663 (2009)
31. A.C. Torrezan, J.P. Strachan, G. Medeiros-Ribeiro, R.S. Williams, Sub-nanosecond switching of a tantalum oxide memristor. Nanotechnology **22**(48), 485203 (2011)
32. J.P. Strachan, A.C. Torrezan, G. Medeiros-Ribeiro, R.S. Williams, Measuring the switching dynamics and energy efficiency of tantalum oxide memristors. Nanotechnology **22**(50), 505402 (2011)
33. Z. Wei et al., Highly reliable TaOx ReRAM and direct evidence of redox reaction mechanism, in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA (2008)
34. H.Y. Lee, P.S. Chen, T.Y. Wu, Y.S. Chen, C.C. Wang, P.J. Tzeng, C.H. Lin, F. Chen, C.H. Lien, M.J. Tsai, Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO_2 based RRAM, in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA (2008)
35. J.J. Yang, M.-X. Zhang, J.P. Strachan, F. Miao, M.D. Pickett, R.D. Kelley, G. Medeiros-Ribeiro, R.S. Williams, High switching endurance in TaOx memristive devices. Appl. Phys. Lett. **97**(23), 232102 (2010)
36. M.-J. Lee et al., A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta_2O_{5-x}/TaO_{2-x} bilayer structures. Nature Mater. **10**(8), 625–630 (2011)
37. Z. Zhiping, W. Yi, H.-S.P. Wong, Nanometer-Scale HfO_x RRAM. IEEE Electron Dev. Lett. **34**(8), 1005–1007 (2013)
38. Y.-B. Kim et al., Bi-layered RRAM with unlimited endurance and extremely uniform switching, in *Symposium on VLSI Technology (VLSIT)*, Honolulu, HI (2011)
39. L. Goux et al., Field-driven ultrafast sub-ns programming in $W/Al_2O_3Ti/CuTe$-based 1T1R CBRAM system, in *Symposium on VLSI Technology (VLSIT)*, Honolulu, HI (2012)
40. C. Cagli et al., Experimental and theoretical study of electrode effects in HfO_2 based RRAM, in *IEEE International Electron Devices Meeting (IEDM)*, Washington, DC (2011)
41. A. Sawa, T. Fujii, M. Kawasaki, Y. Tokura, Hysteretic current–voltage characteristics and resistance switching at a rectifying Ti/Pr0.7Ca0.3MnO3 interface. Appl. Phys. Lett. **85**(18), 4073–4075 (2004)
42. H. Choi et al., The effect of tunnel barrier at resistive switching device for low power memory applications, in *3rd IEEE Interanational Memory Workshop (IMW)*, Monterey, CA (2011)
43. C.J. Chevallier, H.S. Chang, S.F. Lim, S.R. Namala, M. Matsuoka, B.L. Bateman, D. Rinerson, A 0.13 μm 64 Mb multi-layered conductive metal-oxide memory, in *IEEE Interantional Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, San Francisco, CA (2010)
44. E. Linn, R. Rosezin, C. Kugeler, R. Waser, Complementary resistive switches for passive nanocrossbar memories. Nat. Mater. **9**(5), 403–406 (2010)
45. S. Tappertzhofen, E. Linn, L. Nielen, R. Rosezin, F. Lentz, R. Bruchhaus, I. Valov, U. Böttger, R. Waser, Capacity based nondestructive readout for complementary resistive switches. Nanotechnology **22**(39), 395203 (2011)
46. L. Ni, F. Demami, R. Rogel, A.C. Salaün, L. Pichon, Fabrication and electrical characterization of silicon nanowires based resistors. IOP Conf. Ser. Mater. Sci. Eng. **6**(1), 012013 (2009)

47. I. Vourkas, G.C. Sirakoulis, A novel design and modeling paradigm for memristor-based crossbar circuits. IEEE Trans. Nanotechnol. **11**(6), 1151–1159 (2012)
48. I. Vourkas, A. Batsos, G.C. Sirakoulis, SPICE modeling of nonlinear memristive behavior, Int. J. Circ. Theor. Appl. **43**(5), 553–565 (2015)
49. A. Flocke, T.G. Noll, Fundamental analysis of resistive nano-crossbars for the use in hybrid nano/CMOS-memory, in *33rd European Conference on Solid State Circuits (ESSCIRC)*, Munich, Germany (2007)
50. J. Mustafa, R. Waser, A novel reference scheme for reading passive resistive crossbar memories. IEEE Trans. Nanotechnol. **5**(6), 687–691 (2006)
51. S.H. Chen et al., A highly scalable 8-layer vertical gate 3D NAND with split-page bit line layout and efficient binary-sum MiLC (Minimal Incremental Layer Cost) staircase contacts, in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA (2012)
52. X. Dong, C. Xu, Y. Xie, N.P. Jouppi, NVSim: a circuit-level performance, energy, and area model for emerging nonvolatile memory. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **31**(7), 994–1007 (2012)
53. K. Walus, T.J. Dysart, G.A. Jullien, A.R. Budiman, QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. IEEE Trans. Nanotechnol. **3**(1), 26–31 (2004)
54. C. Wasshuber, H. Kosina, S. Selberherr, SIMON-a simulator for single-electron tunnel devices and circuits. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **16**(9), 937–944 (1997)
55. I.G. Karafyllidis, A simulator for single-electron devices and circuits based on simulated annealing. Superlattices Microstruct. **25**(4), 567–572 (1999)
56. I.G. Karafyllidis, Quantum computer simulator based on the circuit model of quantum computation. IEEE Trans. Circuits Syst. I Regul. Pap. **52**(8), 1590–1596 (2005)
57. G. Zardalidis, I.G. Karafyllidis, SECS: a new single-electron-circuit simulator. IEEE Trans. Circuits Syst. I Regul. Pap. **55**(9), 2774–2784 (2008)
58. H. Li, P. Huang, B. Gao, B. Chen, X. Liu, J. Kang, A SPICE model of resistive random access memory for large-scale memory array simulation. IEEE Elec. Dev. Let. **35**(2), 211–213 (2014)
59. Linear Algebra for JAVA (LA4J). Available: http://la4j.org. Accessed Aug 2014

Chapter 6
High-Radix Arithmetic-Logic Unit (ALU) Based on Memristors

6.1 Introduction

It is well-known that faster arithmetic operations could be achieved via high-radix numeric systems [1]. However, due to the absence of appropriate storage devices, such a practice was not given much attention so far because it would require memory capacity doubling in order to represent high-radix numbers in binary mode. Moreover, in view of the prospected end of the so called “Moore’s law”, many new approaches, which aim to extend the applicability of Complementary Metal-Oxide-Semiconductor (CMOS)-based circuits and systems, have been proposed by the semiconductor industry. In fact, Flash memory is approaching fundamental scaling limits due to several reliability issues. Therefore, currently there is a growing interest in new devices for information processing and memory, as well as new technologies and paradigms for system architecture [2–9]. An exciting approach would be the development of a competitive, reliable, multi-level storage cell technology, which will enable storing multiple bits of information in a single memory element [10, 11]. Most such approaches so far are based on transistors [2], but the recent discovery of the memristor has renewed the interest for this promising scientific area [12–14]. Owing to their analog nature, memristors have a remarkable ability to store multi-bit values in a single cell [15, 16]; a property which adds significantly to their potential use in future multi-level storage cell technologies and high-radix arithmetic units [17].

Nowadays, there is a notable variety of systems that exhibit memristive behavior [18]. As it was also mentioned in the previous chapters, memristors offer several advantages, such as nonvolatility, rapid switching, low power consumption, and compatibility with conventional CMOS technology. Additionally, they provide an unconventional computation framework, which combines information processing and storage in the memory cell itself; the major distinction from the present computing paradigms [19, 20]. Such favorable performance characteristics render them a candidate technology, able to bring the next technological revolution in electronics,

I. Vourkas and G.Ch. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Emergence, Complexity and Computation 19,
DOI 10.1007/978-3-319-22647-7_6

while serving as a bridge between CMOS and the realm of nano-electronics, beyond the end of CMOS dimensional and equivalent functional scaling.

Storing multiple levels of data in a single memory element would also be particularly useful for representing synaptic weights in artificial neural networks and in neuromorphic information processing systems [21–23]; each different memory level (i.e. resistance state) can represent a different strength between two neurons. Nevertheless, despite the aforementioned favorable features, memristor technology is still at an early stage. As a consequence, memristors in practice demonstrate some weaknesses; e.g., due to their nonlinear response, it is usually very difficult to determine the proper pulse-width and the amplitude which will adjust their resistance to a particular value. Moreover, it is expected that such nonlinearity varies spatially within the chip die, further complicating the creation of a common multi-level programming procedure.

Considering these problems, multi-level resistive random access memory (ReRAM) was discussed in [14] where the authors proposed the use of a feedback loop with a comparator and an array of reference resistors in order to achieve better resistance programming. In [24] several crossbar-based architectures, using diodes in series with memristors or CMOS multiplexers to mitigate the effect of current sneak paths, were discussed for variation-tolerant multi-level ReRAM. Also, [13] describes a multi-level ReRAM architecture where every crossbar column represents a single memory cell, being able to deliver higher or lower storage density depending on the needs of the application. Hybrid memristor-transistor ternary content addressable memories were also investigated in [25]. Nonetheless, such approaches eventually limit significantly the memory density and/or the overall performance with complex read/write algorithms.

In this chapter, we present a novel method for implementing crossbar-based multi-level memories, where each cross-point cell stores multiple bits. Furthermore, we propose a conceptual solution for novel CMOS-compatible, memristive, high-radix arithmetic logic units (ALUs) for future computing systems. More specifically, we describe a hybrid ALU circuit nano-architecture, where:

- CMOS peripheral circuits are used for binary arithmetic operations;
- A memristive reconfigurable crossbar-based memory block is used to:
  1. Allow parallel read/write of data;
  2. Facilitate the implementation of efficient arithmetic algorithms (e.g. fast partial product creation for multiplication);
  3. Store information in a compact, high-radix form.

Such a system combines fast CMOS circuitry with high-density resistive memory, where the compact high-radix storage of numbers takes place. Instead of using single memristors, the crossbar nodes here comprise a type of multi-state composite memristive switches [9, 26, 27], described in Chap. 3, which permit multi-bit storage in a more robust manner. Programming of the cross-point cells does not rely on reference resistors, but instead on the switching thresholds of memristors. Without reference resistors, the comparisons required to determine the exact state of the memory cells are significantly reduced. Of course, as it was highlighted throughout Chap. 5, read noise margin is a very important design parameter to consider in the development of memory architectures. In the case of multi-level memories, this parameter varies significantly with the number of memory states that can be stored in a cell. Here we particularly use radix-4 representation because: (i) it balances the offered advantages with the peripheral binary conversion circuitry overhead; and (ii) it provides a good density/reliability trade-off. The memory module of the system allows for parallel read/write operations and achieves inherently the parallel creation of partial products, to be used for faster multiplication. Such a high-radix memristive storage component could be an excellent device for storing register values nearby ALUs, where we assume data processing still to occur in conventional CMOS logic. Specifically, the binary inputs activate the corresponding multi-state programming signals, whereas the radix-4 stored information is converted to binary representation with the use of a network of comparators, before it is supplied to a computational layer of fast adders.

The fine operation and accuracy of the proposed system architecture is demonstrated through SPICE-level simulations, which are based on the threshold-type switching memristor model [28], presented in Chap. 2. The necessary driving circuitry is designed and explained in detail. All the operational phases are demonstrated in a step-by-step manner via comprehensive circuit schematics, whereas the main characteristics and the input pulsing requirements are also discussed. For a detailed presentation of circuit modeling and analysis of the resistive crossbar geometry, the reader is kindly requested to refer to Chap. 5.

6.2 Overall Layout of the Memristive Multi-level Memory System

A block-level representation of the proposed multi-level memory architecture is shown in Fig. 6.1. The main circuit component concerns a memristive crossbar-based memory module, properly interfaced in its periphery by programming circuitry from the top, and by reading circuitry from the right and bottom sides. The part at the bottom, which includes the sensing circuitry and the network of comparators, providing with the binary representation of the stored information in a target crossbar row (memory word), operates in an alternate manner with the programming inputs; it is activated exclusively during the reading mode, while the programming inputs are properly isolated, and vice versa.

In order to exploit the benefits of reliable, compact, multi-level storage, while maintaining relatively low the overall complexity of the peripheral CMOS driving and data-processing circuitry, in our implementation we use the radix-4 arithmetic system, thus storing two bits per cross-point element. The latter comprise multi-state memristive switches, which are composed of small networks of memristors.

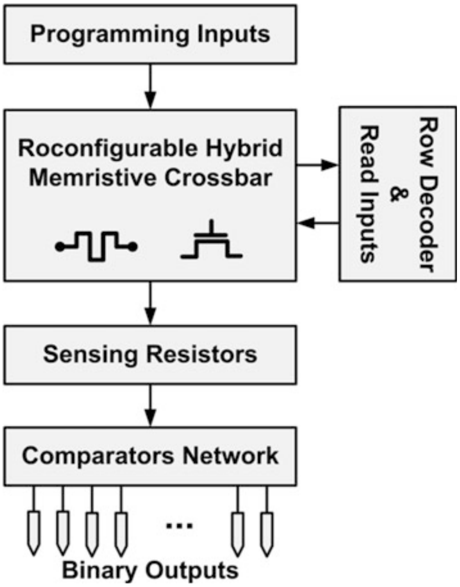


Fig. 6.1 Block diagram of the proposed circuit nano-architecture

6.2.1 Multi-level Storage Cell

In memristive memories, which are composed of memristive memory cells usually placed in crossbar geometry, the resistive state of the memristors indicates the data stored. For example, when storing one bit of information per cell, then the high resistive state of the device can represent one binary state, e.g. logic ‘0’, and its low resistive state can represent logic ‘1’. This particular convention was used throughout the memristive storage structures investigated in Chap. 5, and a characteristic schematic representation of the memristance distribution for storing one bit per cell, is shown in Fig. 6.2a. There are three memristance zones in total, defined by two resistive bounds which facilitate comparisons, namely the higher bound of logic ‘0’ [High(0)] and the lower bound of logic ‘1’ [Low(1)]. A typical reading method, as we will discuss later in more detail, includes comparing the resistivity of the memristive cell against a set of reference resistors, to determine the stored state. Two comparisons per stored state are required to verify the current resistance of the cell; one to determine if it is equal to or greater than High(0), and another to determine if it is equal to or less than Low(1). Any memristance value, in-between these boundaries, does not correspond to a binary value, so it is considered unspecified. Such resistive bounds need not be symmetric, as demonstrated here, but are rather selected after taking into consideration the device behavior and its variation characteristics, in order to achieve higher reliability.

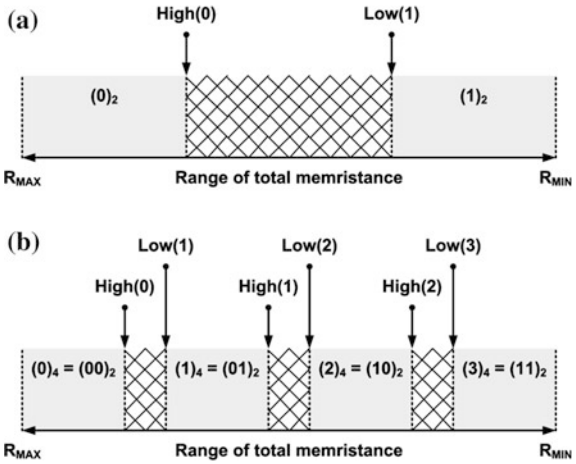


Fig. 6.2 Distribution of the total memristance range to represent **a** one-bit and **b** two-bit memory cells

However, a memristive memory cell can store multiple bits of data, as well. In general, as demonstrated in Chap. 2, the memristance is a function of the state variable of the device. Therefore, it can be divided into several resistance zones in order to represent multiple memory values. Throughout the rest of this chapter, a memory cell storing two bits will be used as a multi-bit memory cell. A schematic representation of the memristance distribution, properly divided to represent four different possible stored states, is shown in Fig. 6.2b.

In-between the six internal resistance boundaries, there are intermediate guard-bands, which facilitate the detection of unintentional changes in resistance levels, leading to an undetermined memory value. It is important to note that, regardless of the number of stored bits, without any guard-bands the memristance could accidentally switch levels due to noise or diffusion, resulting in an undetected error. Of course, it is not always possible and/or required to evenly divide the total memristance range into noise margins and guard-bands, as it is deliberately shown here. Instead, the most appropriate distribution depends on the desired number of memory levels, the interface circuitry, which translates the stored resistance to a corresponding voltage value, and the specific structure of the storage cell, which might include more than one memristor. Multi-level resistive memories exhibit a trade-off between memory density (the number of adjustable resistance levels) and reliability. With more available resistance levels per storage element, the noise margins are inevitably reduced, and the same happens with the reliability of the memory. Depending on the target application, higher memory density may be achieved at the cost of reliability reduction, or reliability may be more important than memory density.

In view of the aforementioned issues, concerning the appropriate distribution of the total memristance for multi-bit storage, the proposed multi-level crossbar-based memristive memory is based on memristive multi-state switches (MSS) composed of networks of memristors. Such switches were presented in Chap. 3 and below we provide with the fundamental properties on which their operation is based. In fact, the parallel connection of n identical memristors of the same polarity, results in an overall conductance range of $n \times [G_{ON}, G_{OFF}]$, where $[G_{ON}, G_{OFF}]$ corresponds to the range of a single memristor. Hence, increasing the number of parallel memristors increases the overall conductance. Connecting n such parallel groups (each having n memristors) in series, eventually increases the cumulative voltage threshold by n times, maintaining the total conductance range equal to $[G_{ON}, G_{OFF}]$.

Based on this property, Fig. 6.3 shows how memristive four-state MSS can be designed. For readability reasons, in Fig. 6.3a we demonstrate an equivalent symbol to represent a number of parallel memristors. This symbol is then used in Fig. 6.3b where another symbol is introduced to represent the four-state MSS and will be used in the rest of this chapter. Specifically, assuming three circuit branches in parallel, if all the memristors are in high memristance (R_{OFF}), the total conductance is very low and its value approximates $3 \times G_{OFF}$. With respect to the memristance distribution of Fig. 6.2b, this low conductance will represent state ‘0’. Moving to any of the higher conducting states is accomplished by applying a programming voltage which exceeds the aggregate threshold of any of the three successive circuit branches. If $\{V_{RESET}, V_{SET}\}$ are the threshold voltages of a single memristor, then a programming voltage of amplitude higher than V_{SET}, $2 \times V_{SET}$, or $3 \times V_{SET}$ will force the MSS to state ‘1’, ‘2’, or ‘3’, respectively. On the other hand, a high-enough negative voltage pulse (of amplitude higher than $3 \times |V_{RESET}|$) resets the MSS to state ‘0’ with all memristors set in R_{OFF}.

Therefore, the MSS of Fig. 6.3b exhibits four different conducting states and, depending on the application, it can be used to represent up to four possible binary states. The robust multi-bit programming capability of such MSS is exploited in the hybrid crossbar described afterwards. The radix-4 arithmetic system, on which the proposed architecture is based, consists of the following values: $[0, 1, 2, 3]_4 = [(00), (01), (10), (11)]_2$. Therefore, it is required that every MSS is able to be programmed into four different composite impedance states, which will correspond to the four possible values of the radix-4 representation. For further information about the MSS design methodology, the reader is requested to refer to Chap. 3.

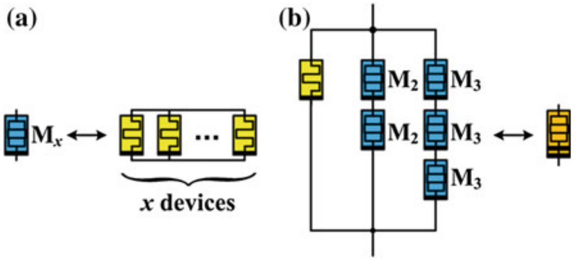


Fig. 6.3 Design methodology for memristive four-state switches based on networks of memristors

6.2.2 Analysis of the Circuit Topology

A more detailed circuit schematic of the proposed multi-level memory nano-architecture, is shown in Fig. 6.4. A typical crossbar network, as also described previously in Sects. 4.4.1 and 5.4.1, is composed of two sets of parallel nanowire-electrodes, crossing each other perpendicularly. In this chapter, a four-state MSS is located in every cross-point of the crossbar-based memory module.

Due to the radix-4 base, used here to represent the stored information, the required peripheral interfacing CMOS circuits are more complex than in the case of binary storage. According to Fig. 6.4, the binary inputs, applied on top of the circuit topology, activate the programming signals which change accordingly the conductance of every cross-point MSS of a target crossbar row (memory word). This

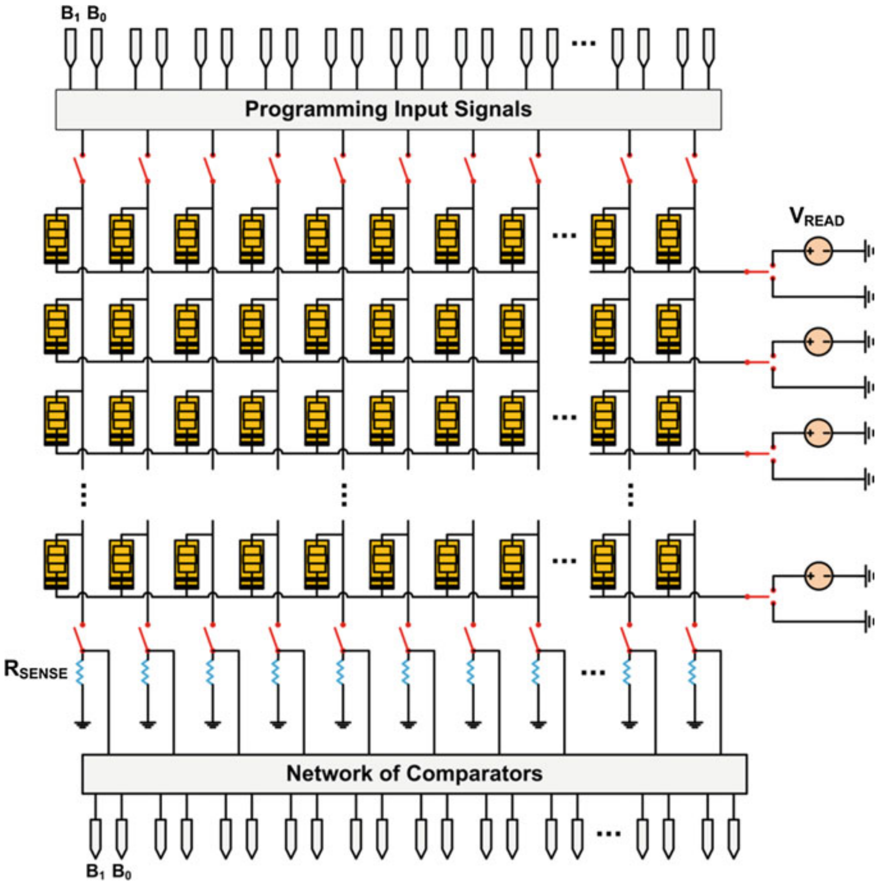


Fig. 6.4 General circuit schematic of multi-level crossbar-based MSS nano-architecture

way, we achieve parallel programming of one memory word per memory access. During the programming phase, the input column control switches are closed and only the control switch of the target row is connected to the ground, while the rest of the row control switches remain floating. Two input bits are encoded in the state of a single MSS. To this end, the state-programming signal activation is based on the summation of two DC voltages via a summing amplifier, like that of Fig. 3.16b, using two different weight coefficients which correspond to the significance of the input bits. For example, the input combination $(00)_2$ will give 0 V, i.e. will not permit the application of any programming signal, thus causing no effect to the MSS conductance. Similarly, the rest of the possible input pairs $\{(01)_2, (10)_2, (11)_2\}$ will result in a programming voltage of amplitude higher than $\{V_{\mathrm{SET}}, 2 \times V_{\mathrm{SET}}, 3 \times V_{\mathrm{SET}}\}$, capable of causing the desired change to the MSS state.

However, this relatively simple programming mechanism does not support bidirectional state changes; while going upwards from state $(0)_4$ to any other radix-4 state is possible, the same does not apply when we want to move backwards from the max state $(3)_4$ to states $(2)_4$ or $(1)_4$. This is because negative programming voltages are needed for this. Consequently, to avoid further state-comparisons, the prior knowledge of the currently stored state, and any additional programming voltages, which will further complicate the high-radix state-programming, instead we include a default RESET step prior to any programming step. This means that a RESET negative pulse first sets all the MSS elements of a target word to state $(0)_4$.

Regarding the row-decoding circuitry on the right side of the crossbar, the row control switches can (i) either leave the row wire floating, or connect a particular crossbar row directly to: (ii) a read DC voltage source, or (iii) the ground. Of course, depending on the selected read-out access scheme, additional options could be included; e.g. for the $V_{\mathrm{READ}}/2$ scheme, additional “protecting” sources of amplitude equal to $V_{\mathrm{READ}}/2$ could be connected to the rest of the row wires when a particular row is being read. Nonetheless, here we follow the simplest circuit approach for reading information from a memristor-based crossbar, which is by applying a certain read voltage across a junction and transforming the current flow into a corresponding voltage in the output of a voltage divider.

6.2.2.1 Reading/Writing Multi-level Data-from/to the Crossbar

The circuit snapshot of Fig. 6.5 shows a circuit configuration example for storing a 10-digit radix-4 number in a particular row of a 3 × 10 multi-level crossbar-based memory. After the activation of the programming signals, which correspond to the binary inputs (not shown in the figure), the generated programming voltages are applied to the crossbar columns by closing all the input column control switches. The selection of the target row is done by connecting the specific row wire to the ground, while leaving the rest of the row wires floating. We assume that the entire crossbar has first been RESET, so all the memory words have only state $(0)_4$ stored in all the cross-points. The RESET procedure can be accomplished in parallel via a negative programming voltage and by grounding all the crossbar rows

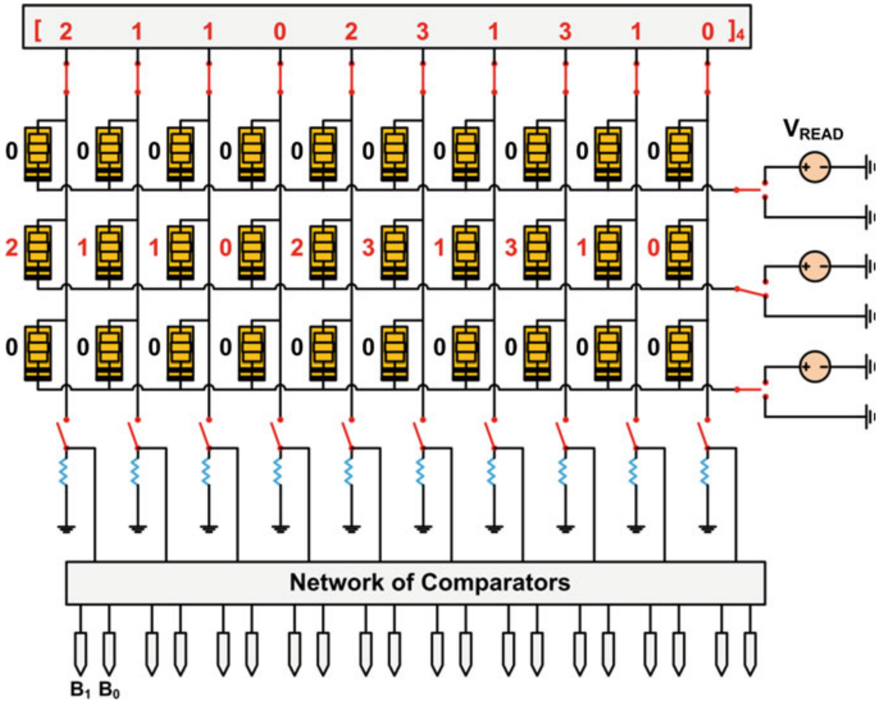


Fig. 6.5 Circuit snapshot corresponding to the programming phase of a 10-digit radix-4 number to the second row of a 3 × 10 crossbar array example. Numbers 0–3 next to every cross-point cell denote the stored resistive state

simultaneously. The sensing resistor array at the bottom should be cut-off during programming, so the output column control switches are all open.

Similarly, in Fig. 6.6 we demonstrate a typical circuit configuration for reading the 10-digit radix-4 number which was previously stored in the second row of the crossbar. The complementary operation between the input and output column control switches is obvious. A row control switch connects the target row wire to a read DC voltage, while the rest of the row wires are left floating. Via the output column control switches we create a voltage divider between every cross-point MSS and a series sense resistor. The voltage at every intermediate node (also called output node) is driven to proper circuitry where the radix-4 to binary conversion takes place. It is important to note that, owing to the threshold-type switching of memristors, the used reading method is nondestructive, meaning that the information stored in the memory word does not change when the read voltage is applied. Therefore, there is no need for reprogramming the recently read data. However, as mentioned above, resetting the contents of a particular word is required before storing different data in it.

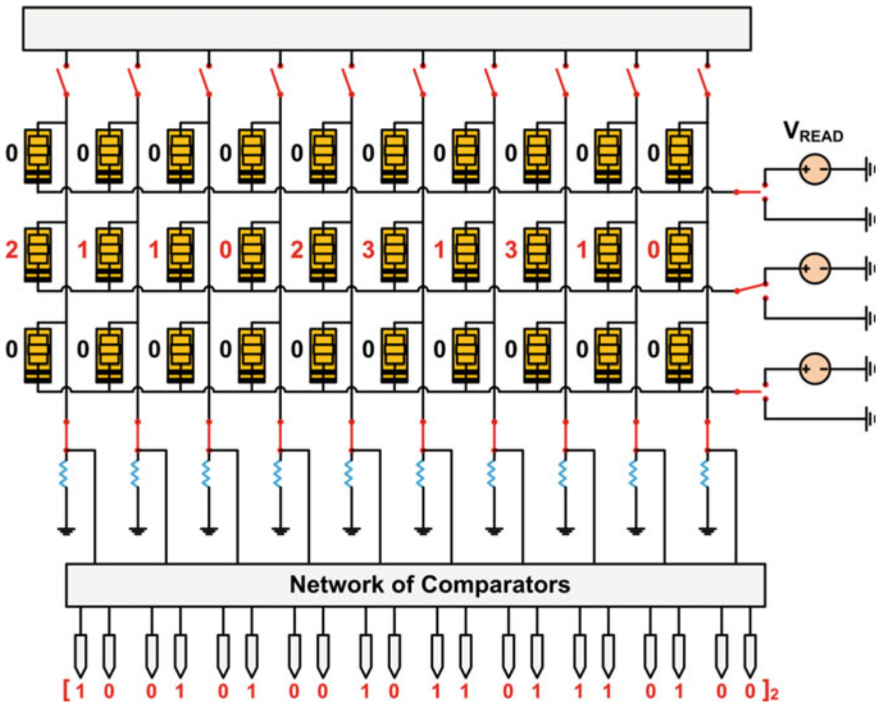


Fig. 6.6 Circuit snapshot corresponding to the reading phase of a 10-digit radix-4 number stored in the second row of a 3 × 10 crossbar array example. Numbers 0–3 next to every cross-point cell denote the stored resistive state

6.2.2.2 Encoding and Decoding of Multi-bit Memristive States

Binary to radix-4 state encoding is achieved via a relatively simple mechanism. Two input bits are encoded in the state of a single four-state MSS. Therefore, the correct state-programming signal is activated according to the significance of the input bits. A two-input summing amplifier with different input-weight coefficients can be used for the summation of two DC voltages. Unlike in the example implementation of Fig. 3.16b, here the two input bits will control two additional series switches, to either allow or cut-off the connection between the DC sources and the input node of the summing amplifier.

Similarly, at the output of the multi-level memory topology we need appropriate state-decoding circuits for the opposite conversion, so as to meet the compatibility requirements between the proposed high-radix memory block and the CMOS data-processing circuits. For the transformation of one of the four composite resistive states of every MSS into two binary signals, we use the three-level circuit implementation shown in Fig. 6.7, which consists of: (i) a voltage divider, (ii) a network of comparators, and (iii) additional digital logic components.

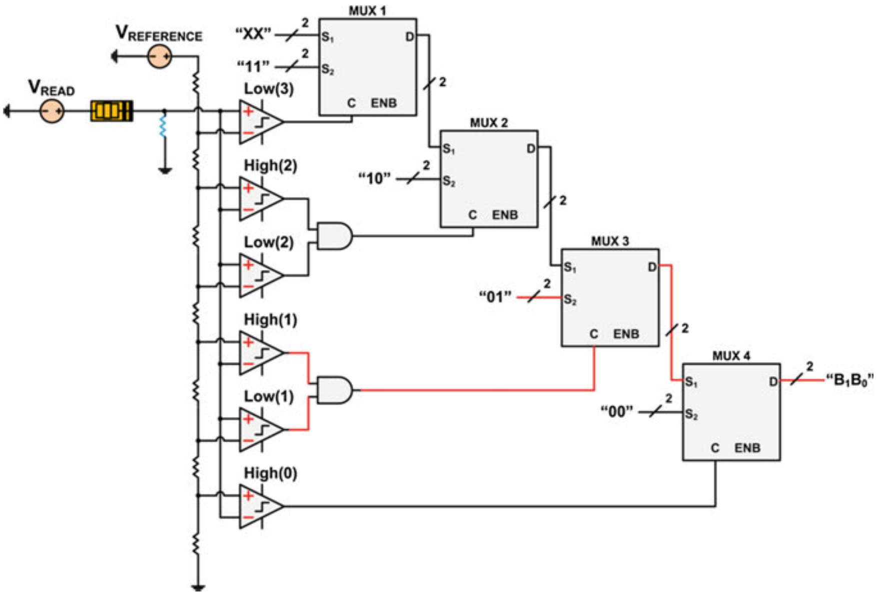


Fig. 6.7 Radix-4 to binary conversion circuit. The comparators are named in accordance with the internal resistive boundaries of the MSS. The *red lines* denote the generated signals which activate the $(1)_4$ to $(01)_2$ conversion

In the first level, a series sense resistor is connected to every selected memory cell through the output column control switches, forming a voltage divider circuit. In such a circuit configuration, the common node between the memory cell and the sense resistor is the intermediate node. The intermediate node will have a unique voltage value depending on the resistive state of the cross-point MSS and the resistance of the sense resistor. This voltage will change uniquely according to the stored composite resistance in the MSS, allowing the use of additional interpretation circuitry which will identify the exact stored state.

In [29] the authors proposed a solution for decoding multi-bit memristive states, in which the intermediate node is connected to a series of diodes. Each diode in this chain contributes a certain bias level. The resulting serially connected bias levels are exploited to detect the different voltage levels, which denote the different memristive states. The bias level in every diode has to be controlled in such a way that one voltage level corresponds exactly to the difference between two neighboring memristive states. The reason for using active elements in this decoding scheme is to avoid the inclusion of several different voltage sources, needed by the comparators, since this is not preferable to realize in Very Large Scale Integration (VLSI) systems. However, in the second level of the proposed decoding circuit, we rather include a series resistor network which is connected in parallel to the intermediate node. This way, by means of a single reference voltage source, we create all the necessary reference voltage levels which correspond to the memristance boundaries shown in

Fig. 6.2b, while avoiding additional voltage sources. The created reference voltage levels are driven to the network of comparators which produce digital CMOS-level signals. As discussed before, two comparisons per possible state are needed to verify the stored state. Generally, for an n-state MSS, $2n$ comparisons are required between the stored state and the upper/lower bounds of all possible states. This number can be reduced to $2n - 2$, by omitting comparisons with the lowermost and uppermost bounds, which constitute the memristance limits of the storage cell. Consequently, $2n - 2$ comparators and a minimum of $2n - 2$ reference resistors are necessary.

The digital CMOS-level signals, generated by the comparators, are fed to additional digital logic circuits, which finally produce the corresponding binary word in the last part of the decoding circuit. More specifically, the outputs of a subset of comparators and two AND logic gates, are driven to the control inputs of four cascaded 2 × 1 multiplexers, of which the last one gives the final output. In the circuit snapshot of Fig. 6.7, the red lines denote the activated control signals and the corresponding data flow during the $(1)_4$ to $(01)_2$ conversion. The total number of decoding circuits needed equals the number of storage cells that are read in parallel (the memory word-size). Therefore, the possible area overhead in the periphery is perhaps something to consider seriously in large multi-level crossbar arrays. On the other hand, the conversion circuit which uses active elements, proposed in [29], reduces the number of necessary comparators by half; hence it could be instead a more attractive design option.

6.3 Enhanced Crossbar for Memristive ALU with Built-in Memory

Using memristor-based (multi-level) memory blocks along with CMOS data-processing circuits, enables the creation of memristive arithmetic and logic units (ALU) with embedded memory, which facilitates the implementation of efficient arithmetic algorithms and the storage of information in the same computing unit. In this section we present an early conceptual approach of a memristive ALU, which is based on reconfigurable crossbar-based multi-level memory. Unlike typical crossbars, the one proposed here includes vertical and horizontal switches among the cross-point elements to determine the data flow during access operations.

A corresponding circuit schematic of the proposed reconfigurable crossbar nano-architecture, is shown in Fig. 6.8. As mentioned before, all the MSS in a crossbar row (memory word) are accessed and programmed in parallel. The row decoder circuitry connects the target row wire to the ground, while the rest of the row wires are left floating. Also, programming multiple words is possible by connecting the corresponding row wires simultaneously to the ground. As a result, the same input data are copied to all the words with grounded row wires. Therefore, our intention is to exploit the parallel programming capability of the crossbar for the implementation of faster multiplier circuit implementations.

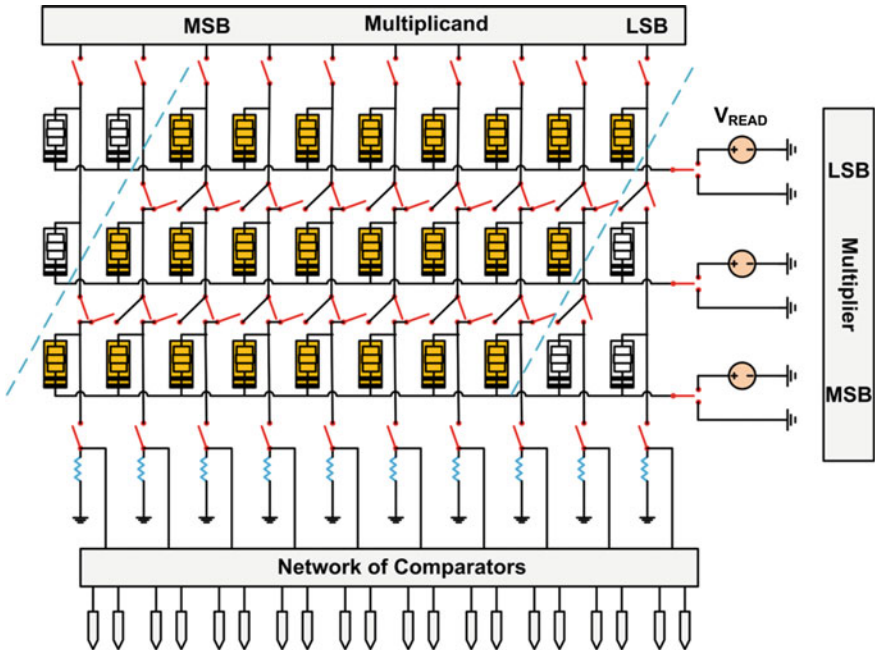


Fig. 6.8 Example of reconfigurable hybrid 3 × 10 multi-level crossbar memory architecture for memristive ALUs. The *diagonal dashed lines* designate the part of the array which is used during multiplication

A variety of computer arithmetic techniques can be used to implement a digital multiplier. Most techniques involve computing first a set of partial products, shifting them to the left, and then summing them together. Therefore, in digital electronics most binary multipliers are built using binary adders. The most difficult part of the process is to obtain the partial products, as that involves multiplying a long number by one digit. In binary encoding, though, each long number (the multiplicand) is multiplied either by ‘0’ or ‘1’ digits of the multiplier, so the partial product by ‘0’ or ‘1’ is just ‘0’ or the multiplicand itself, respectively. Computer CPUs usually perform the multiplication by using the shift and add features of their ALU.

As demonstrated in Fig. 6.8, in the proposed reconfigurable hybrid crossbar the bits of the binary multiplier are assigned to the crossbar rows, with the more significant bits being arranged downwards in the crossbar rows that are in lower position in the array. Specifically, in this circuit snapshot which involves a 3 × 10 crossbar, we assume that the first row is correlated with the least significant bit (LSB), and the last row with the most significant bit (MSB) of the multiplier.

On the other hand, the multiplicand is used to generate the appropriate programming signals on top of the topology, which will finally copy this number to all the memory words whose row wires are grounded. However, a typical crossbar would simply store the multiplicand several times without introducing any left-shift operations, which are necessary for the preparation of the partial products. In view of this problem, this hybrid crossbar topology includes vertical and horizontal switches inside the crossbar which: (i) divide the crossbar columns in smaller segments; and (ii) connect adjacent crossbar columns in a zig-zag form to permit left-shifting of the programming signals. To this end, the internal horizontal (vertical) switches are selectively closed (open) during programming, whereas the opposite happens in reading mode when the topology is converted to a typical crossbar. This way, the nth created partial product incorporates correctly $(n - 1)$ left shifts, owing to the inherent shifting capability. The memory words containing the partial products are finally read sequentially and the data are fed to a computational layer of fast binary adders (not shown in Fig. 6.8).

Moreover, we note here that the additional switches do not need to be introduced uniformly in the entire crossbar, but are rather placed in such a way that facilitates propagation of the programming signals, while taking into consideration the required shift operations for every partial product. Consequently, the cross-points located in the top left and bottom right parts of the crossbar, cannot be used in multiplication operations. Generally, within a $m \times n$ crossbar, when multiplying a k-digit multiplicand by a l-digit multiplier, we reserve a total area of $l \times (k + l - 1)$ cells, of which $l \times k$ cells are used in multiplication and $l \times (l - 1)$ are not included in the operation, but can be normally used as typical multi-level storage cells.

6.3.1 Parallel Creation of Partial Products for Fast Multiplication

Next we provide with two example circuit configurations regarding the generation of partial products, via the proposed reconfigurable hybrid multi-level crossbar memory. Figure 6.9 shows a circuit snapshot with the necessary configuration for the programming procedure. The 8-digit radix-4 multiplicand $(10100110)_4$ is assigned to the eight rightmost columns of the crossbar, whereas the 3-digit multiplier $(101)_2$ extends to all the three rows of this 3×10 crossbar example array. In this stage, the input column control switches on top of the topology, are all closed to allow the application of the generated programming signals; no input signals are applied to the two leftmost columns of the array. A crossbar row is connected to the ground when the corresponding input bit of the multiplier is logic ‘1’, whereas the row control switch leaves the row wire floating otherwise. From the additional internal switches, the horizontal switches are closed (designated with red color) to facilitate the correct left-shifted propagation of the programming signals, whereas the vertical switches (designated with blue color) are all open. As a consequence of this configuration, the multiplicand is appropriately copied to the first and the last row of this example topology. Moreover, a zero partial product is stored in the intermediate row. In fact, as mentioned before, a prior RESET step sets all the

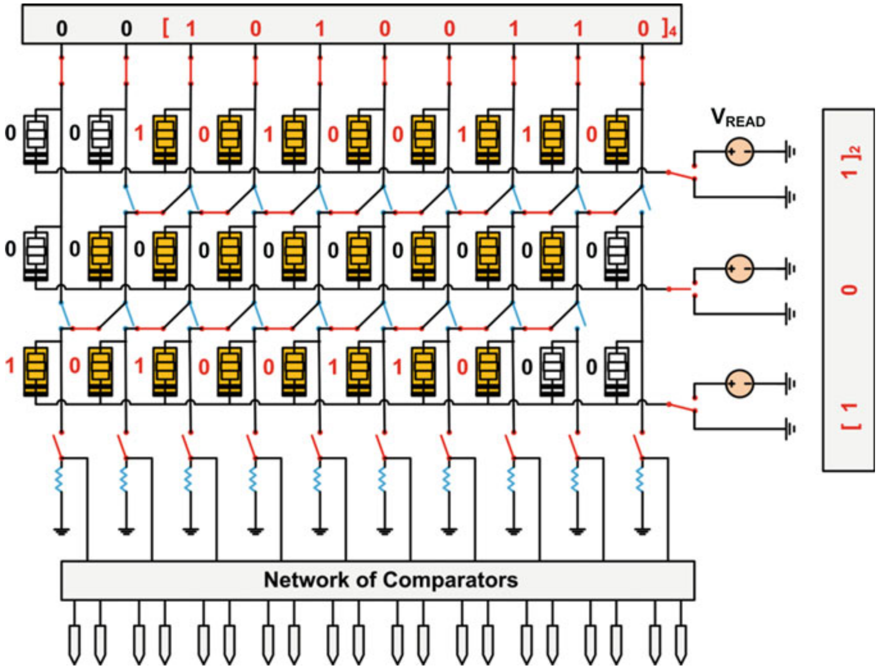


Fig. 6.9 Circuit snapshot corresponding to the programming phase during the multiplication of an 8-digit multiplicand by a 3-digit multiplier, taking place within a 3 × 10 crossbar array example. Numbers 0–3 next to every cross-point cell denote the stored resistive state

involved MSS to the less conducting state, equivalent to $(0)_4$. So, a zero partial product means that the corresponding MSS cross-points simply hold their initial value and are not affected during programming.

During the reading phase, the memory words, which contain the computed partial products, are downloaded from the crossbar in the same way as it was described in the previous section. A characteristic circuit snapshot is given in Fig. 6.10, where the currently stored information concerns the previously presented programming procedure. From the additional internal switches, the horizontal switches are now open, whereas the vertical switches are closed and the topology is converted to a typical crossbar. The target memory words are read sequentially. The output column control switches are closed to connect the multi-level crossbar with the sensing circuitry. When reading the first memory word, which is the case corresponding to the circuit configuration shown in Fig. 6.10, the binary outputs represent the following word $(0010100110)_4$. Similarly, the second word will return only logic ‘0’ values, whereas reading the last word will give the binary conversion of $(1010011000)_4$.

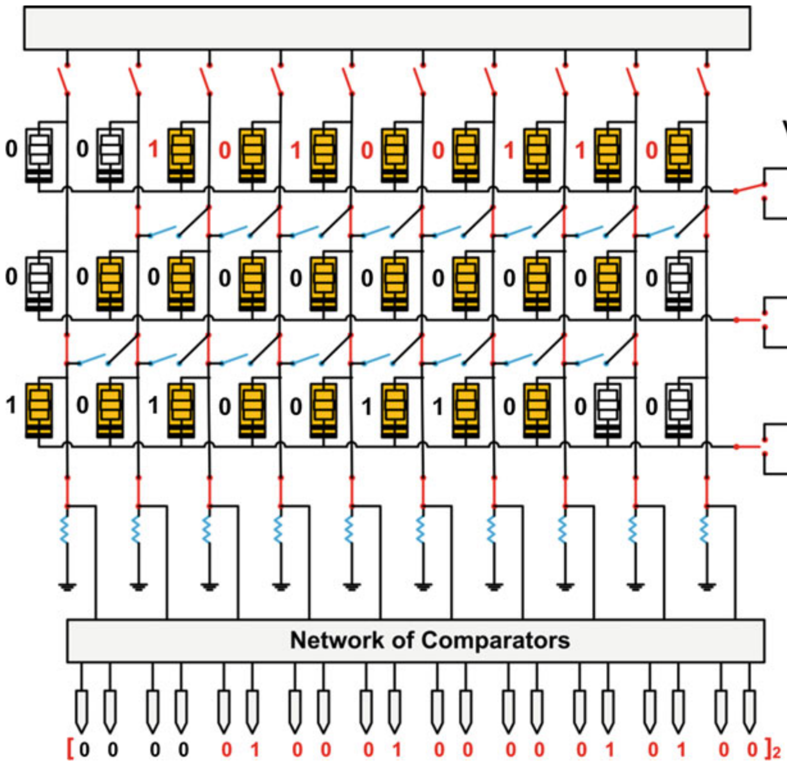


Fig. 6.10 Circuit snapshot corresponding to the reading phase of the partial products created by the multiplication of an 8-digit multiplicand by a 3-digit multiplier, within a 3 × 10 crossbar array example. Numbers 0–3 next to every cross-point cell denote the stored resistive state

6.4 Simulation Results

As a proof of concept, in this section we provide with the SPICE-based simulation results of multi-level memristive crossbar memory blocks during programming and reading of radix-4 data. Values of the parameters of the used memristor model are common for all the devices and are set as $\{a_x, b, c, m, f_0, L_0, V_{SET}, V_{RESET}\} = \{2 \times 10^3, 0, 0.1, 82, 310, 5, 2\text{ V}, -2\text{ V}\}$. The memristance ratio is $R_{OFF}/R_{ON} \approx 10^2$ with $R_{OFF} \approx 200\text{ k}\Omega$ and $R_{ON} \approx 2\text{ k}\Omega$, Moreover, we apply a read DC voltage of 1.2 V along with protecting DC voltage of 0.8 V. All assumptions regarding the switching thresholds and the programming voltages have been made only in the context of our simulations. The circuit schematic of a 4-state cross-point device is shown in Fig. 6.11a and, for readability reasons, a more compact schematic of the same MSS is given in Fig. 6.11b, in which the two rightmost parallel branches are represented with different equivalent circuit models. After having proved the operation of the original MSS in SPICE, with this simplification we

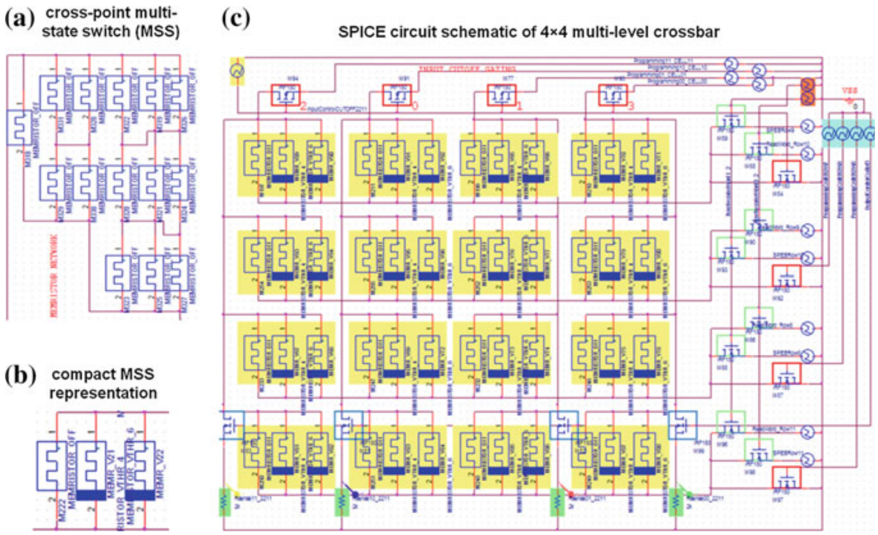

(a) cross-point multi-state switch (MSS)
(b) compact MSS representation
(c) SPICE circuit schematic of 4×4 multi-level crossbar
MEMRISTOR_OFF
INPUT CUTOFF GATING
VSS

Fig. 6.11 SPICE circuit schematics of **a** a 4-state memristive MSS, **b** a more compact representation of a 4-state memristive MSS, and **c** a 4 × 4 multi-level crossbar with programming and reading peripheral circuits

achieve to speed up simulation time because, instead of multiple interconnected memristors, we end up with only three parallel memristors whose only difference lies in their switching thresholds.

In Fig. 6.11c we demonstrate the SPICE-level circuit schematic of a 4 × 4 multi-state memristive crossbar, including the programming and the reading circuitry. The cross-point devices, highlighted in yellow color, comprise 4-state MSS. Several DC voltage sources in the periphery of the crossbar represent either programming input, read, or control signals. Such signals, although we assume to be generated by external circuitry, as it was explained in the previous sections, here they are produced directly via specific voltage sources for simplicity. Moreover, the resistors, which are highlighted in green color at the bottom of the topology, constitute the sensing elements used to create a voltage divider with the accessed MSS cell (the state-decoding circuit is not shown in the circuit schematic). All the control switches in SPICE are modeled using n-type MOSFETs, which are selected to demonstrate much higher channel resistance than memristors or composite MSS when in the less conductive state. Specifically, the n-MOS devices in red rectangles, found at the top of the topology, constitute the input column control switches; those in blue rectangles at the bottom are the output column control switches. Similarly, there are three n-MOS devices per crossbar row in the right side of the crossbar, which either: (i) determine the crossbar row wire which will be grounded, so that the corresponding memory word is programmed, (ii) connect the row wire to the read voltage source, (iii) connect a row wire to a protecting voltage source, or (iv) leave it floating.

Figure 6.12 demonstrates the equivalent SPICE circuit schematic of a 2 × 8 hybrid reconfigurable memristive crossbar, along with programming and reading peripheral circuits. Even though simple memristive cross-points are used here for readability reasons, in fact the major difference from the previous crossbar schematic consists in the internal CMOS reconfiguration switches, also modeled using n-type MOSFETs, which modify the topology during the parallel programming of the partial products. In this circuit snapshot, the transistors which are inside red rectangles are conductive only during the application of the programming signals, whereas those inside blue rectangles are conductive only during the reading phase. The rest of the circuit components, which appear in this figure, function similarly to those in Fig. 6.11c.

As it was also mentioned in Sect. 4.4.2.1, it is important to note that since both positive and negative input voltages are required to SET and RESET the memristors, the control switches should be able to control the power flow in both directions. However, although FETs will conduct equally in both directions when they are turned "on," when they are turned "off" they will still conduct in the reverse direction, as a consequence of the body-source connection which is typically attributed to the pn junction formed between the body (p) and the drain (n) (for n-channel). Therefore, by using two FETs with their source nodes connected

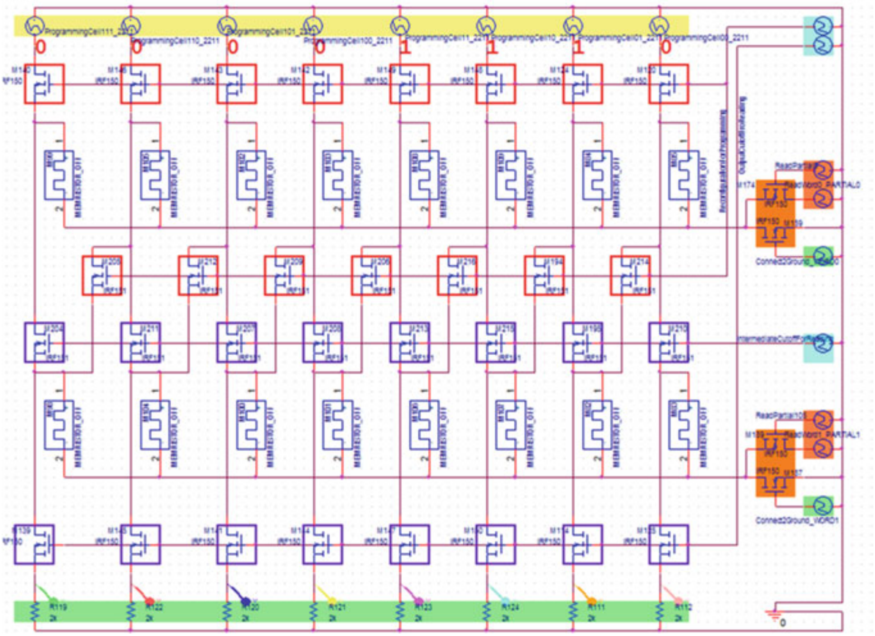


Fig. 6.12 SPICE circuit schematic of a 2 × 8 hybrid reconfigurable memristive crossbar with programming and reading peripheral circuits

back-to-back, instead of only one, in our simulations we achieved the necessary power flow control.

In Fig. 6.13 we demonstrate the simulation results regarding programming and reading of a 3-digit radix-4 number to/from a 2 × 4 multi-level hybrid reconfigurable crossbar memory, when adjusted to operate as a typical crossbar. In the circuit

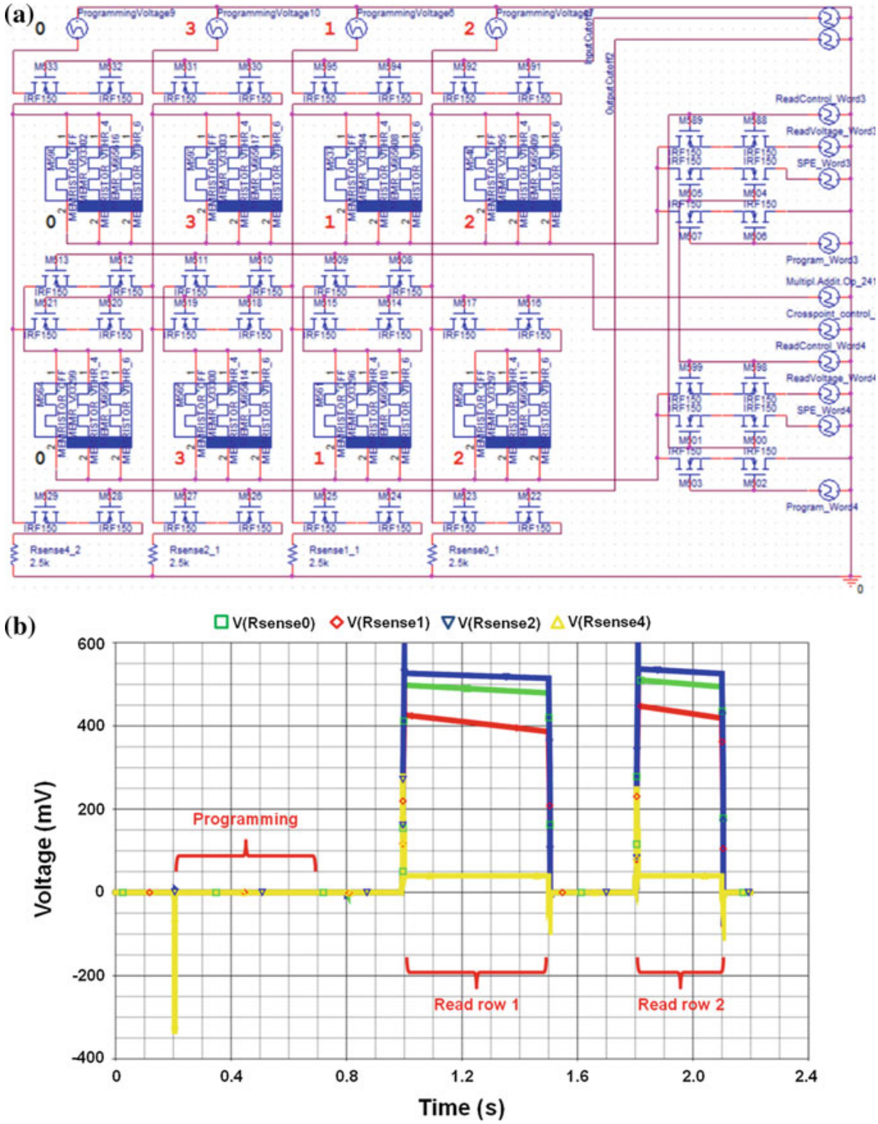


Fig. 6.13 SPICE simulation results for a 2 × 4 multi-level reconfigurable crossbar memory when the same radix-4 input is stored in both memory words. **a** shows the circuit schematic and **b** the measured voltage on the sense resistors

schematic of Fig. 6.13a, the numbers next to the programming input voltage sources and the MSS cross-points denote the corresponding radix-4 digit to be stored. Also, there are four 2.5 kΩ sense resistors and the voltage drop on them during simulation is illustrated in Fig. 6.13b.

Both row wires are grounded and the state-programming phase, concerning the number $(312)_4$, lasts 0.5 s. After a 0.3 s interval, we apply twice the reading signals to first read the top (1.0–1.5 s) and then the bottom (1.8–2.1 s) word of the crossbar. The measured output voltage levels in both memory words are indicative of the same multi-level stored information. Consequently, both words are correctly programmed to hold the multi-level input values. It is notable, though, that the voltage margins concerning the higher among the different stored states, seem to degrade. Nevertheless, sense resistors of optimal value can improve the read-out results.

Next we provide the simulation results of the same reconfigurable hybrid crossbar when it is configured to hold the partial products resulting from the multiplication operation. The latter means that, during the programming procedure, the internal horizontal control switches are activated to impose left-shift to the programming signals. Finally, the topology is converted to a typical crossbar before the read-out process takes place. Therefore, with the same multiplicand $(312)_4$ and multiplier “11” like in the previous example, now the bottom word is expected to store the input word after shifting it once to the left. The simulation procedure involved a programming phase and two sequential reading phases for the two rows. The results in Fig. 6.14 confirm the correct operation of the reconfigurable crossbar;

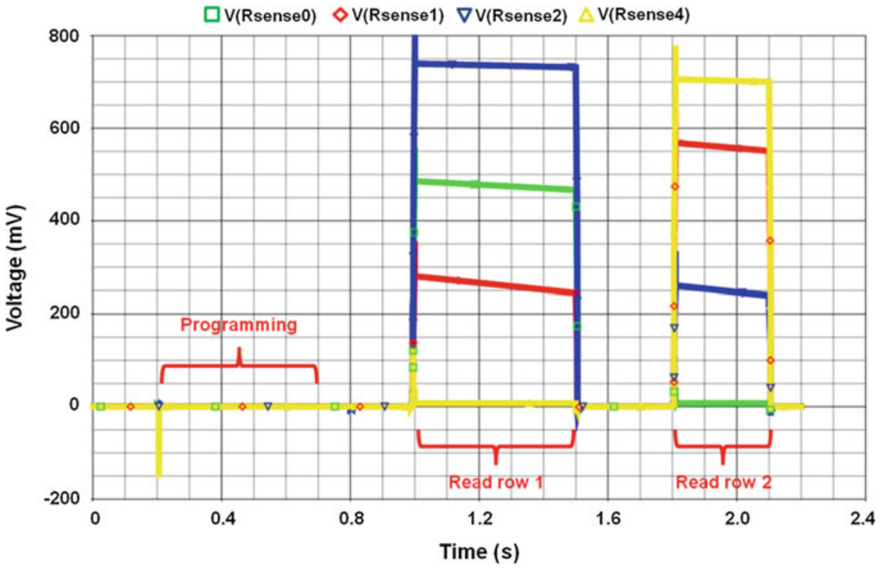


Fig. 6.14 SPICE simulation results for a 2 × 4 multi-level reconfigurable crossbar memory configured to hold the two partial products resulting from the multiplication operation between $(312)_4$ and $(11)_2$

the measured voltages on the sense resistors for the first word correspond to the number $(0312)_4$, whereas for the second word we get $(3120)_4$.

Furthermore, in the last simulation we calibrated the value of the sense resistors, thus improving significantly the voltage margins which correspond to the different stored states. This calibration involved the application of the same programming signal in all crossbar columns and next probing with different sense resistance values to maximize the output voltage levels. This improvement is apparent when comparing the read-out voltage levels of the first row between Figs. 6.13b and 6.14, where the same radix-4 number is stored.

6.5 Overview and Discussion

The outcome of this chapter concerns novel dense memory architectures, based on emerging nano-electronic devices, to be used in future computing systems. We presented a novel multi-level crossbar-based memory architecture, which supports arithmetic encoding of high-radix data within multi-state composite memristive switches (MSS), rather than in single memristors. Novel parallel read and write schemes were discussed, which support the fine control and robust detection of the different stored states. Furthermore, an enhanced hybrid version of the typical crossbar topology was proposed, including additional internal CMOS switches among the cross-points, which enable the fast computation of partial products and their storage in the memory itself.

A prominent feature of this multi-level memory architecture is the high storage density due to the utilization of compact nanoscale devices and circuits, such as the crossbar topology and the memristor. As mentioned previously, such an $n \times n$ multi-level hybrid memory block would consist of n^2 MSS. Given that between almost every two columns and/or rows of the reconfigurable crossbar there are internal CMOS transistors, the distribution of the memristive cross-points will depend on the corresponding transistor distribution in the CMOS layer(s), because the crossbar nanowires are normally deposited later, along with the memristors, on top of the CMOS layer(s). In order to estimate the total chip area of such a crossbar-based multi-level memory architecture, one must also consider the area occupied by the internal CMOS transistors, the necessary vias, as well as the number of transistors in the peripheral and control circuitry; i.e. the read/write circuitry, the row/column decoder, and the data-encoding/decoding circuits. The aforementioned peripheral circuits result in chip area overhead comparable to that in current Flash nonvolatile memory. However, the increased memory density by the multi-level memristive storage cells is expected to overcome the area overhead for the required peripheral circuitry.

Furthermore, in large crossbar arrays the read margins are expected to significantly degrade due to the current sneak-path effect and the small equivalent resistance of the memristive multi-level cross-points; a similar effect was observed in Chap. 5 in crossbar arrays comprising anti-parallel memristors as cross-point devices. Nevertheless, using more sophisticated types of MSS, like those presented in Chap. 3, and/or circuit level approaches to the mitigation of the current sneak-paths, will result in more noise-tolerant implementations. Of course, replacement of MSS cross-point devices with single memristors, and of CMOS computing circuitry (most of it, if not all of it) with memristor-based circuits, will eventually improve density and power consumption of such an architecture.

Several published works in the literature lately focus on the exploitation of the multi-bit storing property of memristors and networks of memristors, for digital signed digit arithmetic circuits based on balanced or redundant numeral systems [30–39]. In this context, the proposed conceptual solution of a memristive arithmetic and logic unit (ALU) could facilitate the faster implementation of arithmetic algorithms. Consequently, high-density memristive data storage, combined with memristive circuit paradigms and novel memristive arithmetic units, certainly pave the way for a promising memristive era in electronic computing systems.

References

1. R.P. Brent, P. Zimmermann, *Modern Computer Arithmetic* (Cambridge University Press, Cambridge, 2010)
2. International Technology Roadmap for Semiconductors (ITRS) (2013). Available: http://www.itrs.net/. Accessed June 2014
3. S. Hamdioui, H. Aziza, G.C. Sirakoulis, "Memristor Based Memories: Technology, Design and Test, in *9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, Santorini island, Greece (2014)
4. Y. Pershin, M. Di Ventra, Practical approach to programmable analog circuits with memristors. IEEE Trans. Circ. Syst. I, Reg. Papers **57**(8), 1857–1864 (2010)
5. E. Lehtonen, M. Laiho, Stateful implication logic with memristors, in *IEEE/ACM International Symposium on. Nanoscale Architectures (NANOARCH)*, San Francisco, CA (2009)
6. S. Kvatinsky, G. Satat, N. Wald, E.G. Friedman, A. Kolodny, U.C. Weiser, Memristor-based material implication (IMPLY) logic: design principles and methodologies, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **22**(10), 2054–2066 (2014)
7. S. Paul, S. Bhunia, A scalable memory-based reconfigurable computing framework for nanoscale crossbar. IEEE Trans. Nanotechnol. **11**(3), 451–462 (2012)
8. G.S. Rose, J. Rajendran, H. Manem, R. Karri, R.E. Pino, Leveraging memristive systems in the construction of digital logic circuits. IEEE Proc. **100**(6), 2033–2049 (2012)
9. G. Papandroulidakis, I. Vourkas, N. Vasileiadis, G.C. Sirakoulis, Boolean logic operations and computing circuits based on memristors. IEEE Trans. Circuits Syst. II Expr. Briefs **61**(12), 972–976 (2014)
10. R. Patel, E.G. Friedman, Arithmetic encoding for memristive multi-bit storage, in *20th IEEE/IFIP International Conference on VLSI and System-on-Chip (VLSI-SoC)*, Santa Cruz, CA (2012)
11. L. Yang, Architectures for memristor-based storage structures (2011). Available: http://dukespace.lib.duke.edu/dspace/handle/10161/5013. Accessed 1 Mar 2015
12. L.O. Chua, The fourth element. IEEE Proc. **100**(6), 1920–1927 (2012)
13. C.E. Merkel, N. Nagpal, S. Mandalapu, D. Kudithipudi, Reconfigurable N-level memristor memory design, in *International Joint Conference on Neural Networks (IJCNN)*, San Jose, CA (2011)

14. K. Hyongsuk, M.P. Sah, C. Yang, L.O. Chua, Memristor-based multilevel memory, in *12th International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, Berkeley, CA (2010)
15. H. Manem, J. Rajendran, G.S. Rose, Design considerations for multilevel CMOS/Nano memristive memory. ACM J. Emerg. Technol. Comput. Syst. **8**(16), 1–22 (2012)
16. A. Emara, M. Ghoneima, M. El-Dessouky, Differential 1T2M memristor memory cell for single/multi-bit RRAM modules, in *6th Computer Science and Electronic Engineering Conference (CEEC)*, Colchester (2014)
17. D. Fey, Using the multi-bit feature of memristors for register files in signed-digit arithmetic units. Semicond. Sci. Technol. **29**, 104008 (2014)
18. Y.V. Pershin, M. Di Ventra, Memory effects in complex materials and nanoscale systems. Adv. Phys. **60**(2), 145–227 (2011)
19. E. Linn, R. Rosezin, S. Tappertzhofen, U. Bottger, R. Waser, Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations, Nanotechnology **23**, 305205 (2012)
20. M. Di Ventra, Y.V. Pershin, The parallel approach. Nat. Phys. **9**, 200–202 (2013)
21. K.H. Kim, S. Gaba, D. Wheeler, J.M. Cruz-Albrecht, T. Hussain, N. Srinivasa, W. Lu, A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. Nano Lett. **12**(1), 389–395 (2012)
22. H. Kim, M.P. Sah, C. Yang, T. Roska, L.O. Chua, Neural synaptic weighting with a pulse-based memristor circuit. IEEE Trans. Circ. Syst. I Reg. Papers **59**(1), 148–158 (2012)
23. C. Yakopcic, R. Hasan, T.M. Taha, M. McLean, D. Palmer, Memristor-based neuron circuit and method for applying learning algorithm in SPICE?, IET Electron. Lett. **50**(7), 492–494 (2014)
24. H. Manem, G.S. Rose, X. He, W. Wang, Design considerations for variation tolerant multilevel CMOS/nano memristor memory, in *20th Great Lakes Symposium on VLSI (GLSVLSI)*, Providence, Rhode Island (2010)
25. P. Junsangsri, F. Lombardi, A memristor-based TCAM (ternary content addressable memory) cell: design and evaluation, in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, Salt Lake City, Utah, USA (2012)
26. I. Vourkas, G.C. Sirakoulis, On the analog computational characteristics of memristive networks, in *20th IEEE International Conference on Electronics, Circuits and Systems (ICECS),* Abu Dhabi (2013)
27. I. Vourkas, G.C. Sirakoulis, On the generalization of composite memristive network structures for computational analog/digital circuits and systems. Microelectron. J. **45**(11), 1380–1391 (2014)
28. I. Vourkas, A. Batsos, G.C. Sirakoulis, SPICE modeling of nonlinear memristive behavior, Int. J. Circ. Theor. Appl. **43**(5), 553–565(2015)
29. Y. Yilmaz, P. Mazumder, Threshold read method for multi-bit memristive crossbar memory, in *Int. Symposium on Electronic System Design (ISED)*, Kochi, Kerala (2011)
30. K. Cho, S.J. Lee, K. Eshraghian, Memristor-CMOS logic and digital computational components. Microelectron. J. **46**(3), 214–220 (2015)
31. A.A. El-Slehdar, A.H. Fouad, A.G. Radwan, Memristor-based redundant binary adder, in *International Conference on Engineering and Technology (ICET)*, Cairo (2014)
32. A.A. El-Slehdar, A.H. Fouad, A.G. Radwan, Memristor-based balanced ternary adder, in *25th International Conference on Microelectronics (ICM),* Beirut (2013)
33. A.A. El-Slehdar, A.H. Fouad, A.G. Radwan, Memristor based N-bits redundant binary adder. Microelectron. J. **46**(3), 207–213 (2015)
34. M. Laiho, E. Lehtonen, Arithmetic operations within memristor-based analog memory, in *12th International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, Berkeley, CA (2010)
35. K. Bickerstaff, E.E. Swartzlander, Memristor-based arithmetic, in *44th Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, Pacific Grove, CA (2010)

36. F. Merrikh-Bayat, S. Bagheri Shouraki, Memristor-based circuits for performing basic arithmetic operations, in *Procedia Computer Science—Proceedings of 2010 World Conference on Information Technology (WCIT),* vol. 3 (2011), pp. 128–132
37. A.H. Shaltoot, A.H. Madian, Memristor based carry lookahead adder architectures, in *55th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boise, ID (2012)
38. Y. Yang, J. Mathew, D.K. Pradhan, M. Ottavi, S. Pontarelli, Complementary resistive switch based stateful logic operations using material implication, in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Dresden (2014)
39. S.J. Lee, B.S. Park, S.W. Cho, K. Cho, K. Eshraghian, Memristor-CMOS reconfigurable multiplier architecture, in *14th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*, Notre Dame, IN (2014)

Chapter 7
Networks of Memristors and Memristive Components

7.1 Introduction

Back in the early days of digital circuits, analog computations embodied a whole area of research which, however, was not as scalable and/or reproducible as former digital solutions. This is one of the main reasons why analog computing was not given much of attention afterwards. Nevertheless, nowadays there are very important engineering and modeling problems which are still very difficult to be addressed digitally, hence calling for innovative analog-based computational methods and structures [1]. In the same context, for decades researchers were convinced that real artificial intelligence (AI) would never be done on conventional and rigidly adherent to Boolean logic hardware (HW), where processing and memory units are strictly separated [2, 3]. However, not until when HP Laboratories built the first memristor prototype in 2008, it was believed that creating something with the function principles of a brain could finally be possible [4–6].

As described in previous chapters, memristors demonstrate a natural basis for computation that is different from familiar paradigms and combines information processing and storage in the memory itself. Owing to their analog memory functionality, an extraordinary type of computing parallelism was introduced in [7], dubbed as *memcomputing*. This type of analog parallelism consists in array-like structures which accommodate large numbers of memristors (or generally *memelements* [8]) where complex, unconventional, and/or neuromorphic computations take place. *Memcomputing* has received much attention because of the ability of such memristive architectures to store and process information on the same physical platform; a major conceptual and practical departure from the present day's von Neumann machine-based architectures [2, 9]. The strength of memcomputing is essentially based on the massively-parallel analog dynamics of many interconnected memristors and the ability to recover the result of the computation from the same computing units, much like the brain is thought to operate. Examples of computing approaches based on memristors (or *memelements*) include

I. Vourkas and G.Ch. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Emergence, Complexity and Computation 19,
DOI 10.1007/978-3-319-22647-7_7

massively-parallel computing with memristive ensembles [10–13], logic design paradigms like those presented in Chap. 4 [14–19], memristive cellular automata (CA) and cellular neural networks (CNN) [20, 21], as well as neuromorphic computing with memristive synapses [22, 23]. The latter, for example, which aims to use biological mechanisms within the brain as a blueprint for novel computer architectures [24–28], is one of the most promising applications of memristors [6]. Nevertheless, there are complex challenges that can be solved with the help of memristors without necessarily mimicking the synaptic functionality. As we will show here, a very powerful computing structure which implements analog parallel computations is the so called memristive grid (network).

In such structure there is continuous information exchange during calculations which renders a tremendous increase of computational power due to the massively-parallel network dynamics; calculation consists in the evolution of the states of all the involved devices. In spite of being still at an early stage, related work in memristive grid-based computing has shown promising results in graph theory optimization problems [10, 11] and signal processing [29, 30]. However, most research approaches have so far focused on homogenous networks which mainly consist of ensembles of identical and reciprocal (connected either in parallel or in series) memristors in hopes of achieving structural (thus manufacturing) simplicity. As a consequence of the symmetry of interconnections, though, we will see that such type of grid sometimes fails to provide a unique solution to the problem since it hardly converges to an easily distinguishable steady state during computation. It is the same symmetry that also impedes the appropriate mapping (projection) of the target problem-space (more generally a target application described in the form of a directed graph) on the computing medium.

From a sea-of-existing inherently complex problems, in terms of computation time, in this chapter we address two of the probably most well-known and well documented, i.e. the shortest path and maze-solving problems, via computations in memristive grids. Solution of the shortest path problem (SPP) has always been a hot topic in graph theory because of its wide application field. There are three well-known alterations of this problem: (i) the single source SPP; i.e. the shortest path from a given vertex of a graph to all others, (ii) the all pairs SPP; i.e. the shortest path between all possible pairs of the vertices, and (iii) the single source single destination SPP. The inherent difficulty in this problem stems from the fact that any two points of the plane are connected by multiple and often degenerate paths, which complicates the finding of a single optimal solution. On the other hand, maze-solving is a tour puzzle, i.e. a complex arrangement of pathways in which the correct path must be found; e.g. the shortest path to enter and exit the maze. Mazes have fascinated people ever since the ancient times. According to an ancient Greek myth, Ariadne gave a ball of thread to Theseus so that he could find his way out of the Minotaur’s labyrinth (we do not distinguish between mazes and labyrinths in the context of this chapter). Ariadne’s thread, named after the heroine of the myth, is nowadays how we call the solving approach of a problem with multiple apparent means of proceeding through exhaustive application of logic to all available routes. Nakagaki et al. in [31] showed that the primitive biological organism Physarum polycephalum can solve a maze in which food has been placed at the entrance and exit; after 8 h the mould changed its shape to that of a single tube connecting the two food sources via the shortest possible path. Pershin et al. were later inspired by this behavior and proposed an early approach to solve mazes with the help of memristive networks [10].

At present, little is still known regarding the consequences of embedding memristors within interconnected network architectures. To explore this concept, numerical and circuit simulations were conducted for the purpose of investigating the network dynamics utilizing the well-documented physics of single memristor devices and known network topologies. In the rest of this chapter, inspired by the aforementioned myth, we will show how the grid could be guided in order to provide better solutions by laying a *memristive thread* across several nodes. We further extend some already proposed memristor network-based approaches by introducing certain modifications in the computing platform, which here may comprise sophisticated memristive structures other than just reciprocal devices. Additionally, we address the proper application-mapping issue via a unique modeling approach which comprises specific memristive circuit models for several types of edges connecting the graph vertices. The presented methodology takes advantage of the polarity-dependent switching of bipolar memristors and enables the precise network projection of any mesh-based directed graph with n vertices arranged on a discrete lattice with nonnegative connection weights. Appropriate selection of the connecting components makes all the memristive network links “resistive-equivalent”; hence no yield loss or interference by the network asymmetry is observed in computations. Several scenarios are examined considering also the inclusion of devices with different switching characteristics in the same computation. The dynamically changing state of this adaptable medium, in response to time-dependent signals or changes in the grid configuration, is monitored and thoroughly discussed. Explicit computing simulation examples, using either a developed memristive network simulator and/or PSPICE simulation environment, provide intuition into the capabilities and the weaknesses of this new class of computing HW where the employed devices themselves retain the result of the computations. The emergence of unforeseen functionalities opens doors to exciting new computing concepts and encourages the development of parallel memristive computing systems which could have a large impact in several practical problems (e.g. in urban planning, routing, scheduling, robotics, etc.). Finally, the presented assisted-evolution of the memristive grid raises the point that the famous thread of Ariadne was made of memristors [32].

7.2 Memristive Network-Based Computations

Appropriately interconnected memristors significantly improve the efficiency of computations via massive parallelism. In its more generic form, the assumed memristive grid comprises a mesh of nodes which are connected by computing components involving bipolar memristors, resistors, and/or switches (transistors); the transistors have two roles: (i) they facilitate mapping of the target problem-space onto the HW network when the latter is regular and homogenous as in [10]; (ii) they provide independent access to individual memristors for the purpose of programming or reading of their state. At present it is inevitable that such dynamical circuits must be interfaced with conventional electronics to produce and observe signals. Therefore, it is assumed that external signals can be applied to any grid nodes for the purpose of initializing the memristors' states as well as to read the calculation results by a memristance read-out of all memristors.

7.2.1 Description of the Computing Platform and Its Function

Overall, the massively-parallel operation of such platform consists in three main stages: (i) initialization; (ii) computation; and (iii) reading of the outcome. The computation stage normally consists in the application of voltage pulses of appropriate amplitude and duration across specific nodes of the grid, while grounding others. Computation is initiated by triggering the source nodes and then a wave of stimulation propagates in all directions and affects accordingly the states of the rest of the devices. In any moment the potential at all grid points can be found as a solution of Kirchhoff's current law (KCL) equations. The time required for the grid to reach to a steady state during calculations depends on the applied signals, the network topology, and the switching characteristics of the individual memristors. The computation result is given by a sub-set of components which are found in a predefined resistive state. Normally all memristors are initially set in the high resistive state (R_{OFF}). So, the network marks the solution with the lowest corresponding memristances (or intermediate states if memristors did not have enough time or voltage across them to completely switch their state), while it successively finds all possible solutions in a determined order depending on the number of memristive connections involved in each one of them. Owing to their nonvolatility, all the memristive devices maintain the resistive states after the input signals have been removed.

Figure 7.1a shows an indicative snapshot of a regular memristive computing grid. Between vertically and horizontally neighboring nodes there is a memristive circuit which can vary according to the target application, as shown in Fig. 7.1b. Moreover, Fig. 7.1c shows two possible ways of creating memristive devices with higher switching thresholds to be used in the grid. In fact, higher thresholds could be achieved either by incorporating pairs of anti-parallel diodes in series with the memristors [33], or by following the concept of composite memristive structures presented in Chap. 3 [34, 35]. The latter consists in using small networks of memristors which, as a whole, exhibit the same overall memristance range [R_{ON}, R_{OFF}] with single memristors but can have n-fold (n×) cumulative switching thresholds.

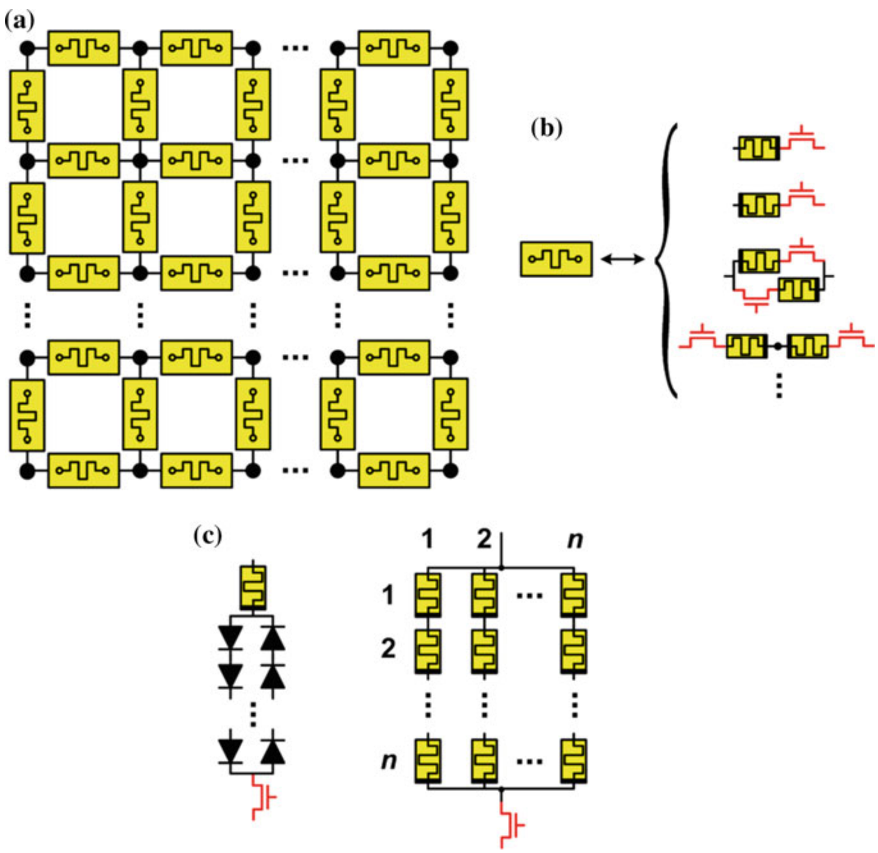


Fig. 7.1 Basic setup of a memristive grid. **a** Typical mesh geometry. **b** Different options of memristive computing components. **c** Possible approaches to composite memristive devices with higher switching thresholds

Depending on the nature of the problem to be solved, the direction of current flow in the grid is not always known a priori. When the current direction is important, then single forward-polarized memristors, whose polarity is in line with the desired current flow, are suitable for modeling such connections; here we assume that their memristance will decrease when forward-biased and it will increase (or remain unaffected in the initially set R_{OFF} state) when reversely-biased, as shown in Fig. 7.2a, b. For threshold-type switching memristors we consider that the resistance change-rate is small below (fast above) a voltage threshold (namely V_{SET} or V_{RESET}), which is viewed as the minimum voltage required to impose a change on the physical structure (and thus the memristance) of the device. On the other hand, if current direction is not a constraint, then a pair of series or parallel

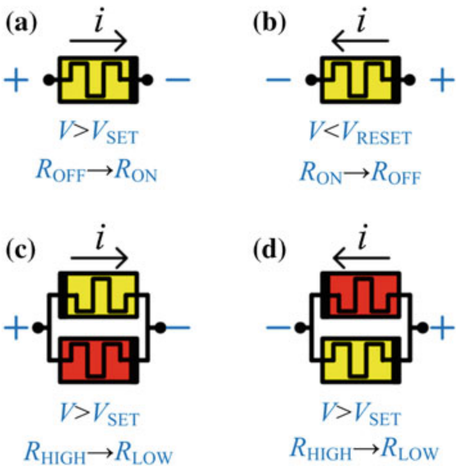


Fig. 7.2 Polarity-dependent switching of bipolar memristors. **a**, **b** Single memristors switch their states in a reciprocal manner when forward or reversely biased, provided that the applied voltage exceeds the corresponding SET or RESET voltage thresholds. **c**, **d** Independency on the sign of the applied voltage is achieved via pairs of parallel memristors where only the forward-biased device (*marked with red color*) is finally SET

memristors with opposite polarity is used, as shown in Fig. 7.2c, d; thus, the symmetry of the memristive compositions provides operation-independency on the sign of the voltage at their common terminals. Depending on the polarity of the corresponding applied voltage at their terminals, each time only the device which is forward-biased will change state. As a consequence, for the parallel connection shown here the composite resistance changes from high ($R_{OFF}||R_{OFF}$) to low ($R_{OFF}||R_{ON}$), where operator || denotes parallel connection of two resistive elements. Likewise, for the series connection of reciprocal memristors the composite memristance changes from high ($R_{OFF} + R_{OFF}$) to low ($R_{OFF} + R_{ON} \approx R_{OFF}$).

An indicative example of memristive modeling for graph connections with current flow constraints is shown in Fig. 7.3. Appropriately polarized memristors are sufficient to model properly the edges of the graph. A slowly ramped waveform voltage is applied to the source node whereas the destination node is grounded. After a certain amount of time the externally applied input voltage gets high enough so that the corresponding voltage drop on the forward-polarized memristors exceeds their V_{SET} threshold. As a consequence, the devices belonging to this sub-set are found in a lower resistive state and they mark the solution of the problem. It is expected that the devices forming the final solution will change state simultaneously. However, as we will demonstrate later, it is worth noting here that this depends on the actual topology of the graph/network and on the complexity of every solution, which sometimes affect the order in which the memristors change state.

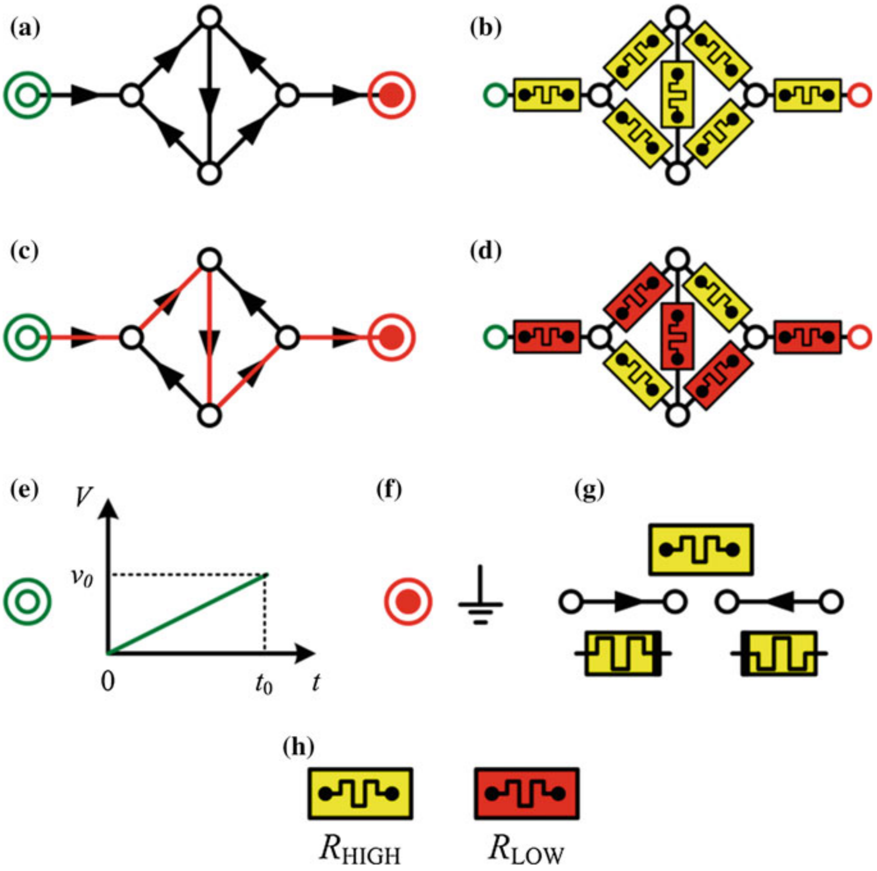
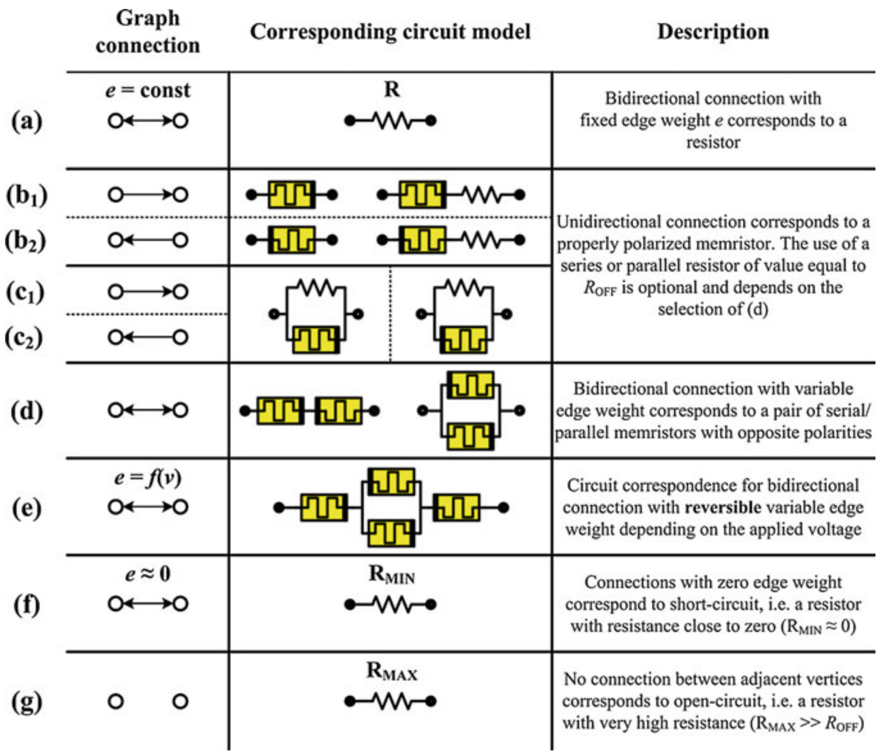


Fig. 7.3 Memristive modeling example for connections where current flow is important. **a** Problem space mapped on a directed graph. **b** The corresponding memristive circuit. **c**, **d** The unique solution to the SPP is marked with *red color* highlighting the graph edges and the memristive components that are involved. **e** The plot of the input voltage applied to the source node. **f** The ground applied to the destination node. **g** Appropriately polarized memristors used to model the particular graph connections. **h** Resistive states of the memristive components

7.2.2 *Memristive Circuits for Modeling Edges of Directed (Oriented) Graphs*

Given the variety of possible interconnections found within a target application graph with *n* vertices, which are arranged on a discrete lattice with nonnegative link-weights, in Fig. 7.4 we propose a list of corresponding circuit modeling approaches (the transistors in series to all memristors are omitted for simplicity). When a given graph is first mapped onto the network, connections between the vertices can be either unidirectional, bidirectional, or completely closed. By



	Graph connection	Corresponding circuit model	Description
(a)	e = const	R	Bidirectional connection with fixed edge weight e corresponds to a resistor
(b_1)			Unidirectional connection corresponds to a properly polarized memristor. The use of a series or parallel resistor of value equal to R_{OFF} is optional and depends on the selection of (d)
(b_2)			
(c_1)			
(c_2)			
(d)			Bidirectional connection with variable edge weight corresponds to a pair of serial/ parallel memristors with opposite polarities
(e)	$e = f(v)$		Circuit correspondence for bidirectional connection with **reversible** variable edge weight depending on the applied voltage
(f)	$e \approx 0$	R_{MIN}	Connections with zero edge weight correspond to short-circuit, i.e. a resistor with resistance close to zero ($R_{MIN} \approx 0$)
(g)		R_{MAX}	No connection between adjacent vertices corresponds to open-circuit, i.e. a resistor with very high resistance ($R_{MAX} >> R_{OFF}$)

Fig. 7.4 Circuit models **a**–**g** corresponding to a variety of given connections between vertices of directed graphs

following the provided list of options, any kind of directed graph whose edges are of equal weight, can be easily projected. Unequal weights could be modeled, as well, by using memristors with different switching characteristics, e.g. different memristance ratio and/or voltage thresholds.

According to Fig. 7.4, typical resistors are used: (a) to model connections which can be by default included or excluded from the solution (e.g. by setting their value equal to the less resistive or the high resistive state of the used memristive components); (g) to represent open circuits (the resistor has much higher resistance than the higher equivalent resistance of a memristor or a memristive composition); (f) or short circuits (very small resistance). Moreover, the series/parallel coupling of memristors with resistors to model unidirectional links in (b) and (c) should be selected according to the choice made for the bidirectional links in (d) in order to maintain a common memristance range for every memristive connection between the nodes of the grid. For example, if anti-parallel memristors are selected from (d) for the bidirectional links, then their memristance varies within [(R_{ON}||R_{OFF}), 1/2 × R_{OFF}]; this exact memristance range also corresponds to the parallel connection of a memristor with a resistor whose resistance is equal to R_{OFF}. For directed graphs with purely unidirectional links single memristors would do fine, whereas for purely bidirectional connections we choose only reciprocal in-series or parallel memristors. The choice between the latter impacts the required input voltages since series elements require higher voltages to switch their states due to the voltage divider effect. Case (e) is a special type of memristive component, combining a pair of anti-parallel with a pair of anti-series memristors, where the latter have higher switching thresholds than those connected in parallel. Its use will be further explained in the following sections.

7.3 Path Computing and Maze-Solving with Ariadne's Memristive Thread

Next we study the behavior of the memristor grid in several scenarios and experiment with the presented composite memristive structures, as well as with different switching properties of the devices which provides the ability to interfere in the computation. In this section we will show how, what appears to be a competition for the available applied voltage across interconnected devices, can be exploited in shortest path and maze-solving computations. For the purpose of this study we developed a GUI-based memristive network simulator using the Easy Java Simulations (EJS) environment [36]; the system of equations to be solved is similar to that of XbarSim [37] described in Chap. 5 and all differential equations of the employed threshold-type memristor model are numerically solved using a 4th order Runge-Kutta integration method, as implemented in [36]. For better visualization of the memristance change induced to each composite computing component, a linear color scale was used to represent all possible equivalent memristance values. Given the proven precise quantitative match of the SPICE version of the employed model [38], we also selectively performed circuit simulations using the Cadence PSPICE environment in order to validate the outcome of our computations. The values of the parameters of the model are set as given in Chap. 2 with $\{R_{\mathrm{ON}}, R_{\mathrm{OFF}}\} = \{2, 200\}$ kΩ and $\{V_{\mathrm{SET}}, V_{\mathrm{RESET}}\} = \{2, -1\}$V.

7.3.1 Fully Interconnected Network

In order to demonstrate the fundamental principles of operation of the memristive grid, we simulated a 9 × 9 regular mesh where we first considered the solution of the SPP along various directions between two (or more) pre-selected nodes. In our first demonstration we assume a fully interconnected undirected mesh; therefore, between every pair of neighboring grid nodes we place two identical anti-parallel memristors initially set in R_{OFF}. A typical computation is initiated by applying a voltage pulse of a suitable amplitude and duration to the source node(s) while grounding the destination node(s). The sub-set of components that first switch to the less possible resistive state ($R_{ON}||R_{OFF}$) gives always the result of computation.

Figure 7.5 shows the simulation results of the first case; Fig. 7.5a provides the visualization of the network evolution during the calculation stage by focusing on three distinct time points where up to three different solutions are given. The grid nodes are colored black and the memristance range of all devices employed in computation is given according to the provided color map (R_{MIN} and R_{MAX} denote the min and max composite resistance of each memristive connection). Figure 7.5b illustrates the current-time plot of the currents flowing through all the 144 memristive components (i.e. pairs of anti-parallel memristors) involved in computation, taken from PSPICE. The vertical dashed lines denote the time moments t_1 and t_2 which correspond to the snapshots shown in Fig. 7.5a. Our results are in very good qualitative agreement with those found in the literature [11]; the expected shortest path is the first to emerge during computation. Two alternate equivalent solutions appear simultaneously after the best solution has been calculated (negative measured currents are due to a typical SPICE convention for the current flowing inwards or outwards of a particular device, depending on which terminal the ammeter is placed).

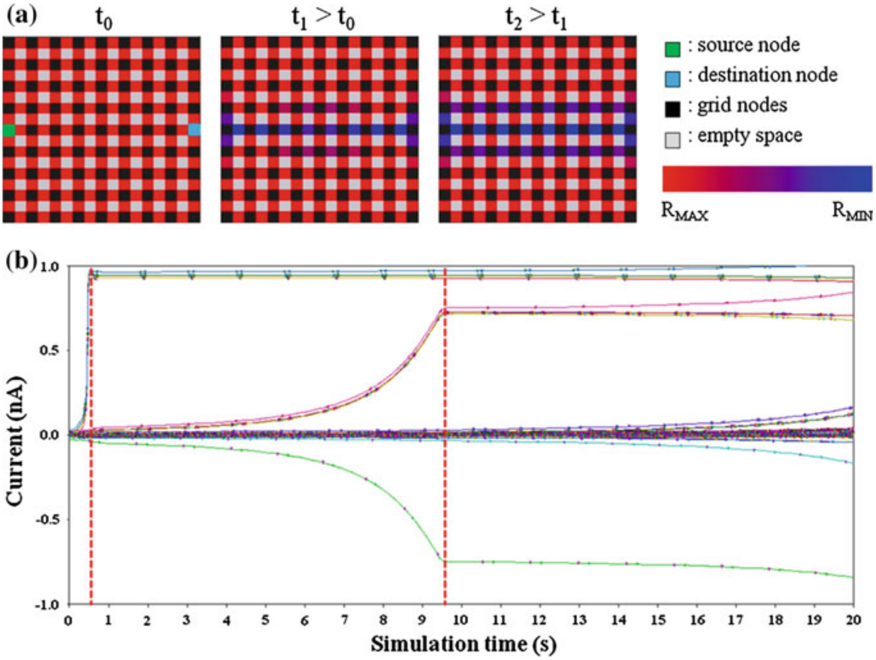


Fig. 7.5 Solution of the SPP for the pair of indicated nodes. **a** Shows the evolution of computation and **b** provides the measured currents flowing through all memristive connections (involving two anti-parallel memristors) in SPICE. Memristance limits {R_{MIN}, R_{MAX}} correspond to {($R_{ON}||R_{OFF}$), 1/2 × R_{OFF}}

In such solutions, the total resistance of every path is proportional to its length. The network has no external clock and finds all the solutions in parallel, although the final read-out requires subsequent resistance measurements. If there are multiple paths, then the shortest one would contain less memristors and, thus, evokes less resistance than the longer ones. All intermediate paths (in terms of length) offer a proportionate resistance. Therefore, since current flows in inverse proportion to the resistance of a path, the change of state of a given path is proportional to the current in the path. Even more paths are revealed as computation continues in time (it is not shown in this example); alternate shortest paths then can be identified by the different state their memristors have during (or after) the switching process. However, multi-state reading with memristors is difficult, so it is preferable to distinguish between fully-switched devices other than devices that are in intermediate resistive states. Between memristive components from two possible paths, those of the shorter path switch their state at an earlier moment than those of the longer path. This allows sorting all possible solutions according to their length. Since all neighbor connections in the network are considered to be of equal weight, the aggregate length of a particular path is here identified by the total number of "hops" in-between source and destination nodes.

After a definite time, almost all the memristive components are expected to finally switch completely or reach an intermediate state, which will possibly ruin the reading process afterwards. Of course, less switching events are expected if the memristors are supposed to be affected exclusively by voltages which exceed their thresholds (zero state-drift corresponds to parameter $b = 0$ in the model presented in Chap. 2). In this circuit simulation example the source node is connected to a DC voltage source whose amplitude is set high enough so as to exceed the accumulated threshold value of the amount of devices belonging to the (easily observed) shortest path. However, when the source/destination nodes are located along an arbitrary direction different from the network symmetry directions (i.e. that cannot be vertically or horizontally connected directly), the minimum necessary voltage amplitude, which will shortly lead the computation to a unique solution, is hard to be estimated. Therefore, it is preferable to apply a slowly ramped waveform voltage while searching for the proper amplitude. This type of input voltage is used in the rest of presented simulation scenarios.

Let us now consider the behavior of the grid in more sophisticated shortest path computations, either when trying to move diagonally or when trying to get past an obstacle in order to reach a destination node (in other words, when the network is damaged). Figure 7.6a depicts the evolution of computation of the shortest path going from node (row, column) = (5, 1) to node (9, 3). Two intermediate computing stages are shown, as well as the final steady state to which the grid reaches after a stipulated amount of time. Figure 7.6b similarly illustrates the current-time plot of all measured currents during circuit simulation using PSPICE, where the vertical dashed line denotes the moment when the network approaches to a steady state as in the final schematic of Fig. 7.6a. However, unlike the previous example which had a straightforward solution, here many paths emerge almost simultaneously, thus making it hard to distinguish a particular optimal solution. The steady state of the

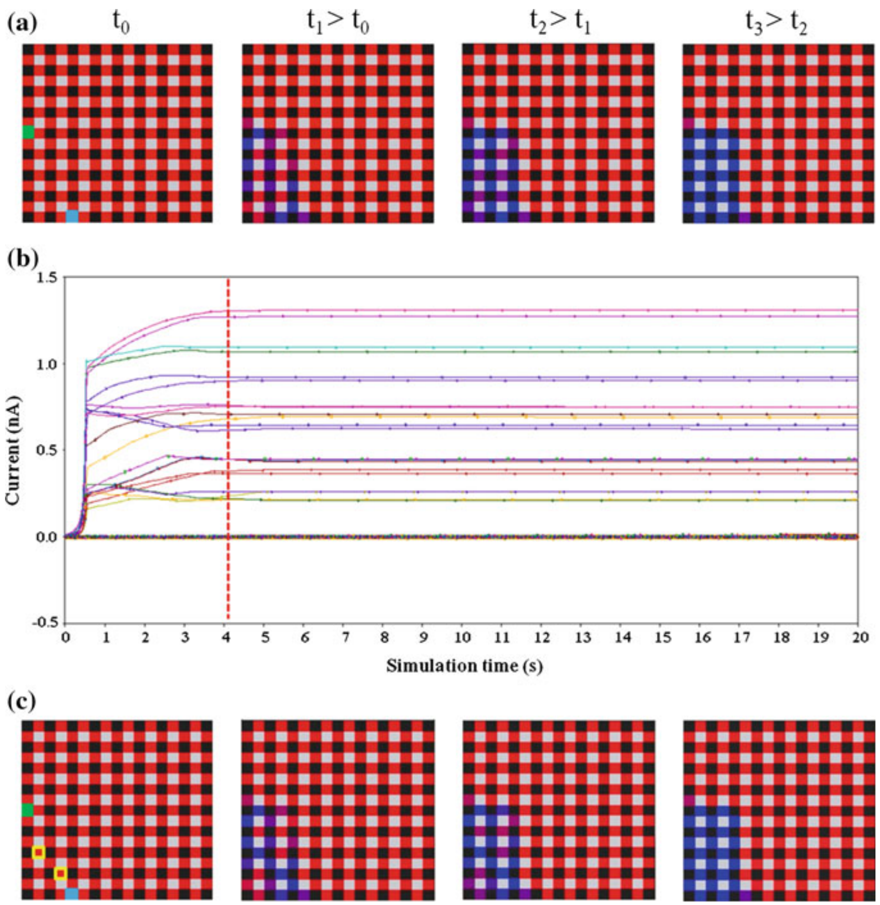


Fig. 7.6 Solution of the SPP given by the grid when the source and destination nodes (colored similarly as in Fig. 7.5) are selected not along a symmetry direction. **a** Provides the evolution of computation and in **b** the measured currents flowing through all memristive components are shown. **c** Shows the evolution of computation when two particular components (indicated by *yellow rectangles*) have been adjusted to switch at lower thresholds

grid consists in all possible shortest path combinations within the 5-by-3 rectangle located in the bottom-left part of the mesh.

Therefore, we can conclude that when the selected nodes are located along an arbitrary direction within a regular grid (i.e. when $\text{row}_{\text{source}} \neq \text{row}_{\text{destination}}$ and $\text{column}_{\text{source}} \neq \text{column}_{\text{destination}}$), this computing platform fails to uniquely solve the problem because of its inherent symmetry of interconnections and of the employed devices. We remind here that the simulations are based on voltage-controlled threshold-type switching bipolar memristors. Because of this, the result of Fig. 7.6a diverges from similar published simulation results in the literature, which were based on current-controlled bipolar memristive devices. Specifically, in [11] the authors observed a self-reinforcement of the shortest path solution with time due to the memristive network dynamics; the least resistive path distinguished faster by attracting more and more current. However, in order to achieve that it was assumed that the rate of memristance (M) change was proportional to the current flowing through the devices [4, 39] as shown below:

$$\frac{dM_{ij}}{dt} = \alpha \cdot I_{ij}(t) \tag{6.1}$$

where α is a constant and $I_{\text{ij}}(t)$ is the current flowing through the memristive connection (ij). Unless Eq. 6.1 is complied, such dynamics are unfortunately retained.

Figure 7.6c presents a variation of the same scenario where, inspired by the famous Greek myth of Ariadne, we tried to lead the path computation through particular intermediate points by spreading a “memristive thread” along the desired path. More specifically, we chose to exploit network heterogeneity by employing devices with different switching characteristics in the same computation. We particularly assigned lower switching thresholds for some selected memristors (they are highlighted in the first snapshot) which are found in an alternate path that we wish to follow. The simulation shows that the first clearly emerging solution appears faster than before (only after t_1) and follows the desired path in a zig-zag form. However, as computation continues in time, more paths appear gradually and the steady state of the grid coincides with that of Fig. 7.6a.

7.3.2 Defective Network

In the next simulation scenario, a few connections of the mesh were removed (i.e. replaced with R_{MAX} resistors) intentionally in order to test the stability of the shortest path problem solution around a defective region of the grid (i.e. an obstacle). According to simulation results in Fig. 7.7a, the computation evidently tends to reach the shortest possible path around the damaged area. We note here that the memristive network has a remarkable ability to repair damaged solutions. Indeed, this property is close to the self-healing ability that can be ascribed to systems or processes which, by nature or by design, tend to correct any disturbances. The missing connections have been removed intentionally in an asymmetric fashion in order to show that the healing occurs along the shortest possible path around the damaged region. However, as Fig. 7.7b confirms, again multiple paths emerge almost simultaneously making it hard to follow the optimal solution. Within a particular simulation time, all possible shortest path combinations that pass from

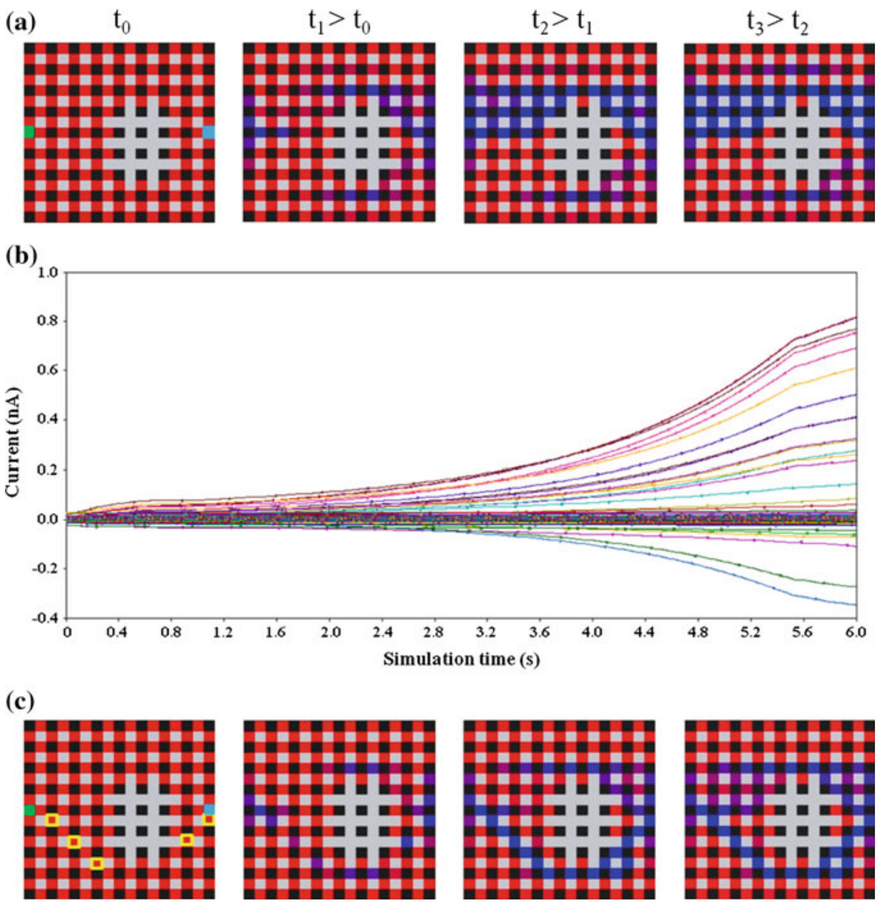


Fig. 7.7 Solution of the SPP given by the network when a damaged region intervenes between the source and the destination nodes (colored similarly as in Fig. 7.5). **a** Provides the evolution of computation and **b** shows the measured currents flowing through all the memristive components during computation. **c** Shows the evolution of computation when five particular vertical components (denoted by *yellow rectangles*) were adjusted to switch at lower thresholds

the most convenient side of the defective area, are indicated. Likewise in the previous case, we again spread memristive components which comprise devices with lower switching thresholds within the grid, aiming to guide the computing platform to a specific solution. As shown in Fig. 7.7c, although the few adjusted devices are located in the less convenient side of the obstacle, the computation obviously follows the desired path passing across all the specified connections. Unlike in Fig. 7.7a, we observe that now the grid converges much faster to the solution and, most importantly, retains it for much longer before more degenerate paths appear.

7.3.3 *Maze-Solving*

As mentioned before, mazes have been proposed to be solved using memristive networks in the recent literature. In Fig. 7.8 we examine the delineation of a given maze with memristors according to the early approach of [10]. The maze and the possible pathways are shown in Fig. 7.8a. The maze is then superimposed to a memristive network as shown in Fig. 7.8b. This particular underlying network was used in [10] and comprises a head-to-head and tail-to-tail checkerboard pattern of memristors, which are connected to others via a switch at their tail. The crossing points of vertical and horizontal lines define grid points of the memristive network. In order to encode the pattern, the switch is closed for possible pathways and opened for blocked passages. This leads to a network configuration as shown in Fig. 7.8c. Larger mazes are prepared in the same way and voltage is applied across the desired entrance and exit points to solve it.

Since the direction of current flow in the network is not known a priori, the polarity of adjacent memristors was chosen to be alternating. According to [10], such architecture allows modeling different mazes on the same memristive grid without the need to fabricate a specific network for each maze. For the memristive dynamics, however, it was again assumed that the rate of memristance change is proportional to the current flowing through the devices, as shown in Eq. 6.1. Moreover, instead of keeping a steady DC input voltage, the sign of the applied voltage was changed during simulation in order to better represent the maze solution. In fact, if we change the sign of the applied voltage after some time, then the resistance of memristors—along the solution path—that are in the ON state will increase towards the OFF state, whereas the resistance of memristors in the OFF state (which was not affected by the positive voltage due to being reversely

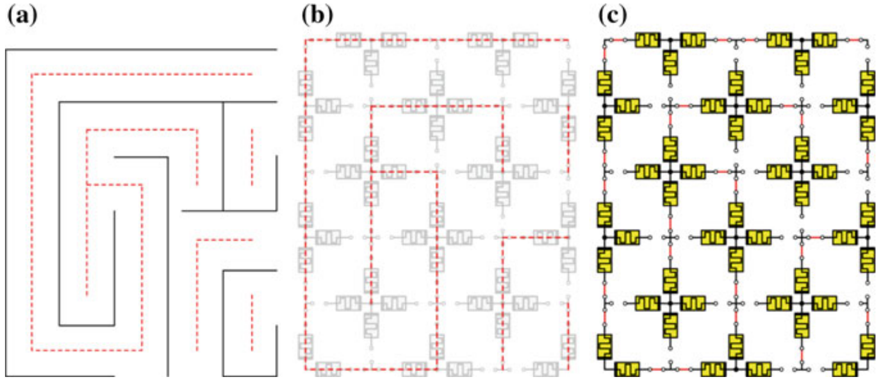


Fig. 7.8 Delineation of maze-solving with memristors. A given two-dimensional maze (**a**) with possible pathways denoted with *red dashed-lines* is first superimposed to a memristive network-switch pattern as shown in (**b**) and the connectivity pattern is stored via the series switches as shown in (**c**). Image redrawn from [10]

polarized) will now decrease. As a result, at a specific moment of time, every memristor along the solution path is in the same intermediate state. Such an approach, however, implies that multi-state reading of memristors is possible.

In our case, by replacing certain memristive components of the fully interconnected mesh grid with R_{MAX} resistors, it was easy to map a particular maze on the same computing structure and experiment with maze-solving. In fact, we further extended the approach of [10] to show how a mesh of memristive components can be utilized for even more sophisticated computations. Entrance and exit nodes of the circuit during simulation are connected to a DC voltage source (or an increasing ramp-waveform voltage) and ground, respectively. Figure 7.9a shows the time evolution of computation when employing identical anti-parallel memristors as computing components for the undirected links. This polarity-independent memristor combination makes unnecessary the change of the sign of the applied voltage since the solution will finally comprise a subset of connections which will have completely switched state towards the less resistive composite state ($R_{\mathrm{ON}}||R_{\mathrm{OFF}}$). The results are in absolute qualitative agreement with those of the previously described approach. The shortest solution emerges first whereas other existing paths, which are longer than the first one, are revealed afterwards as possible alternative options; the steady state of the memristive grid includes all possible ways to the exit of the maze without any distinctions. We note that in maze-solving we have always observed the system to evolve towards a unique (for a given maze) stable solution. Therefore, the success rate of correct solution is 100 %. The same, however, did not happen for the fully interconnected mesh grid and the SPP which was discussed before.

In the next example we worked on the same maze where we selectively modified the switching thresholds of a few memristive components found in the non-optimal solution, which appeared last in Fig. 7.9a. In the simulation result of Fig. 7.9b we notice that the desired solution is the one which now emerges first. Nevertheless, by the end of computation, again all available paths are present with no differentiation between them. After a considerable amount of similar experiments with various types of maze, we figured out that, in order to force the grid to converge faster to a non-optimal solution, the total number of remaining memristive components with high thresholds in the alternate path have to be less in number than those which form the originally shortest solution; i.e. in our case that of Fig. 7.9a. Therefore, the longer the thread of Ariadne is, the higher the chances are for the grid to reach a "desired" solution. Of course, in this assisted-type of network evolution, the same result occurs if we place typical resistors instead of memristors with lower voltage thresholds; only that these resistors will have resistance equal to ($R_{\mathrm{ON}}||R_{\mathrm{OFF}}$) in order to be a priori included in the solution path.

Moreover, it is worth noting that, unlike mentioned in similar studies in the literature, the memristive grid does not require only a single step to find the maze solution. The same applies of course for the SPP; the evolution of computation depends on the switching characteristics of the employed memristive devices, on the network connectivity pattern, and on the applied input voltage. Higher applied voltages might cause all solutions to emerge simultaneously and not gradually as it

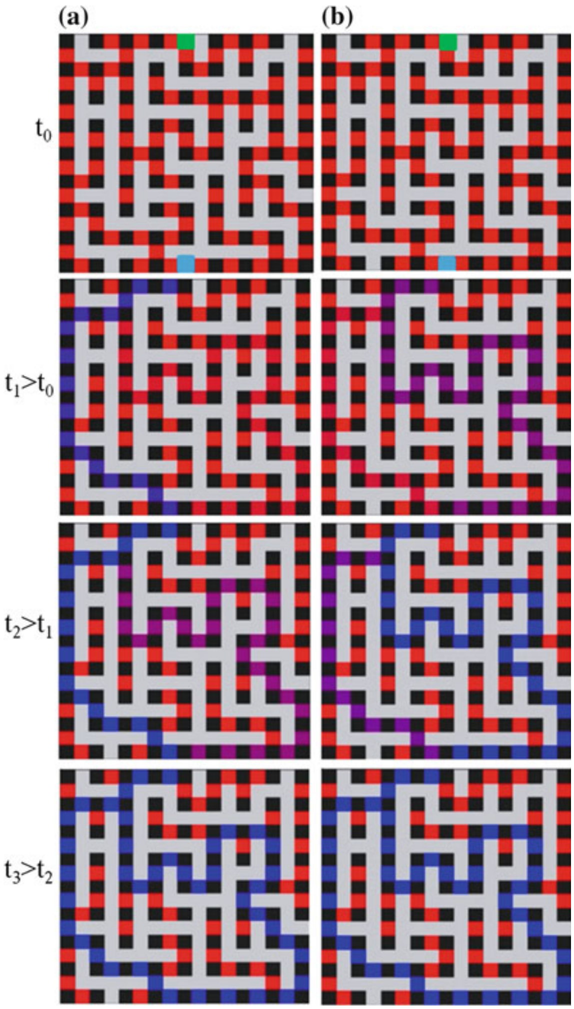


Fig. 7.9 Evolution of maze-solving computations using a memristive grid. In **a** all memristive components involve identical memristive devices, whereas in **b** some devices on the longer path were adjusted to switch at lower thresholds

is desired; the minimum necessary voltage amplitude which gives a first solution might be only roughly estimated. In any case, though, the presented solution approach with memristors is a priori more efficient than any multistep algorithm in present use.

7.3.3.1 Spotting Closed Loops with Composite Memristive Structures

We anticipate that memristive networks will have a wide range of applications beyond the SPP. As we mentioned before, the emergence of unforeseen functionalities could pave the way to exciting new computing concepts. To this end, in Fig. 7.10 we present the simulation results for three different maze-like network

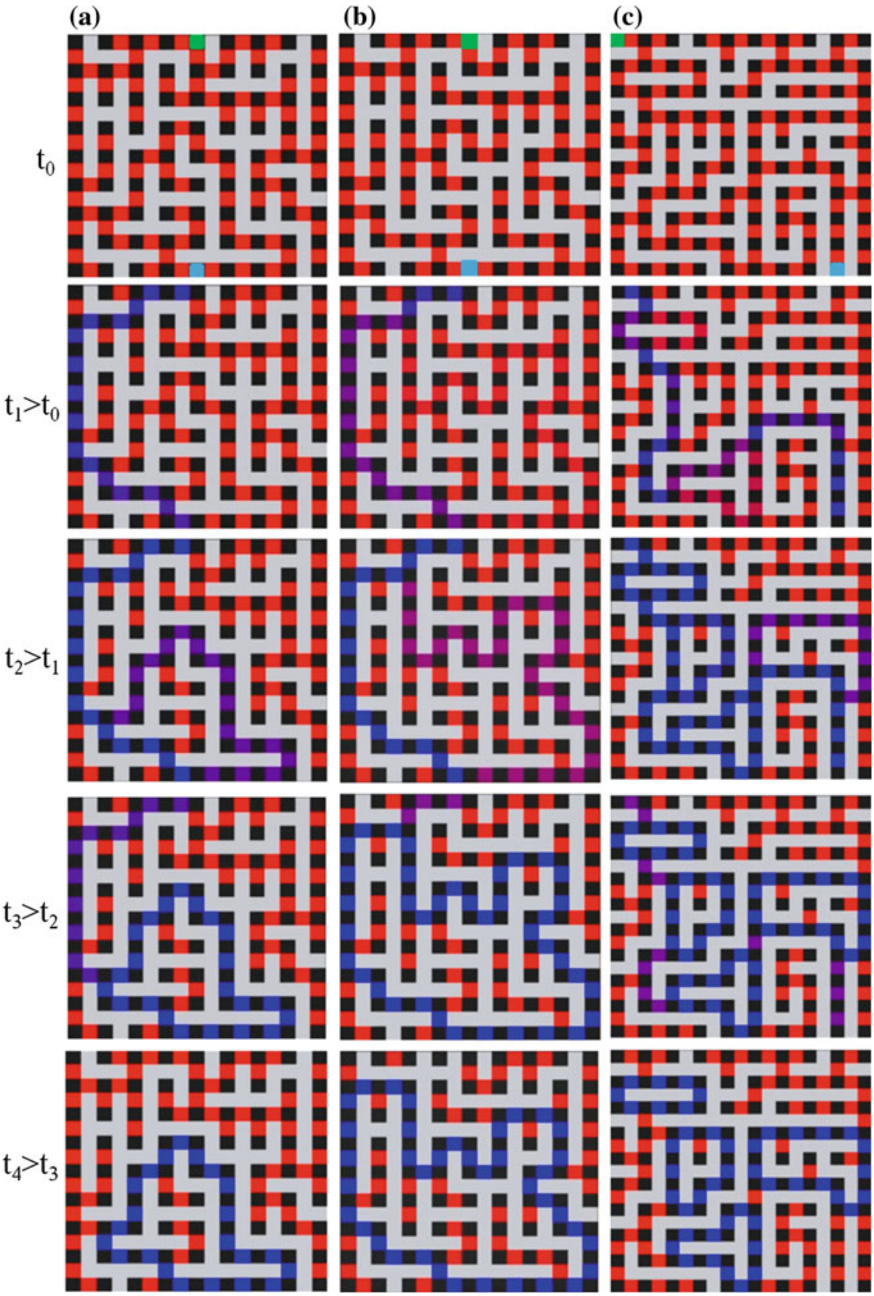


Fig. 7.10 Three different examples **a**–**c** of closed-loop spotting abilities of memristive grids after the introduction of more sophisticated memristive connections between the nodes of a maze-like network

structures. Unlike before, now we used the case (e) of Fig. 7.4 to model network connections. In this composite structure, an additional pair of anti-serial memristors is found in series with the anti-parallel memristors which were used so far. These two additional devices are initially found in the less resistive state (R_{ON}) and their thresholds are selected higher than those of the anti-parallel memristors; hence this new configuration will not affect significantly the previously presented computations since the equivalent resistance of this composition is only slightly higher than before. In fact, the initial resistance is equal to $1/2 \times R_{OFF} + 2 \times R_{ON} \approx 1/2 \times R_{OFF}$, since $R_{OFF} \gg R_{ON}$.

Under the application of a slowly ramped waveform voltage across the indicated nodes, the evolution of the grid will occur as expected, revealing gradually all possible alternate ways from the start to the end points according to their length. However, as the total applied voltage increases, the corresponding voltage drop on the additional series memristors of each component soon exceeds their threshold, thus one of them (depending on the voltage polarity) switches to the high resistive state. As a consequence, the composite memristance rises as well and becomes equal to $(R_{ON}||R_{OFF}) + R_{OFF} + R_{ON} \approx R_{OFF}$. This way of increasing the composite resistance between certain parts (connections) belonging to the initial solution sub-set, provides the ability to eventually exclude these parts from the final solution, thus leaving only the components that have switched to the less resistive composite state.

As particularly shown in Fig. 7.10, computation evolves up to a certain point similarly to when employing only anti-parallel memristors between the grid nodes. Afterwards, as computation goes on and the applied voltage keeps increasing, we demonstrate that this new memristive structure enables the localization of closed loop paths between common points in any undirected graph mapped on a maze-like structure. In all presented cases of Fig. 7.10, the common parts of the solution gradually disappear (i.e. their composite memristance increases) and computation reaches a steady state where only closed loops are shown. This extraordinary behavior of the grid is due to the fact that the equivalent resistance of any closed loop, i.e. of any two parallel circuit branches between two particular nodes, is smaller compared to the resistance of the single common parts of a solution. Therefore the corresponding voltage drop on the single parts results higher and hence exceeds sooner the thresholds of the additional series memristors, thus causing one of them to switch OFF as described before. This novel ability of memristive grids which utilize composite memristive structures, could find many applications in routing and path planning problems, e.g. for the identification of the critical components in specific routes within congested transportation or communication networks, provided that the latter are appropriately mapped on the computing medium.

7.4 Mapping Problems Defined in Directed Graphs

Using the circuit models presented in Fig. 7.4, we show here how a particular directed graph can be efficiently mapped onto the memristive network. Figure 7.11 presents a given directed (oriented) graph where we wish to find the shortest path from the left top vertex to the right bottom vertex. This particular graph is the same with the one in [40] where Adamatzky tested one of the first proposed algorithms for shortest path computations based on CA. CA constitute an inherently parallel computing paradigm, able to capture globally emerging behavior from collective interaction of simple and local components, and have been successfully applied to a range of computational problems including path planning and SPP [41, 42]. CA-based models are also straightforward to implement in conventional HW where the parallelism of the CA structure is well exploited [43]. The sparse nature of computations within a memristive grid resembles, to some extent, certain operational features and computing capabilities of CA, whereas it also leads to scalable and dense HW architectures.

In order to map (project) the graph we use anti-parallel memristors for the bidirectional links and parallel memristor-resistor combinations for the unidirectional links, where memristor polarity follows the edge directivity; this way in the unidirectional links the memristors will switch ON only if they are forward

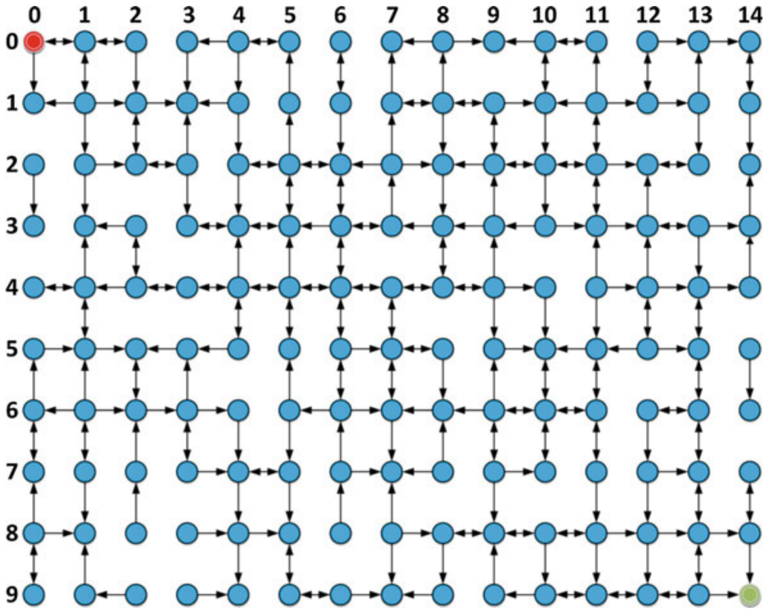


Fig. 7.11 Example of a directed graph taken from [40]. *Arrows* indicate the directivity pattern of the connecting edges. In this example the (0, 0) vertex (*red*) is the source and the (9, 14) vertex (*green*) is the destination

polarized and the applied voltage exceeds their threshold. Finally, R_{MAX} resistors connect adjacent network-nodes where no link exists between the corresponding graph vertices. Due to the arbitrary nature of this scenario, there is no indication about the approximate number of hops between the given nodes. Therefore, we chose to apply a slowly increasing ramp-waveform voltage while we monitored the simulation outcome.

Figure 7.12 presents the solution of the problem for this specific scenario and the visualization of the memristive network state when computation was over. In Fig. 7.12a the black arrows indicate the solution to the problem whereas the blue arrows denote a few alternate equivalent paths between specific nodes; the latter paths appear simultaneously with those of the black arrows. With the switching thresholds of all memristors set at 1 V and the switching time of memristors calculated around 50 ms, the first solution appeared after almost 200 ms when the applied input voltage reached almost 40 V. The presented results are in absolute agreement with those in [40]. However, as the input voltage kept increasing, the alternate path, indicated with orange arrows, appeared as well. We note here that, due to the complex connectivity inside the network, it is not sure that after a certain amount of time (or beyond a minimum input voltage magnitude) all memristors will

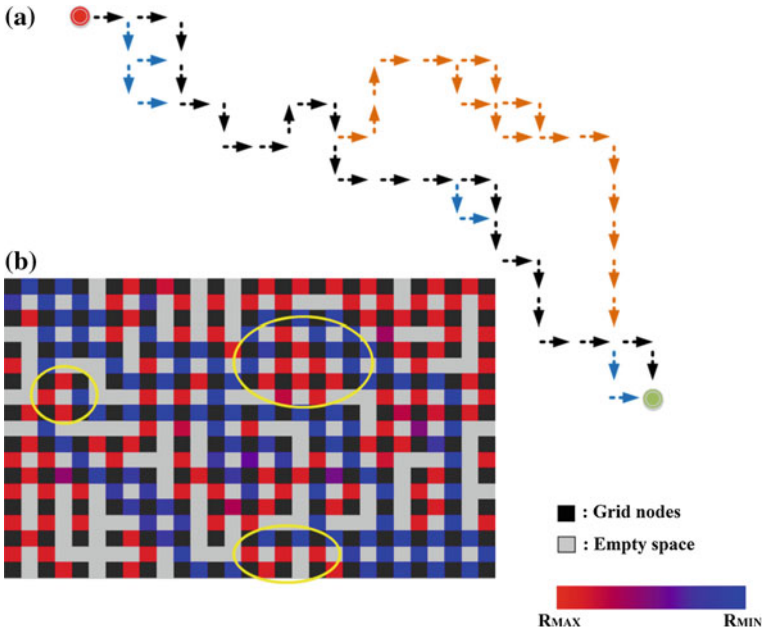


Fig. 7.12 Shortest path computation for the directed graph of Fig. 7.11. **a** Macroscopically shows the solution to the problem where each *arrow* indicates the direction of every subsequent move. **b** Shows the visualization of the memristive network state when computation was complete. The highlighted areas contain memristive connections which correctly have remained in high resistance obeying to the directivity pattern of the graph

switch their states. Focusing on the indicated regions (*yellow* circles) in the visualization snapshot in Fig. 7.12, we see that certain memristors never switch, so some alternate paths are never complete. The network evolution absolutely follows the directed connectivity of the original graph. Of course, before this alternate path appeared, several degenerate paths between specific nodes emerged which, however, did not ruin the initial solution (e.g. see around the source and destination nodes where multiple memristors have switched by the time the snapshot was taken). It is also worth mentioning that, the resulting switching time discussed above is much longer than the typical switching times of experimental nanoscale memristive devices that can be in the sub-ns regime [44]. Therefore, with an appropriate choice of input voltage magnitude, since the switching time of all the memristors along the solution path is on the order of the switching time of a single memristor, we can argue that the minimum time required to find a first solution in a mesh of arbitrary complexity can be in practice as short as few nanoseconds.

Using the same original graph to define the network directivity and connectivity pattern, we show here that the memristive network can be used to solve multiple source-single destination or single source-multiple destinations SPP as well, where computation continues provided that there is at least one connecting path between source and destination pairs. We show this ability in the simulation results of Fig. 7.13. We particularly searched for the shortest connections between source node (0, 2) and destination nodes {(0, 3), (9, 6), (3, 10)}. In spite of the complexity of the routes, the network gradually converged to the correct solutions as shown in Fig. 7.13a. The first path to appear was the path towards destination (0, 3). Afterwards, the last two paths leading to the rest of the destinations were completed almost simultaneously. The connections indicated with red arrows in the macroscopic view of the routes in Fig. 7.13b, were the last to emerge after 300 ms and when the applied voltage reached 60 V, even though the rest of the connections belonging to the solution-paths had already emerged. In the same fashion, one can compute shortest paths originating from multiple sources towards one destination just by placing DC sources and grounds at the corresponding nodes.

7.5 Overview and Discussion

This chapter focused on the exploration of complex memristive networks where emergent computation arises through collective and simultaneous device interactions. The main advantage of this approach is based on the analog parallel dynamics of many interconnected memristive devices. Such highly controllable architectures are not complex networks with respect to either physical interconnectivity or dynamical properties. They serve, though, to emulate or simulate complex networks by implementing designer algorithms. Numerical modeling of memristive networks is easily implemented and, thus, offers by itself a practical computational algorithm for several known problems whose solution is hard to be computed fast. Here both high-level as well as circuit-level simulations via SPICE revealed the pros and cons

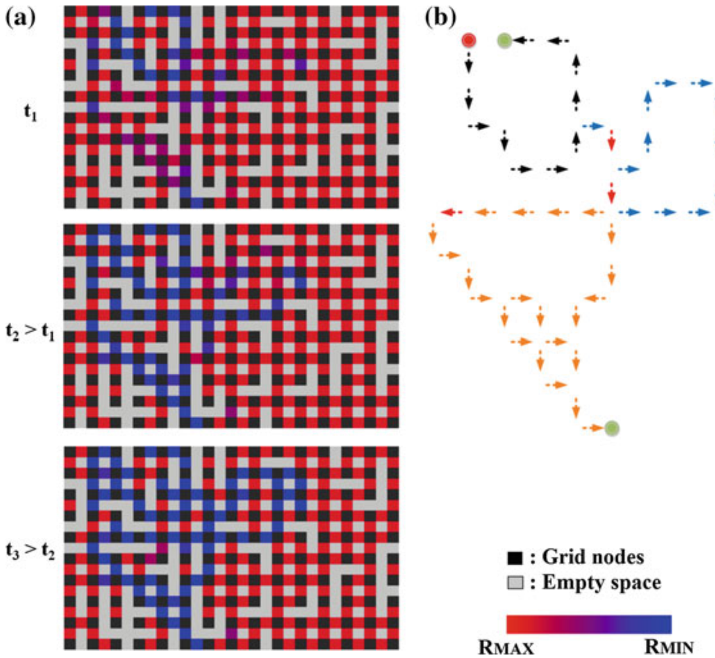


Fig. 7.13 Single source-multiple destinations shortest path computation for the directed graph of Fig. 7.11 between source node (0, 2) and destination nodes {(0, 3), (9, 6), (3, 10)}. **a** Visualization of the time-evolution of the memristive network state. **b** Shows macroscopically the solution to the problem containing all of the formed pathways. The connections indicated with *red arrows* were the last to appear

of such unconventional computing approach for shortest-path and maze-solving applications. Important computing inefficiencies were attributed to the dependence of the computing medium behavior on the symmetry of both the underlying geometry and the employed devices. We reported certain limitations of regular networks that in some cases provide degenerate solutions. Random and quasi-periodic networks without any symmetry are expected to be more promising in this regard. However, extraordinary functionalities emerged when novel memristive computing components, comprising different electrical characteristics from their structural elements, were introduced in the undirected mesh grid. Applying assisted-computation, by incorporating the concept of *Ariadne's thread*, leaded to better computing results which could find application in routing and path planning problems.

Experimentally, the suggested networks could be fabricated, e.g. by combining one (or more) memristor layer(s) (or memristor emulators [45] in small-scale versions of networks) with an additional complementary metal-oxide-semiconductor (CMOS) layer. The physical production of such complex circuit architectures would be a great achievement because such systems would provide adaptability and computing capabilities at a rate determined by device physics and not by algorithmic complexity! It is therefore highly desirable to probe their unique characteristics in hopes of harnessing the dynamical emergent properties in HW implementations as well. In this context, unlike mentioned in similar studies in the literature [10, 11], the memristive grid does not require only a single step to find the solution. In fact, the evolution of computation depends on the switching characteristics of the employed memristive devices and on the underlying network which, in turn, imposes different requirements for the applied input voltage. Higher applied voltages might cause all solutions to emerge simultaneously and not gradually as it is desired; the minimum necessary voltage amplitude which gives a first solution might be only roughly estimated. Nevertheless, the computation time is independent of the network size and/or complexity.

With respect to path planning problems, these are typically evaluated using four metrics: (i) time complexity (the time needed to find a solution); (ii) space complexity (memory needed for the search); (iii) completeness (a solution is found if it exists); and (iv) optimality (the best solution is found) [46]. Regarding (i), for systems that can search multiple grid nodes in parallel, such as in [27], if we assume equal connection weights then the propagation of the signal is uniform in the grid. This is in-line with the presented parallel solution approach with memristors, which is a priori more time-efficient than any multistep algorithm in present use. Compared to initial neuron training of a neuron IC, which could be time-consuming, the memristive platform only requires a common initialization of all involved devices. Of course, the state read-out time cannot be avoided but we believe it is comparable with measurements of current neuromorphic solutions like in [27] and certainly here is compensated by the minimal computation time. Concerning (ii), in our case it is related to both the network size and the node connection type; the more adaptive and robust the platform is made, the larger its size complexity. Finally, requirements (iii) and (iv) are absolutely covered by memristive networks.

Overall, the most important parameters about memristive grid-based computations concern: (i) the size of the grid, which should be large enough to accommodate the target application graph; (ii) the switching time and thresholds of the memristors, which should be known to facilitate the selection of the applied input voltage; (iii) the type of memristors, which should demonstrate threshold-based switching; and (iv) the connection type of the target application graph, which should correspond to the available node-connecting components of the grid.

References

1. E.A. Vittoz, Future of analog in the VLSI environment, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, New Orleans, LA, USA (1990)
2. E. Linn, R. Rosezin, S. Tappertzhofen, U. Bottger, R. Waser, Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations, Nanotechnology **23**, 305205 (2012)

3. D. Stathis, I. Vourkas, G.C. Sirakoulis, Solving AI problems with memristors: a case study for optimal “bin packing”, in *18th Panhellenic Conference on Informatics (PCI)*, Athens, Greece (2014)
4. D.B. Strukov, G.S. Snider, D.R. Stewart, R.S. Williams, The missing memristor found. Nature **453**, 80–83 (2008)
5. K.H. Kim, S. Gaba, D. Wheeler, J.M. Cruz-Albrecht, T. Hussain, N. Srinivasa, W. Lu, A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. Nano Lett. **12**(1), 389–395 (2012)
6. H. Kim, M.P. Sah, C. Yang, T. Roska, L.O. Chua, Neural synaptic weighting with a pulse-based memristor circuit. IEEE Trans. Circ. Syst. I Reg. Papers **59**(1), 148–158 (2012)
7. M. Di Ventra, Y.V. Pershin, The parallel approach. Nat. Phys. **9**, 200–202 (2013)
8. M. Di Ventra, Y.V. Pershin, L.O. Chua, Circuit elements with memory: memristors, memcapacitors and meminductors. IEEE Proc. **97**(10), 1717–1724 (2009)
9. J.J. Yang, D.B. Strukov, D.R. Stewart, Memristive devices for computing. Nat. Nano. **8**, 13–24 (2013)
10. Y.V. Pershin, M. Di Ventra, Solving mazes with memristors: a massively parallel approach. Phys. Rev. E **84**, 046703 (2011)
11. Y.V. Pershin, M. Di Ventra, Self-organization and solution of shortest-path optimization problems with memristive networks. Phys. Rev. E **88**, 013305 (2013)
12. Y. Pershin, V. Slipko, M. Di Ventra, Complex dynamics and scale invariance of one-dimensional memristive networks, Phys. Rev. E **87**, 022116 (2013)
13. I. Vourkas, G.C. Sirakoulis, Study of memristive elements networks. J. Nano Res. **27**, 5–14 (2014)
14. I. Vourkas, G.C. Sirakoulis, A novel design and modeling paradigm for memristor-based crossbar circuits. IEEE Trans. Nanotechnol. **11**(6), 1151–1159 (2012)
15. J. Borghetti, G.S. Snider, P.J. Kuekes, J.J. Yang, D.R. Stewart, R.S. Williams, Memristive switches enable ‘stateful’ logic operations via material implication. Nature **464**(7290), 873–876 (2010)
16. E. Lehtonen, J.H. Poikonen, M. Laiho, Implication logic synthesis methods for memristors, in *IEEE Int. Symp. Circuits Syst.(ISCAS)*, Seoul, South Korea (2012)
17. S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U.C. Weiser, E.G. Friedman, MRL—memristor ratioed logic, in *13th International Workshop on Cellular Nanoscale Networks and their Applicarions (CNNA)*, Turin, Italy (2012)
18. G. Papandroulidakis, I. Vourkas, N. Vasileiadis, G.C. Sirakoulis, Boolean logic operations and computing circuits based on memristors. IEEE Trans. Circuits Syst. II Expr. Briefs **61**(12), 972–976 (2014)
19. S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E.G. Friedman, A. Kolodny, U.C. Weiser, MAGIC—Memristor Aided LoGIC. IEEE Trans. Circuits Syst. II Expr. Briefs **61**(11), 895–899 (2014)
20. E. Lehtonen, M. Laiho, CNN using memristors for neighborhood connections, in *12th Internationl Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*, Berkeley, CA (2010)
21. D. Stathis, I. Vourkas, G.C. Sirakoulis, Shortest Path Computing using Memristor-based Circuits and Cellular Automata, in *11th International Conference on Cellular Automata for Research and Industry (ACRI)*, Krakow, Poland (2014)
22. Y.V. Pershin, M. Di Ventra, Neuromorphic, digital and quantum computation with memory circuit elements. Proc. IEEE **100**(6), 2071–2080 (2012)
23. W. Zhao, D. Querlioz, J.O. Klein, D. Chabi, C. Chappert, Nanodevice-based novel computing paradigms and the neuromorphic approach, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seoul, South Korea (2012)
24. E. Neftci, J. Binas, U. Rutishauser, E. Chicca, G. Indiveri, R.J. Douglas, Synthesizing cognition in neuromorphic electronic systems, Proc. Nat. Acad. Sci. (PNAS), **110**(37), E3468–E3476 (2013)

25. B.V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A.R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J.V. Arthur, P.A. Merolla, K. Boahen, Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. IEEE Proc. **102**(5), 699–716 (2014)
26. A. Basu, S. Ramakrishnan, C. Petre, S. Koziol, S. Brink, P.E. Hasler, Neural dynamics in reconfigurable silicon. IEEE Trans. Biomed. Circ. Syst. **4**(5), 311–319 (2010)
27. S. Koziol, S. Brink, J. Hasler, A neuromorphic approach to path planning using a reconfigurable neuron array IC. IEEE Trans. VLSI Syst. **22**(12), 2724–2737 (2014)
28. S. Brink, S. Nease, P. Hasler, S. Ramakrishnan, R. Wunderlich, A. Basu, B. Degnan, A learning-enabled neuron array IC based upon transistor channel models of biological phenomena. IEEE Trans. Biomed. Circ. Syst. **7**(1), 71–81 (2013)
29. C.K.K. Lim, T. Prodromakis, computing motion with 3D memristive grid. arXiv:1303.3067
30. F. Jiang and B.E. Shi, The memristive grid outperforms the resistive grid for edge preserving smoothing, in *European Conference* on *Circuit Theory and Design* (*ECCTD*), Antalya, Turkey (2009)
31. C. Nakagaki, H. Yamada, A. Toth, Maze-solving by an amoeboid organism, *Nature* **407**(470), 6803 (2000)
32. I. Vourkas, D. Stathis, G.C. Sirakoulis, Massively parallel analog computing: Ariadne's thread was made of memristors. IEEE Trans. Emerg. Top. Comput. (2015, in press). doi: 10.1109/TETC.2015.2420353
33. G. Ligang, F. Alibart, D.B. Strukov, Programmable CMOS/memristor threshold logic. IEEE Trans. Nanotechnol. **12**(2), 115–119 (2013)
34. I. Vourkas, G.C. Sirakoulis, On the generalization of composite memristive network structures for computational analog/digital circuits and systems. Microelectron. J. **45**(11), 1380–1391 (2014)
35. I. Vourkas, G.C. Sirakoulis, On the analog computational characteristics of memristive networks, in *20th IEEE International Conference on Electronics, Circuits and Systems (ICECS),* Abu Dhabi (2013)
36. Easy Java Simulations (EJS). Available: http://fem.um.es/Ejs/. Accessed 2014
37. I. Vourkas, D. Stathis, G.C. Sirakoulis, XbarSim: an educational simulation tool for memristive crossbar-based circuits, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Lisbon, Portugal (2015)
38. I. Vourkas, A. Batsos, G.C. Sirakoulis, SPICE modeling of nonlinear memristive behavior. Int. J. Circ. Theor. Appl. **43**, 553–565 (2015)
39. M.D. Pickett, D.B. Strukov, J.L. Borghetti, J.J. Yang, G.S. Snider, D.R. Stewart, R.S. Williams, Switching dynamics in titanium dioxide memristive devices. J. Appl. Phys. **106**, 074508 (2009)
40. A.I. Adamatzky, Computation of shortest path in cellular automata. Math. Comput. Model. **23** (4), 105–113 (1996)
41. K. Ioannidis, G.C. Sirakoulis, I. Andreadis, A path planning method based on cellular automata for cooperative robots. Appl. Artif. Intell. **25**(8), 721–745 (2011)
42. S. Golzari, M.R. Meybodi, A maze routing algorithm based on two dimensional cellular automata, in *7th International Conference* on *Cellular Automata for Research and Industry (ACRI)*, Perpignan, France (2006)
43. I. Vourkas, G.C. Sirakoulis, FPGA based cellular automata for environmental modeling, in *19th IEEE International Conference* on *Electronics*, *Circuits, and Systems (ICECS)*, Seville, Spain (2012)
44. A.C. Torrezan, J.P. Strachan, G. Medeiros-Ribeiro, R.S. Williams, Sub-nanosecond switching of a tantalum oxide memristor. Nanotechnology **22**(48), 485203 (2011)
45. C. Sánchez-López, J. Mendoza-López, M.A. Carrasco-Aguilar, A floating analog memristor emulator circuit. IEEE Trans. Circuits Syst. II Expr. Briefs **61**(5), 309–313 (2014)
46. S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach* (Prentice-Hall, Englewood Cliffs, NJ, 2003)

Chapter 8
Memristive Computing for NP-Hard AI Problems

8.1 Introduction

The memristor has definitely shown abilities that could revolutionize computing in the coming decades [1–4]. Its unique adaptive properties are ideal for computational purposes and, so far, they have motivated the exploration of novel computing paradigms [5–7]. The pinched current-voltage hysteresis feature indicates the potential of using it in a continuous operational mode as part of an analog computational paradigm [8–11]. For example, reported properties of network configurations of memristors, as presented in Chap. 7, showed that composite memristive systems significantly improve the efficiency of logic operations via massive analog parallelism, where calculation consists in the evolution of the memristance of all the involved devices. Massive parallelism is commonly found in nature, hence massively parallel computing systems and architectures constitute a great technological engineering challenge [12–14].

However, in Chap. 7 we extensively discussed the major disadvantages and the computing inefficiencies of memristive networks which are mostly attributed to the dependence of the computing medium behavior on the symmetry of both the underlying circuit geometry and the employed devices. Such inefficiencies were properly treated with the inclusion of asymmetries in the computing structure through a variety of composite memristive components [15–17]. Nevertheless, this necessary range of electronic components available so that the requirements of different application are met, reduces significantly the overall structural homogeneity and simplicity of the hardware (HW). Moreover, among the mentioned drawbacks we distinguish: (i) the need to access all the memristors sequentially, both for their initialization and for the read-out of the network state; (ii) the power consumption (the minimum necessary applied input voltage normally varies). It was mentioned, though, that the sparse nature of such memristor network-based computations resembles certain operational features and computing capabilities of

I. Vourkas and G.Ch. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Emergence, Complexity and Computation 19,
DOI 10.1007/978-3-319-22647-7_8

Cellular Automata (CA), a powerful parallel computational model which leads to scalable HW architectures with very high device densities.

CA constitute a well-studied inherently parallel computing paradigm of high efficiency and robustness [18, 19]. Owing to their potential to capture globally emerging behavior from collective interaction of simple and local components, CA have found successful application in several computational problems in physics, chemistry, biology, geology, and computer science, to name a few [20–28]. Additionally, when CA-based models are implemented in HW, the circuit design reduces to the design of a single cell and the overall layout results regular with exclusively local interconnections [14, 29, 30]. Moreover, the models are executed fast by exploiting the parallelism of the CA structure, thus meeting the necessary information processing requirements in modern computationally demanding applications. The CA approach is consistent with the modern notion of unified space-time; memory (cell state) and processing unit (local rule) are inseparably related to a CA cell. Therefore, it is very justifiable to search for computing paradigms which combine the capabilities and the structural simplicity of CA with the unique properties of memristors, which show promise to be used for in-memory information processing.

This chapter focuses on CA-based algorithm implementations using circuits which comprise memristors, composite memristive devices, and conventional electronic components. It builds upon our previous work in memristive networks in an attempt to address the majority of their computing inefficiencies and permit the full exploitation of their massively parallel computing power for the solution of a set of classic NP-hard artificial intelligence (AI) problems. To this end, we propose a CA-inspired circuit-level approach, capable of executing computation within memory (memristors). We particularly develop a circuit design methodology for the implementation of memristive CA cells which will implement (almost) any CA evolution rule. We prove the fine application of the proposed methodology to a set of target AI computational problems through system-level simulations. In each case we describe the fundamental memristive CA cell and then employ it to create sophisticated array-like circuit structures, able to execute the CA-based algorithms. The main contribution of this methodology consists in the combination of a powerful computational tool with the unique circuit properties of one of the latest technological breakthroughs in electronics, which could further improve CA-based massively parallel HW accelerators for NP-hard AI problems.

8.2 Basics of Cellular Automata and Suitable HW Structures for Their Implementation

Cellular Automata (CA), originally postulated in the 1940s by Ulam and von Neumann [31], are computational models of physical systems where space and time are discrete and interactions are only local. A Cellular Automaton consists of a regular and uniform *d*-dimensional lattice (or array). At each site of the lattice (cell) a physical quantity takes values. This physical quantity is the global state of the CA, and the value of this quantity at each cell is the local state of this cell. Each cell is normally restricted to local neighborhood interaction only, and, as a result, it is incapable of immediate global communication. The neighborhood of a cell is generally taken to be the cell itself and some (or all) of the immediately adjacent cells. The states at each cell are updated simultaneously at discrete time steps, based on the states in their neighborhood at the preceding time step. The algorithm which is used to compute the next cell state is referred to as the CA local evolution rule. Below we present a formal definition [18] according to which, a CA generally requires:

1. A regular lattice of cells covering a portion of a d–dimensional space;
2. A set of variables $\mathbf{C}(\vec{r},t) = \{C_1(\vec{r},t), C_2(\vec{r},t), \ldots, C_{\mathrm{m}}(\vec{r},t)\}$ attached to each site $\vec{r}$ of the lattice, giving the local state of each cell at the specific time value t;
3. A rule $R = \{R_1, R_2, \ldots, R_m\}$ which specifies the time evolution of the states $\boldsymbol{C}(\vec{r},t)$ in the following way: $C_j(\vec{r},t+1) = R_j\left(\boldsymbol{C}(\vec{r},t),\ldots,\boldsymbol{C}\left(\vec{r}+\vec{\delta}_k,t\right)\right)$, where $\vec{r}+\vec{\delta}_k$ designate the cells which belong to a given neighborhood of cell $\vec{r}$.

It is important to notice that, in the above definition, the evolution rule R is homogeneous; i.e. it is identical for all sites and it is applied simultaneously to each of them, leading to synchronous dynamics. However, spatial (or even temporal) inhomogeneities can be introduced by having a subset of cells (in some given locations of the lattice) systematically set at a fixed value, in order to mark particular cells to which a different rule is applied. Furthermore, the new state of a particular cell $\vec{r}$ at time $t + 1$ is only a function of the previous state of the specific cell and of the cells which belong to its designated neighborhood. The neighborhood of cell $\vec{r}$ is the spatial region in which it needs to search in its vicinity. In one-dimensional (1-d) elementary CA, the typical neighborhood consists of the central cell (which is to be updated) and the two adjacent cells on both sides. The neighborhood size n is usually taken to be $n = 2r + 1$, where r is a positive integer parameter known as the radius (i.e. the number of cells on each side that are involved). On the other hand, for two-dimensional (2-d) CA, two types of neighborhood are usually considered: (i) the von Neumann neighborhood, which consists of a central cell (the one which is to be updated) and its four geographical neighbors {north, west, south and east}; and (ii) the Moore neighborhood which contains, in addition, second nearest neighbors {northeast, northwest, southeast and southwest}; i.e. a total of nine cells, whereas the von Neumann neighborhood comprises only five cells. All mentioned types of CA neighborhoods are schematically shown in Fig. 8.1.

CA are decentralized, spatially extended systems consisting of simple and identical components with absolutely local connectivity. In spite of the apparent simplicity of their structure, they have sufficient expressive dynamics to represent phenomena of arbitrary complexity. They have the ability to perform complex computations with a high degree of efficiency and robustness, as well as to model the behavior of complex systems found in nature [24, 25, 28, 32]. Furthermore, they can easily handle complicated boundary and initial conditions, as well as anisotropies. Most importantly, though, their implicit spatial locality allows for very

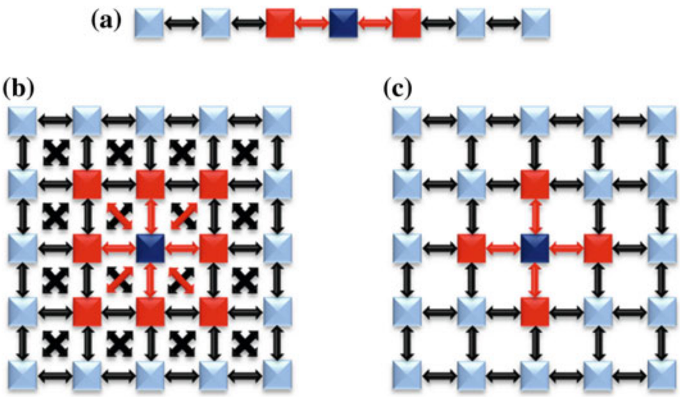


Fig. 8.1 Cellular Automata (CA) neighborhood representations. **a** A typical neighborhood for one-dimensional CA of radius $r = 1$. **b** Moore neighborhood and **c** von Neumann neighborhood for two-dimensional CA. Central cells are marked with *dark blue* whereas the rest of the cells in each neighborhood are marked with *red*

efficient high performance implementations in hardware (HW). CA can be simulated exactly by digital electronic computing systems because of their intrinsic discreteness, i.e. the topology of the simulated object is reproduced in the simulating device [14, 29]. Moreover, the CA approach is consistent with the modern notion of unified space–time. In computer science, space corresponds to memory and time to processing units. In CA, memory (CA cell state) and processing unit (CA local rule) are inseparably related to the CA cell.

Modern high performance HW platforms include logic density equivalent to millions of logic gates per chip and can implement very complex computations [1]. CA consists of a uniform structure composed of many identical synchronous cells where both memory and computation are involved. Hence, it matches the inherent design layout of state-of-the-art parallel electronic systems, such as a Field Programmable Gate Array (FPGA). Figure 8.2 demonstrates the aforementioned structural similarities and summarizes how memory and processing unit are closely related, both in CA cells and in sequential logic circuit blocks; e.g., in the configurable logic blocks (CLBs) of FPGAs. The structure of a cell consists of a combinational part connected with one or more memory elements in a feedback loop shape. The state of the memory elements is defined by the inputs and the present state of these elements. Specifically, FPGAs appear to be very attractive for CA-based algorithms; a CA-based circuit design reduces to the design of a single, relatively simple cell, and the total layout is uniform. Therefore, CA-based models are straightforward to implement in parallel HW platforms where they are executed very fast by taking advantage of the inherent parallelism of the CA structure. Consequently, up to now many such CA-based parallel electronic systems have been proposed and implemented as a way to speed-up execution of computationally intensive applications, especially by using the potential of FPGAs [14, 23, 28–30].

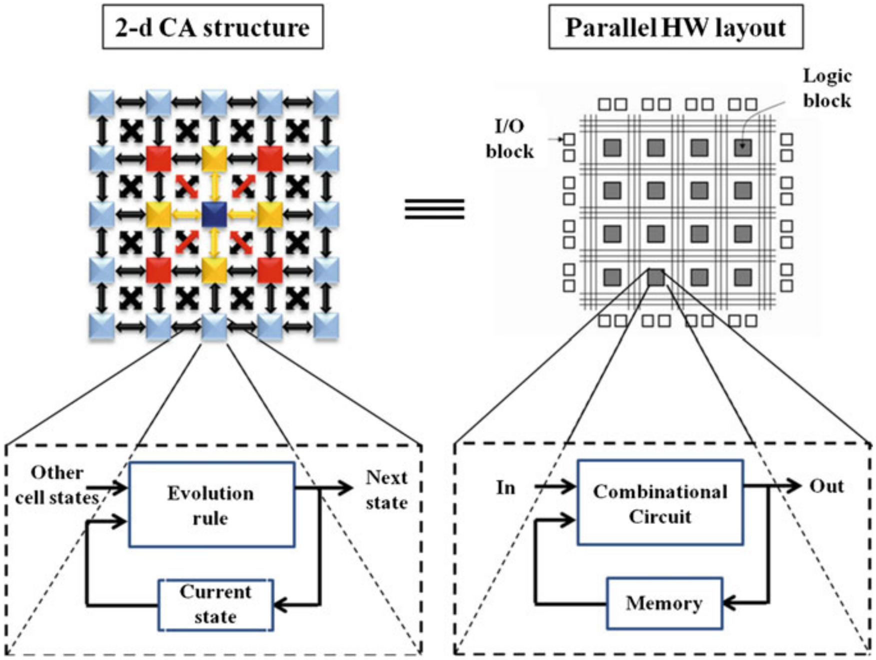


Fig. 8.2 A 2-d CA lattice compared to a common design layout of parallel hardware (HW), e.g. that of a field programmable gate array (FPGA). The internal structure of the CA cells and of the sequential logic HW blocks is schematically shown, highlighting the inseparable relation of memory and processing unit to the CA cell

8.3 Application Mapping Methodology to Memristive CA-Based Circuits

Having sufficient and powerful available hardware (HW) resources sounds exciting when it comes to the HW-based acceleration of processes and applications. However, mapping Cellular Automata (CA)-based models and algorithms from an abstract concept to the given HW is a major challenge. In this section, inspired by the work of Itoh and Chua [33] who first discussed simulation of CA in networks of memristors, we aim to exploit the threshold-based resistance switching behavior of memristors and of their multi-state composite components to propose a novel circuit-level approach to the design of memristive HW CA structures. We particularly describe the basic guidelines which enable the circuit implementation of memristive CA cells using memristors and memristive configurations as storage and/or computing elements. The memristive CA cells implement the desired CA evolution rule and are employed to create one- or two-dimensional, structurally-dynamic (also called topological) CA arrays, where the CA-based algorithms are executed.

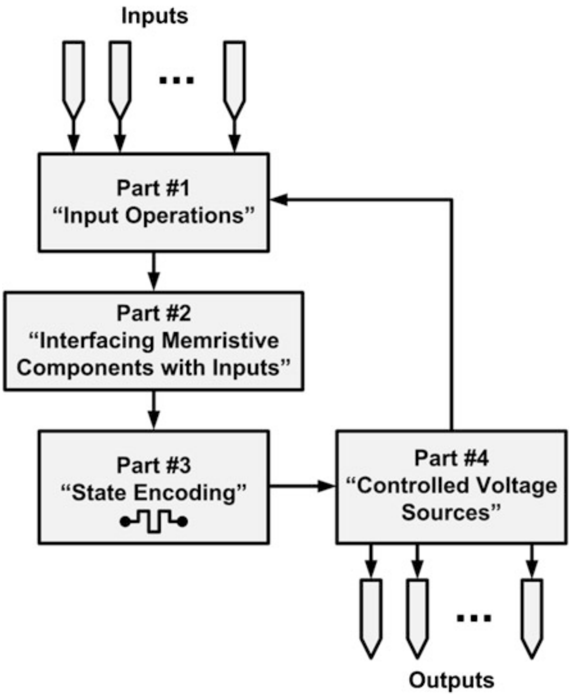


Fig. 8.3 Block diagram of the general layout for all memristive CA cell circuit implementations

As explained before, in CA-based structures there is only local communication between the CA cells which belong to a defined neighborhood. Each cell computes its next state while taking into account its own previous state and that of its closest neighbors. Depending on the number and the type of the possible cell states, each cell includes a number of memristors which serve both for encoding/storing the current state, as well as for computing the next value. The resistive cell-state is then translated properly to a voltage value, which is used internally in computations and is also driven to the output of the cell. In general, the circuit implementation of any such CA cell has a fundamental layout which is composed of four parts, as shown schematically in Fig. 8.3, which are described in more detail below:

1. “Input Operations”; in this part, depending on the CA evolution rule, a set of logical and/or typical arithmetic operators +, −, * and / are used with the internal cell-state signal and the external input voltage signals, which are applied to the cell.
2. “Interfacing Memristive Components with Inputs”; this part controls the programming voltage pulses which are applied to the memristive components of the cell. It involves a set of control switches where the control signals can be the external inputs, the internal cell state signal, or a signal which resulted by any logic and/or mathematical operation in the previous part.

3. “State Encoding”; this part contains a number of memristors which are properly interconnected so as to serve the computation and the encoding/storage of the cell state, according to the applied voltage pulses which are defined in the previous parts. It also includes the necessary circuitry to translate the stored resistive states to proper voltage signals.
4. “Controlled Voltage Sources”; in this part there is a set of voltage- or current-controlled DC voltage sources, whereas there may also be a number of control switches. The voltage sources hold the output signal which is communicated to the neighboring cells and is used internally in computations as well. The voltage amplitude of the sources and the state of the control switches (if any) in this circuit part are both defined by the resistive state of the memristive components in the “State Encoding” circuit part.

Depending on the actual algorithm to be executed and on the evolution rule, any of the aforementioned parts can be modified accordingly. As we will see in the following sections, for example, the “State Encoding” part can include single memristors, composite multi-state memristive switches, or complex network configurations of memristors. Similarly, sometimes there might be no need for any processing of the input signals which could be instead applied directly to the memristive components. However, as the complexity of the simulated system increases, the number of different resistive states, necessary to represent the values of the physical quantities at each site of the CA lattice, may become quite large, hence calling for the use of more memristors or sophisticated memristive ensembles.

According to Fig. 8.2, the structure of each CA cell is generally separated in two parts: (i) the combinational one, which mainly includes all computations taking place in the cells, and (ii) the memory part, which passes the combinational results to adjacent cells during the next time step. Taking this into account, the memristive CA cell circuit implementations operate in two stages, namely the “Computation Stage” and the “Read Stage”. During the “Computation Stage” the first three parts of the cell layout are activated, so the new cell state is computed and encoded in the impedance of the memristive components. This resistive state-programming is achieved with the application of voltage pulses of appropriate amplitude, as stated previously. It is important to note that the memristive components might need to undergo a reset step which will program them to a predefined resistive state beforehand; it is also possible to selectively split the “Computation Stage” in two separate sub-stages, namely “Set” and “Reset”, whose names denote the exact programming operation taking place over the memristive components. Finally, during the “Read Stage”, parts 3 and 4 of the circuit layout interact so that the resistive state of the memristive components is properly decoded to a voltage value, which is stored and used internally in next state computations and also is communicated to the adjacent CA cells.

By following this general methodology, including probably some necessary modifications made to any of the four parts of the common cell circuit layout, the reader will be able to implement (almost) any cell of a CA-based computing model in modern HW using memristors as the basic memory and/or processing elements. This way, a powerful computational tool is successfully combined with the unique circuit properties of memristors in a computational scheme capable of executing computations within memory. We denote here that all the assumptions regarding both switching thresholds and programming voltage values, as well as for the memristance range of the memristors, have been made only in the context of this study; thus, they do not relate to any real, manufactured or measured devices.

8.4 Solving NP-Hard Artificial Intelligence Problems

Based on the above guidelines, in the following sections we present several sophisticated CA-inspired circuit structures and prove their ability to efficiently solve a set of well-known NP-hard artificial intelligence (AI) problems. The provided system-level simulation-based validation is based on the memristor device model of Chap. 2 [34] and on proper software tools developed via the Easy Java Simulations (EJS) environment [35].

8.4.1 Shortest Path and Traveling Salesman Problems

Solution of the shortest path problem (SPP) has always been a hot topic in graph theory because of its wide application field. There are three well-known alterations of this problem: (i) the single source shortest path; i.e. the shortest path from a given vertex of a graph to all others, (ii) the all pairs shortest path; i.e. the shortest path between all possible pairs of the vertices, and (iii) the single source single destination shortest path. In this section we describe a fundamental memristive cell which can implement the desired CA local evolution rule and then employ it to create a two-dimensional (2-d) structurally-dynamic (also called topological) CA, able to compute the shortest path between given nodes of a mesh with weighted edges. The main contributions of this approach are: (i) a memristor-based CA capable of detecting the nodes of a given mesh belonging to the shortest path from one source-node of the mesh to one or multiple destination-nodes; (ii) the iterative application of the proposed memristive CA for the solution of the traveling salesman problem in undirected graphs [36].

8.4.1.1 Circuit-Level Implementation

The basic memristive CA cell is schematically shown in Fig. 8.4. As we did in previous chapters, hereinafter we will again refer to forward (reversely) polarized

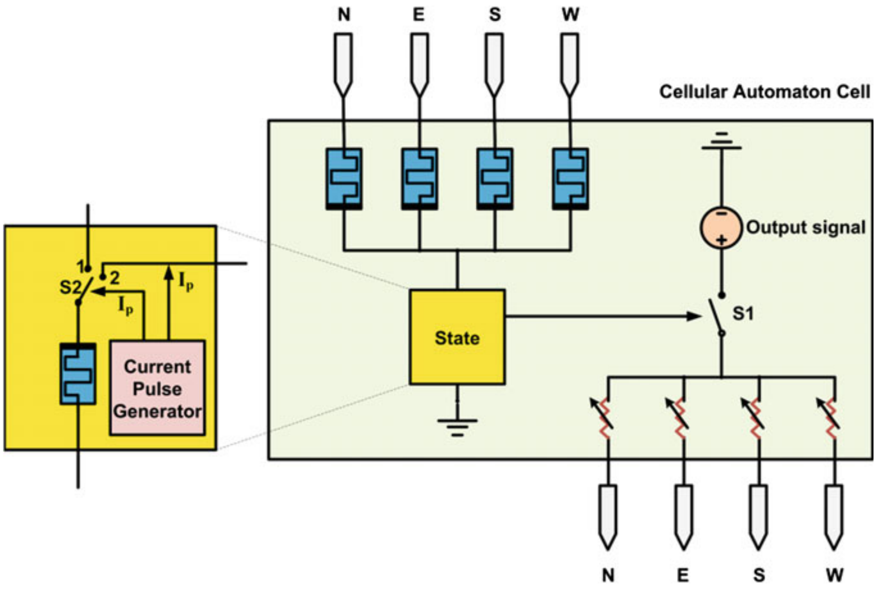


Fig. 8.4 A memristive CA cell with four inputs and one output connecting with the nearest cells belonging to the von Neumann neighborhood. *N*, *E*, *S*, and *W* denote the four geographical neighbors, namely north, east, south and west

memristors as FPMs (RPMs). Regarding the memristor circuit schematic, we remind here that for a FPM the top terminal is the one with the thin line whereas for a RPM it is the one with the thick line. Hence, the memristance of a FPM will decrease (increase) when it is forward (reversely) biased, whereas a RPM exhibits the opposite behavior.

The cell includes five memristors: a RPM holding the state of the cell (state-memristor), and four FPMs which are driven by four inputs corresponding to the connections of a von Neuman neighborhood (input memristors). With respect to the cell design methodology, apparently here there is no pre-processing of the input signals and/or the current state signal. The inputs are applied directly to the memristive part of the cell. Similarly, there is no controlled-source to hold the cell state. Instead, there is a simple voltage source with a fixed value which is communicated to the output of the cell when the conditions are met. The input memristors are identical and have maximum memristance $R_{\mathrm{OFF}} \approx 20\ \mathrm{k}\Omega$, whereas $R_{\mathrm{OFF}} \approx 1\mathrm{M}\Omega$ is used for the state-memristor. These anti-serially connected memristors form a voltage divider circuit and the aforementioned memristance boundary values were chosen so that the state-memristor, after switching to the high resistive state, it would effectively prevent any state-change of the input memristors which are still in the high-resistive (OFF) state. Therefore, we have a complementary resistive switch (CRS), only that all the FPMs here are connected to a common RPM. Moreover, the cell comprises two switches, a DC voltage source, a current pulse generator (PG), and a set of four variable resistors, which are used for programming the intercellular connections;

setting a variable resistor to a very high resistance (higher than that of the input memristors) will cut off the communication of the output to the connecting cell in the corresponding direction. Of course, memristors could be used here as well, only that their state is not supposed to change during computations, which is why we preferred to use variable resistors in the schematic.

All input memristors are initialized in R_{OFF} whereas the state-memristor is initialized in R_{ON}. Switch S_1 is initially open and the current pulse generator is driving switch S_2. In this particular configuration the cell only receives input signals from its neighbors but does not send any signal back to them. This way we take into account structural dynamics of CA; i.e. when links between cells can be activated or deactivated depending on the states of the cells that these links connect. The variable resistors are used to facilitate mapping of directed (oriented) graphs on the CA-based HW platform; resistors are given a low/high enough value so as to practically allow/prevent particular directed connections corresponding to directed edges of a graph.

Following the approach in [33], PG drives switch S_2 and provides a current pulse wave I_{p} to the state-memristor in order to read its state. I_{p} consists of positive-negative paired current pulses of appropriate amplitude and duration. Switch S_2 is set to position '2' only when a positive current pulse is applied. The negative pulse sets switch S_2 to position '1' so the cell updates its state while taking into account the incoming signals (if any) and the previous stored state. Whenever a cell receives at least one input signal from its neighbors, then the corresponding input memristor(s) change their state from OFF to ON and subsequently the state-memristor switches from ON to OFF. This change is due to the complementary orientation of input and state memristors and prevents any subsequent change to the rest of the input memristors since voltage drop on their terminals will always be below the SET voltage threshold V_{SET}. Next, when the state-memristor is being read by a short positive pulse I_{p}, if it is found in the high-resistive (OFF) state then switch S_1 closes and the output ports are activated, thus allowing the cell to transmit DC voltage signals to its neighbors. Afterwards, regardless of the received inputs, the state of the cell cannot be further changed. In short, all cells have two possible states; 1whenever they receive an excitation input from at least one neighbor, they switch to a different and constant state. Taking into account the boundary memristance values for the state and the input memristors, the output DC voltage signal is set to specific amplitude capable of causing a change to the memristors of the adjacent cells. This voltage value is common in all cells regardless of the size of the CA. The proposed CA cell can be easily modified to consider different types of local neighborhoods even in three-dimensional mesh grids. Next we describe how a 2-d array, comprising such memristive cells, can be used for various scenarios of shortest path computations.

8.4.1.2 Algorithm Description

In order to conduct shortest-path computations we consider mesh grids of n vertices arranged on a discrete lattice, where any vertex is connected to its closest neighbors by links of nonnegative weights. The memristive CA which we propose consists of an $M \times N$ rectangular array of memristive cells $C_{\mathrm{M}}(i, j)$ with Cartesian coordinates (i, j), where $i = 1, 2, \ldots, M$, and $j = 1, 2, \ldots, N$. All cells are assumed to be identical as described in the previous section.

When searching for the shortest path between two given vertices of a mesh (single source, single destination), the given graph is first mapped onto the CA. Connections between the cells are configured as unidirectional, bidirectional, or completely closed, by using the variable resistors, thus facilitating the mapping of any kind of directed graph whose edges are of equal weight. Computation is initiated by triggering the source cell and then a wave of stimulation propagates in all directions and modifies accordingly the states of the cells. Since all neighbor connections are considered to be of equal weight, the aggregate weight of a particular path is here identified by the total number of "hops" in-between source and destination nodes. Computation is assumed finished when the state-memristor of the destination cell switches to the OFF state, or if the computation steps have exceeded the total number of nodes. The later means that there is no path connecting the two given cells. This computing approach is based on the work of Adamatzky [20] where one of the first CA-based algorithms for shortest path computations was proposed.

The source cell is the only one which is initialized with its state-memristor in OFF state so as to be able to transmit the output signal to its neighbors from the very beginning. In every step the switch S_2 is maintained in position '1' for time $t = \Delta t$, which is enough time for the memristors of the cell to complete their state transition when they are biased with an input signal. Next, switch S_2 returns to position '2' for the state-memristor to be read. As explained earlier, after receiving the very first input(s), any cell afterwards remains unaffected by the rest of its neighbors during computation.

When the computation is completed, the shortest path is found by reading the state of the input memristors of the cells (the reading circuit for either the input- or the state-memristor is not included in Fig. 8.4). Those found in the ON state indicate the exact neighbor(s) from where the input signal was received; these neighboring cells belong to the shortest path solution. If more than one of the input memristors is found in the ON state, it means that there are multiple paths reaching simultaneously to a particular cell. In this case all options are by default of equal total weight (hop count), so only one of the available paths can be randomly selected to be included in the final result. This way, all nodes forming the shortest path(s) can be located by searching backwards from the destination cell until the source cell is reached. The proposed memristive CA is also suitable for the computation of shortest paths between one source and multiple destination cells. The following pseudo-code describes the entire evolution of the memristive CA during shortest path computations:

```
initialize CA and define source and destination(s);
steps = 0;
while (no path is found) {
    $t$=0;
    for all cells do {
          calculate voltage drop on memristors;
          update memristance values according to the model;
          increase $t$;
    } while ($t$ <= $\Delta t$);
    for all cells {
          if (state-memristor is OFF) {
                close switch S1;
          }
    }
    increase steps;
    if (state-memristor of destination cell(s) is OFF) {
          shortest path found;
    }
    if (steps > total cells) {
          //no path was found;
          exit;
    }
}
calculate backwards the shortest path(s);
```

8.4.1.3 Simulation Results

Using the EJS environment [35] we developed a GUI-based simulation tool where we tested the effectiveness of the proposed memristive CA for different types of 2-d mesh grids. In all the conducted simulations, the parameters of the memristor model [34] were set to the following values $\{a_{\mathrm{x}}, b, c, m, f_{\mathrm{o}}, L_{\mathrm{o}}, V_{\mathrm{RESET}}, V_{\mathrm{SET}}\} = \{5 \times 10^4, 10, 0.1, 82, 310, 5, -1.5, 1.5\}$. The input memristors have $R_{\mathrm{OFF}} \approx 20$ kΩ, whereas the state-memristor has a much higher max resistance equal to $R_{\mathrm{OFF}} \approx 1$ MΩ. In each step, wherever input signals are applied to a cell, the voltage drop on each memristor inside the cell is calculated using the Kirchhoff's current law (KCL). While the input signal(s) are applied, the states of the memristors are updated according to the equations of the employed model during time $t = \Delta t$, which in our case was selected equal to 0.5 s for the reasons explained before. Afterwards, the state-memristor is being read; if its memristance is found higher than the predefined resistive threshold which defines the OFF state, the switch S_1 closes, otherwise S_1 remains open. We also note here that the output DC voltage in all cells was set to 20 V.

For comparison reasons we will use again the directed graph example of Fig. 7.11 where we wish to find the shortest path from the left top vertex to the right bottom vertex. This particular graph is the same with the one in [20] where Adamatzky tested his CA-based algorithm for shortest path computations. Figure 8.5 presents the solution of the problem for this specific scenario and also the visualization of the memristive CA simulation tool when computation is over. Boundary conditions are easily applied to the outmost cells of the square lattice by using the variable resistors of the cells.

In Fig. 8.5b the CA cells are the blue square nodes. Edge directivity is indicated as follows: the four adjacent neighboring squares of a specific cell can be either filled red to denote that there is no incoming connection from the opposite cell in this direction, or have no color to denote a normal incoming connection. After computation is over, some of the aforementioned adjacent squares of the cells, which belong to the shortest path solution, are filled green to indicate the direction of the input signal which was received from the previous cell in the shortest path. Hence, starting from the destination cell, the source cell can be reached by step-by-step moving to the next cells that are found in the direction indicated by the green marks in the visualization panel. The number of necessary computation steps for the proposed CA is equal to the number of hops within the shortest path. The presented simulation results are in absolute agreement with those published in [20].

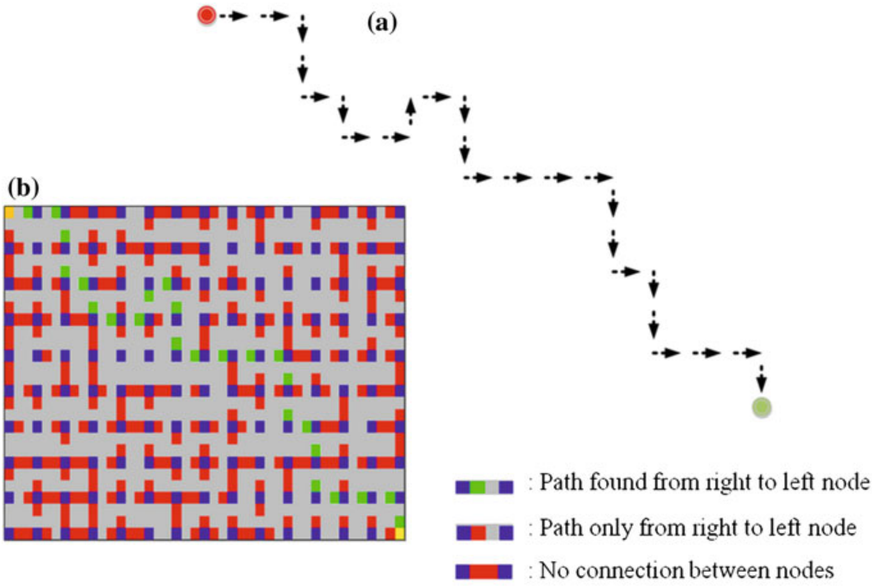

(a)
(b)
: Path found from right to left node
: Path only from right to left node
: No connection between nodes

Fig. 8.5 Shortest path computation for the directed graph of Fig. 7.11. **a** Macroscopically shows the solution to the problem where each arrow indicates the direction of every subsequent move. **b** Shows the visualization of the memristive CA when computation is completed. *Source* Vertex is marked with *orange* and *Destination* Vertex is marked with *yellow* color. The legend explains the color correspondence in the visualization of the solution

Figure 8.6 shows the simulation results of shortest path computations on a graph different from that of Fig. 7.11. More specifically, Fig. 8.6a concerns a single source single destination scenario, whereas in Fig. 8.6b the proposed memristive CA is used for a single source with multiple destinations shortest path search. In the latter, computation continues until all destinations are reached, provided that there is at least one path connecting them with the source cell. Afterwards, starting from any destination one navigates easily backwards to the source cell.

Figure 8.6c concerns the application of the memristive CA for the solution of the traveling salesman problem, defined on the same graph but when all the connecting edges are defined as bidirectional; i.e. when this particular graph is undirected. Otherwise, i.e. when working with a directed graph for this problem, it is possible for the CA to get stuck before the computation is over. This weakness, though, is attributed to the CA algorithm and not to its circuit implementation. In this specific example, the same source and destination cells with Fig. 8.6b are used. However, here we operate the same memristive CA in an iterative fashion; when the shortest path to a particular destination is found, the last destination cell is defined as source

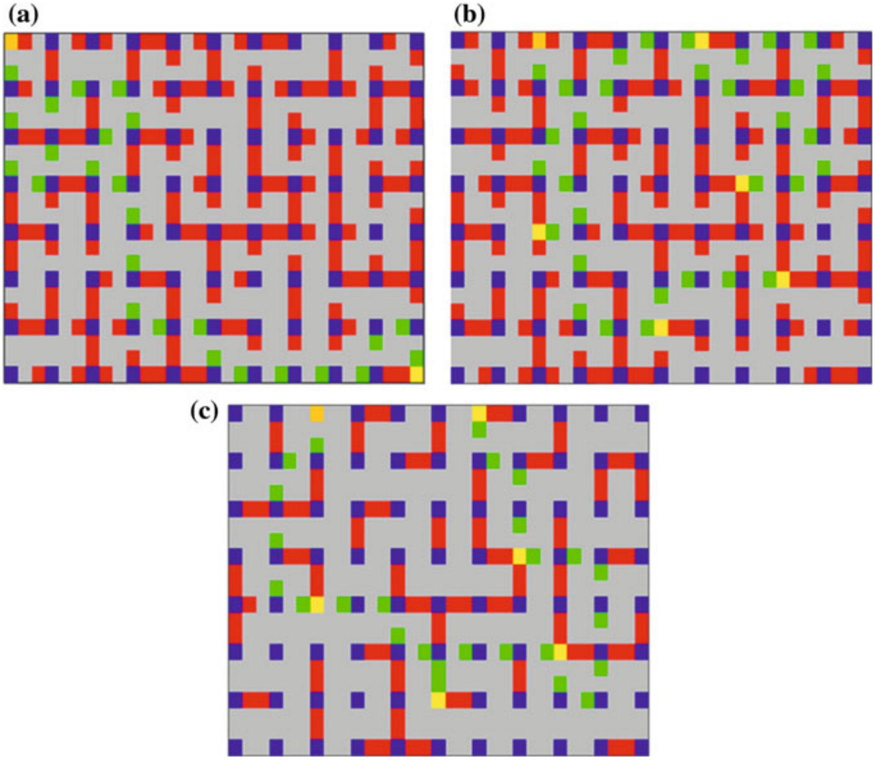


Fig. 8.6 Different shortest path computation scenarios on the same graph. **a** Single source single destination solution, **b** single source multiple destination solution, and **c** solution of the traveling salesman problem when all cell connections are set bidirectional. *Source* Vertex is marked with *orange* whereas *destination* vertices are marked with *yellow* color

cell and computation continues. In other words, the CA each time computes the shortest path to the closest of the available destinations. Therefore, the presented simulation results confirm the effectiveness of the proposed CA for both single source single destination and also multiple destination scenarios of the shortest path problem. Moreover, an iterative operation of the same CA is capable of providing a solution for the traveling salesman problem as well.

8.4.2 The Max Clique Problem

In the mathematical area of graph theory we define graphs as $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the vertex set and E is the edge set of graph G. The adjacency matrix of G is an $n \times n$ matrix denoted by A_{G} and defined as: $a_{\mathrm{ij}} = 1$ if there is an edge between vertices i and j of the graph, or $a_{\mathrm{ij}} = 0$ otherwise. When all the vertices in a graph are pair-wise adjacent (i.e. when for any $i, j \epsilon V$, there is an edge $(i, j) \epsilon E$), then the graph is called complete. In computer science, the clique problem refers to any of the problems which are related to finding particular complete sub-graphs ("cliques") in a graph; i.e., sets of elements where each distinct pair of elements is connected by an edge [37]. Cliques are considered one of the basic concepts in graph theory and are used in many mathematical problems and constructions on graphs with several applications in social networks, classification theory, economy, bioinformatics, and many more [38–40].

Particularly, a maximum clique of a graph G is a clique such that there is no other clique with more vertices [41]. The clique number $\omega(G)$ of a graph G is the number of vertices in a maximum clique of G. In the sections that follow we focus on the maximum (max) clique problem, an NP-hard problem in which the input is an undirected graph and the output is a max clique in the graph; whenever there are multiple max cliques, only one need be the output of computations.

8.4.2.1 Algorithm Description

The proposed memristive CA implementation is based on a novel approach presented in [42]. Such approach combines cellular neural networks (CNNs) and two-dimensional (2-d) binary CA. The CA neighborhood is the 5-cell von Neumann neighborhood, whereas the boundary conditions are periodic; i.e. the CA lattice is considered a torus and the CA neighborhood at the boundaries includes the cells that are found in the opposite edge of the CA lattice. We based our work on this particular parallel solving method because, according to [42], a simulation-based analysis showed that the quality of the provided solutions is superior even to that of the best existing parallel algorithm.

According to the solving algorithm, in computations the target graph is represented by its adjacency matrix which constitutes the initial state-space for the CA.

The CA evolution rule is summarized as follows: the next state of the cells is ‘1’ if the total number of ‘1’ states in the neighborhood is greater than the total number of ‘0’ states, or is ‘0’ otherwise. After every evolution step, the new CA state-space is compared with the previous state-space. Afterwards, the CNN uses the data from the comparison to compute a new graph which results after removing particular nodes from the initial graph (the circuit implementation of the CNN and the comparator modules is not considered within the scope of this chapter). Similarly, the adjacency matrix, which corresponds to the new graph, is computed and loaded back to the CA. Progressively, the vertices that are not directly connected are found. The aforementioned process continues until the max clique of the initial graph is found, i.e. when the adjacency matrix of the new graph includes only ‘1’.

More specifically, the rules that govern the generation of the new input state-space are given below [42], whereas the entire process is also summarized in the flowchart of Fig. 8.7:

- Case 1: For changes from ‘0’ to ‘1’ in the CA state-space:
  1. Delete the vertices that participate the most in those changes; e.g. if three such changes occur in the cells with coordinates $(i, j) = (0, 1)$, $(0, 5)$, and $(1, 3)$, apparently the vertex ‘0’ participates the most (i.e. appears more times), so it is deleted and its edges are shared with the vertices ‘1’ and ‘5’.
  2. If there is a pair (i, j) which occurs only once (i and j are unique among the observed changes) then combine these two vertices between them.
- Case 2: For changes from ‘1’ to ‘0’ in the CA state-space:
  1. If an ordered pair (i, j) is in those changes, then take the vertices $i - 1$, $i + 1$, $j - 1$, and $j + 1$, i.e. those in the von Neumann neighborhood, which have ‘0’ and apply the same rules as above.
  2. In case 1 give priority to the neighbors of the vertices that changed from ‘1’ to ‘0’.

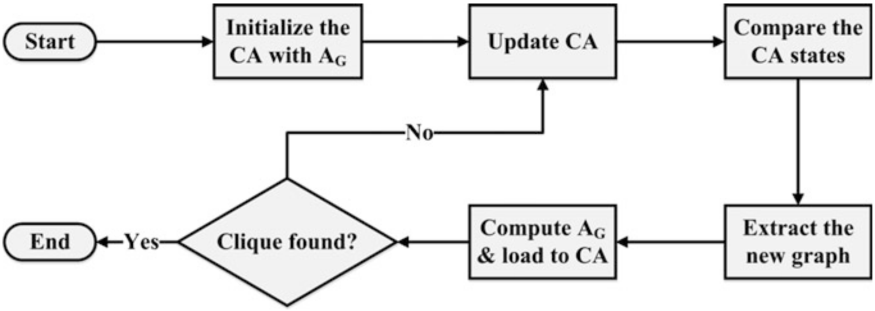


Fig. 8.7 Flow chart describing the communication flow among the different circuit modules during computation

8.4.2.2 Circuit-Level Implementation

Following the developed circuit layout methodology, below we describe in detail the memristive CA cell which implements the CA evolution rule, given in the previous section. As shown in Fig. 8.8, the proposed circuit bears several similarities with that presented before for the shortest path computing CA. Here part 1 of the CA cell circuit layout involves a summing component (it can be implemented with a summing amplifier as previously shown in Fig. 3.16) where the state voltages of the neighborhood are summed and the aggregate input voltage is then applied to the FPM (the state memristor) in the memristive part (part 3) of the layout.

The binary cell state is encoded in the memristance of this single FPM as R_{ON} for ‘1’ and R_{OFF} for ‘0’. The memristance is then decoded to a corresponding voltage V_{STATE} which is driven to the output of the cell and is also used internally in part 1 of the layout. More specifically, if the state of the memristor is R_{ON} then the switch S_2 connects the positive voltage source to the output of the cell (position 1), otherwise the switch S_2 is set to position 2 and the negative voltage value is communicated to the output. Consequently, since the cell output can be either V_{STATE} or $-V_{\mathrm{STATE}}$, the aggregate voltage, which results from the summation in part 1, will generally be either higher than V_{STATE} or lower than $-V_{\mathrm{STATE}}$. We assume here that a voltage $V \geq V_{\mathrm{STATE}}$ is capable of causing the memristor to switch to R_{ON}, i.e. the applied

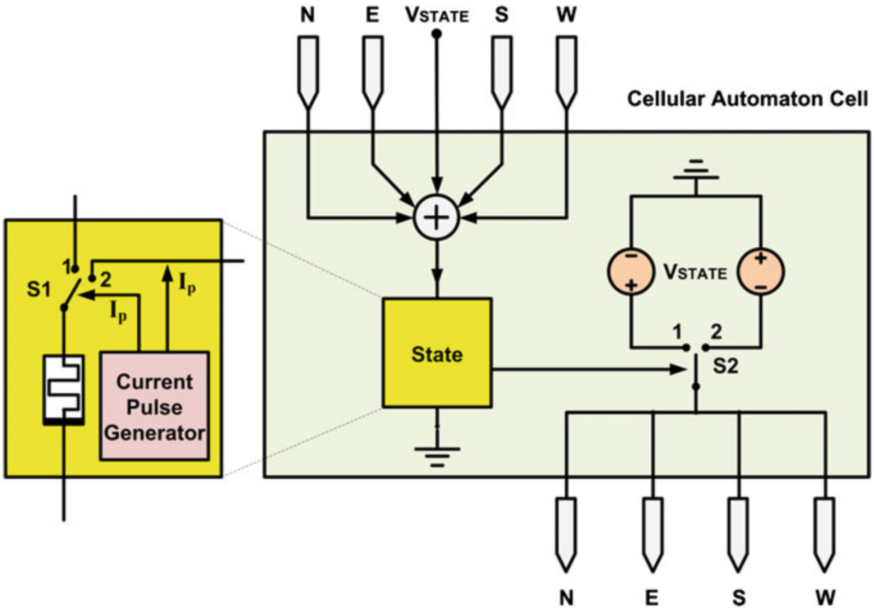


Fig. 8.8 A memristive CA cell with four (external) inputs and one output connecting with the nearest cells belonging to the von Neumann neighborhood. N, E, S, and W denote the four geographical neighbors, namely north, east, south and west

voltage is higher than the corresponding SET threshold V_{SET}. Similarly, a voltage $V \leq -V_{\mathrm{STATE}}$ will be lower than the RESET threshold V_{RESET}, so it will cause the memristor to switch to R_{OFF}.

Therefore the output DC voltage is adjusted to a specific value after taking into consideration the switching thresholds of the memristors. This voltage value is common in all cells regardless of the size of the CA lattice, which will be by default $n \times n$ to be able to hold the values of the adjacency matrix of the initial (i.e. the largest) graph. Apparently, the smaller the new graphs that result during the computation, the smaller the portion of the CA lattice that will be used each time in computations. Regarding the maximum input voltage applied to the memristor, we assume that it is not enough to exceed the device tolerance and thus to cause any damage; otherwise, additional protecting circuitry for the memristors will be needed. Moreover, the proposed CA cell can be easily modified to support a Moore neighborhood, e.g. by using composite memristive switches with higher switching thresholds instead of a single memristor, as presented in Chap. 3.

Inside the cell, the current pulse generator drives the switch S_1 and provides a current pulse wave I_{p} to the state-memristor in order to read its state. I_{p} consists of positive-negative paired current pulses of appropriate amplitude and duration. The positive current pulse sets the switch S_1 to position 2 so the resistive state of the memristor is translated to a corresponding voltage which controls the switch S_2. On the other hand, a negative current pulse sets the switch S_1 to position 1, so the cell state can be updated while taking into account the incoming voltages, including the previous stored cell state (V_{STATE}).

8.4.2.3 Simulation Results

Using the EJS environment [35] we developed a simulation software tool where we tested the effectiveness of the proposed memristive CA for the set of small (for readability reasons) graph examples which are presented in Fig. 8.9. In all conducted simulations the parameters of the memristor model [34] were set to the following values $\{a_{\mathrm{x}}, b, c, m, f_{\mathrm{o}}, L_{\mathrm{o}}, V_{\mathrm{RESET}}, V_{\mathrm{SET}}\} = \{5 \times 10^4, 10, 0.1, 82, 310, 5, -0.5, 0.5\}$. The memristance ratio was selected to be $R_{\mathrm{OFF}}/R_{\mathrm{ON}} \approx 2 \times 10^2$ with $R_{\mathrm{OFF}} \approx 400$ kΩ and $R_{\mathrm{ON}} \approx 2$ kΩ. We note here that there is no special requirement for the memristance range, provided that the two boundary memristances of the memristor can be easily distinguished during read-out. Moreover, the value of the V_{STATE} voltage in all cells was set to 1 V. The input signals of the cell are applied to the FPM during time $\Delta t = 0.5$ s, which is enough for the memristor to update its state according to the equations of the employed model and the aforementioned set of parameter values. When the state-memristor is read, its memristance is compared with two predefined resistance thresholds which denote whether the stored state is ON or OFF. Afterwards, the state of the controlled switch S_2 is adjusted accordingly.

Next, we present in detail the computing process for the first of the demonstrated graph examples whereas, for readability reasons, for the rest of them we simply discuss the solution which resulted from the proposed CA-based approach.

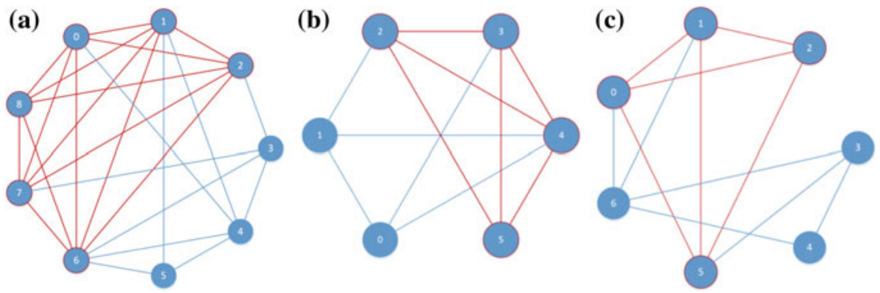


Fig. 8.9 A set of three **a**–**c** target graph examples. The *maximum clique* in each graph is highlighted with *red* color

Considering the graph of Fig. 8.9a which comprises 9 vertices, the initial state of the CA (i.e. the 9 × 9 adjacency matrix of the target graph) is demonstrated in Fig. 8.10a. After one step of evolution according to the local rule, the CA state-space becomes that of Fig. 8.10b. As mentioned earlier, here ‘1’ corresponds to the ON state and ‘0’ to the OFF state of the memristor, respectively. We highlight the CA state-changes from ‘0’ to ‘1’ which concern the cells with coordinates: (5, 3), (5, 0), (4, 2) and (3, 1). The vertex which appears the most in these changes is apparently vertex 5; hence vertex 5 is combined with vertices 0 and 3. Furthermore, the pair (4, 2) occurred only once, i.e. the vertices 4 and 2 are unique among the observed changes. So vertex 4 is also combined with vertex 2. On the other hand, the CA cells whose value changed from ‘1’ to ‘0’ have coordinates (4, 8), (5, 0) and (3, 0). Similarly, here the vertex which participates more is vertex 0, which however is already combined with vertex 5 from the previous case, so no further action is needed. The new adjacency matrix, corresponding to the updated graph (given from the CNN), is shown in Fig. 8.10c. It can be seen that for such a simple graph the max clique was found correctly after only a single computation step; the adjacency matrix is now smaller in size but full of ‘1’, meaning that there is an edge between any two of the remaining six vertices in the extracted sub-graph.

For the graph examples of Fig. 8.9b, c we simulated a 6 × 6 and a 7 × 7 CA lattice, respectively. In both cases the algorithm computed the solution after only one evolution step. The max clique in these two cases comprises four vertices. The memristive CA cell implementation worked fine even in more complicated simulated graph examples and the memristors switched state as expected according to the neighborhood state. Therefore, the presented results confirm the effectiveness of the proposed memristive CA for max clique computations.

8.4.3 The Sorting Problem

Since the dawn of computing the sorting problem has been one of the most extensively researched subjects [43]. The main purpose of sorting information is to

(a)

	0	1	2	3	4	5	6	7	8
0	1	1	1	0	**1**	**0**	1	1	1
1	1	1	1	**0**	1	1	1	1	1
2	1	1	1	1	**0**	0	1	1	1
3	0	**0**	1	1	1	**0**	1	1	0
4	**1**	1	**0**	1	1	1	1	0	0
5	**0**	1	0	**0**	1	1	1	0	0
6	1	1	1	1	1	1	1	1	1
7	1	1	1	1	0	0	1	1	1
8	1	1	1	0	0	0	1	1	1

(b)

	0	1	2	3	4	5	6	7	8
0	1	1	1	0	**0**	**1**	1	1	1
1	1	1	1	**1**	1	1	1	1	1
2	1	1	1	1	**1**	0	1	1	1
3	0	**1**	1	1	1	**1**	1	1	0
4	**0**	1	**1**	1	1	1	1	0	0
5	**1**	1	0	**1**	1	1	1	0	0
6	1	1	1	1	1	1	1	1	1
7	1	1	1	1	0	0	1	1	1
8	1	1	1	0	0	0	1	1	1

(c)

	0	1	2	3	4	5
0	1	1	1	1	1	1
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1

Fig. 8.10 CA state-space during max clique computation for the graph in Fig. 8.9a. **a** Shows the initial CA state-space, and **b** shows the state-space after one step of evolution. The result of computation is shown in (**c**)

optimize its usefulness for specific tasks which require sorted input data. Related work on memristive circuit implementations for this particular problem includes sorting networks which use comparators built of series of reciprocally placed memristors, first proposed in [2]. However, the aforementioned network-based hardware (HW) approach has the following disadvantages which render its practical use difficult: (i) the comparators' output levels gradually degrade through cascading, thus signal rectification is occasionally required; (ii) the complexity of interconnections increases significantly as the size of the problem (the number of inputs) increases.

In the following sections we address the classic sorting problem of n values (*Keys*) in a linear array by proposing a novel memristor-based circuit-level parallel approach inspired by one-dimensional (1-d) Cellular Automata (CA) [44]. Compared with the approach described in [2], the proposed design combines the computational capabilities and the size-independent simple structure of 1-d CA with the unique circuit properties of memristors, to provide a HW capable of executing computations within memory [45].

8.4.3.1 Algorithm Description

The algorithmic approach is based on [44]. Each cell of the 1-d CA is assigned a *Key* to be ordered. Within the cells, the *Key* value is encoded in the resistive state of threshold-type switching memristors and composite memristive components [15].

Sorting is performed by the parallel local exchange of *Keys* between neighboring cells. A cell exchanges its *Key* with either its right or its left neighbor; priority is given to the right swap in case a conflict is detected. All cells operate using the same update rule without knowing neither their index *i* nor the length *n* of the array. Figure 8.11a shows the 1-d CA neighborhood which, unlike mentioned previously for 1-d CA, here it consists of four cells. The state of each cell is defined by three memory elements: an integer value for the *Key*, and two binary elements for the Left/Right swapping rules, S_{L} and S_{R}. Fixed boundary conditions are applied to the extreme cells of the array. The CA update rule is described by Eqs. 8.1a, 8.1b and 8.2.

$$S_L = \begin{cases} 1, & \left(Key_i^t \geq Key_{i+1}^t\right) \cap \left(Key_i^t > Key_{i-1}^t\right) \\ 0, & else \end{cases} \tag{8.1a}$$

$$S_R = \begin{cases} 1, & \left(Key_i^t < Key_{i+1}^t\right) \cap \left(Key_{i+1}^t \geq Key_{i+2}^t\right) \\ 0, & else \end{cases} \tag{8.1b}$$

$$Key_i^{t+1} = \begin{cases} Key_{i-1}^t, & S_L = 1 \\ Key_{i+1}^t, & S_R = 1 \\ Key_i^t, & else \end{cases} \tag{8.2}$$

Every computation step comprises two stages: first the swapping rules are computed; then *Key* exchanges are locally performed. At a certain moment, the array is sorted and there are no more valid exchanges to take place; the swapping rules remain the same. Execution of the CA could be stopped: (i) either via a global control mechanism which would supervise at each step if at least a swap has occurred; (ii) by determining the worst-case for the computational time to sort the array, thus

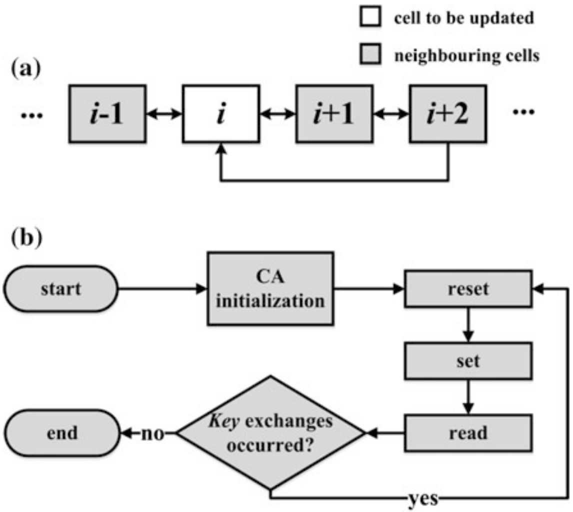


Fig. 8.11 **a** The 1-d CA neighborhood. **b** Flow chart describing the overall memristive sorting CA operation

bounding the total steps to a fixed limit. The worst-case occurs when the *Keys* are initially in the inverse order and the required time complexity is $O(2n\text{-}3)$. Here we assume the case (i) of the aforementioned termination mechanisms; the system stops functioning when the *Keys* have been finally sorted in descending order.

8.4.3.2 Circuit-Level Implementation

Here we describe in more detail the design of the fundamental memristive CA cell which implements the CA update rule and which we employed to create the 1-d parallel sorting computational structure. Figure 8.11b describes the overall function of the memristor-based sorting CA. First the CA array is initialized with the integer *Keys* to be ordered. Afterwards, the CA evolves and it stops when no more changes occur in the global CA state-space. CA cell circuit operation consists of three consecutive computational stages:

- RESET stage: all memristors are reset to the high resistive state (R_{OFF});
- SET stage: CA cells compute their next state;
- READ stage: the computed cell-state is stored and the cell's output is defined.

For readability reasons we have separated the whole circuit in separate schematics corresponding to the particular stages. The circuit for the SET stage is shown in Fig. 8.12a. Prior to their application to the memristive part (part 3) of the CA cell circuit layout of Fig. 8.3, the input signals undergo particular processing in parts 2 and 3. Specifically, S_{L} and S_{R} define how the input *Keys* will affect the next

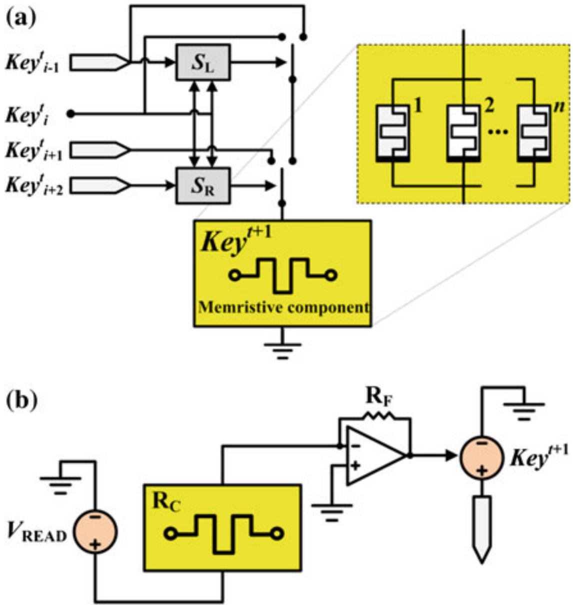


Fig. 8.12 Circuit schematics corresponding to the computational stages of the CA update rule. **a** SET stage, where the memristive composite component is shown in detail. **b** READ stage

cell-state by controlling two switches while operating according to Eqs. 8.1a, 8.1b. In part 3, the *Key* value is stored in the state of a composite memristive component, built according to the methodology presented in Chap. 3. Such device consists of n parallel memristors which have different V_{SET} thresholds, namely $V_{\mathrm{SET,1}} < V_{\mathrm{SET,2}} < \cdots < V_{\mathrm{SET,n}}$, but equal memristance ranges [R_{ON}, R_{OFF}].

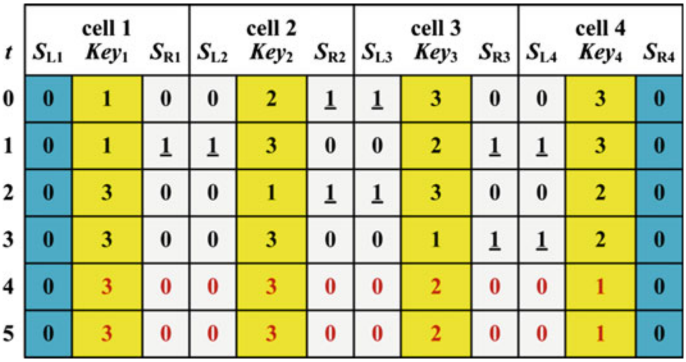
All memristors are formerly reset to the R_{OFF} state via a common negative voltage pulse. Since gradual resetting is not important, we assume a common negative threshold V_{RESET} for all memristors. So, the unconditional application of a negative voltage, higher than $|V_{\mathrm{RESET}}|$, will reset all memristors. However, under positive applied voltage the composite device operates as a multi-threshold memristive device. When the voltage amplitude falls between $V_{\mathrm{SET,1}}$ and $V_{\mathrm{SET,2}}$ only the first memristor switches to R_{ON}. If the voltage amplitude is between $V_{\mathrm{SET,2}}$ and $V_{\mathrm{SET,3}}$ two of the memristors switch states, and so on. Finally, a voltage higher than $V_{\mathrm{SET,n}}$ causes all the n memristors to switch. Regardless of possible variation in memristance boundaries, a high enough ratio $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ assures that the equivalent memristance will evolve while taking values which approximate the following: $\{R_{\mathrm{OFF}}/n, R_{\mathrm{ON}}, R_{\mathrm{ON}}/2, R_{\mathrm{ON}}/3, \ldots, R_{\mathrm{ON}}/n\}$ depending on the number of memristors that are set to R_{ON}. Hence, the number of parallel memristors determines the max value of the *Key*. Finally, in part 4 of the general circuit layout, there is a current-controlled DC voltage source which stores the next cell-state both for internal use as well as to define the output of the cell.

This is better explained in Fig. 8.12b where the circuit which implements the READ stage is demonstrated. The resistance of the composite memristive device is read by applying a positive DC voltage V_{READ} of low enough amplitude, which does not exceed any of the switching thresholds. The memristors are connected to a current-to-voltage converter (*I*/*V*) of variable external gain $R_{\mathrm{F}}/R_{\mathrm{C}}$, where the resistance of the feedback resistor is selected $R_{\mathrm{F}} = R_{\mathrm{ON}}$ and R_{C} is the variable composite memristance of the parallel memristors. The *I*/*V* output is approximately given by $m \times V_{\mathrm{READ}}$, where m is the number of memristors that are in R_{ON}. Only when all the n parallel memristors are in R_{OFF}, it is $R_{\mathrm{C}} \approx (R_{\mathrm{OFF}}/n) \gg R_{\mathrm{F}}$ and the *I*/*V* output voltage becomes very small. Based on the *I*/*V* output, the current-controlled DC voltage source is adjusted to hold the next *Key* value (the output of the cell).

8.4.3.3 Simulation Results

Figure 8.13 shows the simulation result from the application of the proposed approach to a set of *Keys* which are initially in the inverse order. Using the EJS environment [35] we developed a simulation tool where we tested the effectiveness of the proposed memristive CA. In all conducted simulations the parameters of the memristor model [34] were set to the following values $\{a_{\mathrm{x}}, b, c, m, f_{\mathrm{o}}, L_{\mathrm{o}}\} = \{5 \times 10^{4}, 10, 0.1, 82, 310, 5\}$.

Three parallel memristors were used in the CA cells to be able to encode the max of the *Key* values. For the memristors we assume [R_{ON}, R_{OFF}] = [2, 650] kΩ,



	cell 1			cell 2			cell 3			cell 4		
t	S_{L1}	Key_1	S_{R1}	S_{L2}	Key_2	S_{R2}	S_{L3}	Key_3	S_{R3}	S_{L4}	Key_4	S_{R4}
0	0	1	0	0	2	1	1	3	0	0	3	0
1	0	1	1	1	3	0	0	2	1	1	3	0
2	0	3	0	0	1	1	1	3	0	0	2	0
3	0	3	0	0	3	0	0	1	1	1	2	0
4	0	3	0	0	3	0	0	2	0	0	1	0
5	0	3	0	0	3	0	0	2	0	0	1	0

Fig. 8.13 Simulation result for sorting four *Keys*. Parameters S_L, *Key*, and S_R are shown for each CA cell. Boundary condition for S_{L1} (S_{R4}) corresponds to an applied voltage which is higher than max *Key* (lower than min *Key*). Simulation is completed after five CA evolution steps

$\{V_{SET,1}, V_{SET,2}, V_{SET,3}\} = \{0.5, 1.5, 2.5\}$V, and $V_{RESET} = -3$ V. The programming voltage signals are applied to the memristors during time $\Delta t = 0.5$ s. The amplitude of the applied SET voltage is 1, 2, or 3 V, depending on the corresponding stored *Key*. The applied voltages for the READ and the RESET stages are 1 and −4 V, respectively. During the READ stage, the memristors are reversely polarized, therefore their state is not affected since $V_{READ} < |V_{RESET}|$. Figure 8.13 presents the sequence of *Key*-exchanges during five sorting steps; computation stops when no more exchanges occur. For each CA cell we show the current *Key* and the two binary elements for the swapping rules (S_L and S_R); the latter remain the same between the last two sorting steps, something which is indicative of the termination of the process. The presented example constitutes a proof of concept of a general methodology for the implementation of fast, parallel data-sorting HW components exploiting the unique processing-in-memory property of emerging memristive technologies and the structural simplicity of CA-based circuits.

8.4.4 The Bin Packing Problem

Bin packing is a classic problem where a collection of objects of different volumes must be packed into a finite number of fixed-size containers (bins) in a way that minimizes the number of total used bins [46]. In computational complexity theory, it is a combinatorial NP-hard problem with many variations (e.g. linear, 2D, or 3D packing) which find application in several every-day practical problems related to minimization of space or time. Despite its NP-hard computational complexity, optimal solutions can be produced with heuristic algorithms [47]. A straightforward greedy approximation algorithm is the "First-Fit", which provides a fast solution by processing the items in arbitrary order while attempting to place each one into the first bin in which it will fit. If no bin is found, the item is put in a new empty bin.

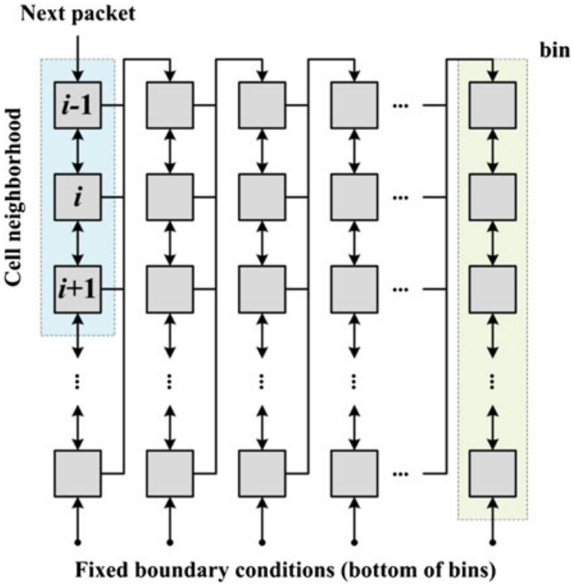
The required time complexity is $O(n \log n)$, where n is the number of elements to be packed. It has been shown that this algorithm achieves an approximation factor of 2, i.e. the number of bins used is no more than two times (2×) the optimal number of bins. However, in real applications the sizes to be packed may all be known in advance, hence better results are achieved by packing the largest objects first. Therefore, the First-Fit can be made much more effective by first sorting the list of elements in decreasing order, thus giving the "First-Fit decreasing" algorithm [48].

In the next sections we address the classic bin packing problem by proposing a CA-inspired circuit-level approach, capable of executing computation within memory. We describe the fundamental memristive cell which implements the desired 1-d CA rule and then employ it to create a sophisticated 2-d computational structure able to execute the First-Fit (decreasing) algorithm [49].

8.4.4.1 Algorithm Description

The general setup of the circuit which implements the First-Fit algorithm for a set of packets (i.e. inputs) which are processed sequentially, either in arbitrary or decreasing order (First-Fit decreasing), is shown in Fig. 8.14. The prior sorting process of the packets is considered readily implemented as described previously in this chapter, whereas here we focus only on the implementation part of the First-Fit algorithm. Overall, this 2-d array consists of a chain of vertically-placed 1-d CA arrays which represent separate equally-sized bins.

Fig. 8.14 Block diagram describing the setup of the CA-inspired circuit approach to the bin-packing problem



The CA neighborhood of the *i*th cell of each bin includes the top and bottom adjacent cells, as shown in Fig. 8.14. Fixed boundary conditions are applied to the last cells of every 1-d CA to denote the bottom of the bins. All employed 1-d CA are identical and the total number of their cells (i.e. the rows of the 2-d array) equals *bin_size* + 1, where *bin_size* is a positive integer. During the packing process, each new packet is introduced to the array from the top cell of the first (leftmost) bin. The top cells of the rest of the bins do not receive any input directly from outside. Instead, they are fed by the cells of the previous (lower-indexed) adjacent bin.

The overall function of the proposed system, which implements the packing process, is described in the flow chart of Fig. 8.15. After initialization, for every recently introduced packet, all bins function as 1-d CA under a common evolution rule which consists of three phases: SET, RESET, and READ. A new packet enters the array only when the previous packet is packed, i.e. when it has reached a constant final position in one of the available bins. The entire system is globally controlled and stops functioning only when all packets are finally packed according to the First-Fit algorithm.

The set of parameters which characterize the state of the *i*th CA cell at time moment t is the {*Packet_Size_on_Grid* ($PSoG^t_i$), *Space_Used* (SU^t_i), *Switch* (SW^t_i)}. $PSoG^t_i$ is a positive integer whose values correspond to the size of the assigned packet. SU^t_i is also a positive integer which corresponds to the total utilized space inside the bin where the cell belongs. SW^t_i is a particular flag which can take as a value either ‘1’ or ‘0’. After initialization, all cells have $\{PSoG^t_i, SU^t_i, SW^t_i\}$ = {‘0’, ‘0’, ‘0’}.

Every new packet sets the $PSoG^t_i$ parameter of the top cell of the leftmost bin to its particular size before the packing procedure is initiated. Then the packet moves sequentially downwards along each bin until it finally settles in an empty cell. Whenever the moving packet encounters another packet in the cell below it, it checks if there is enough space left in this bin by consulting the *SU* parameter of the next cell (SU^t_{i+1}). If it fits in the bin, then it remains in the current position and

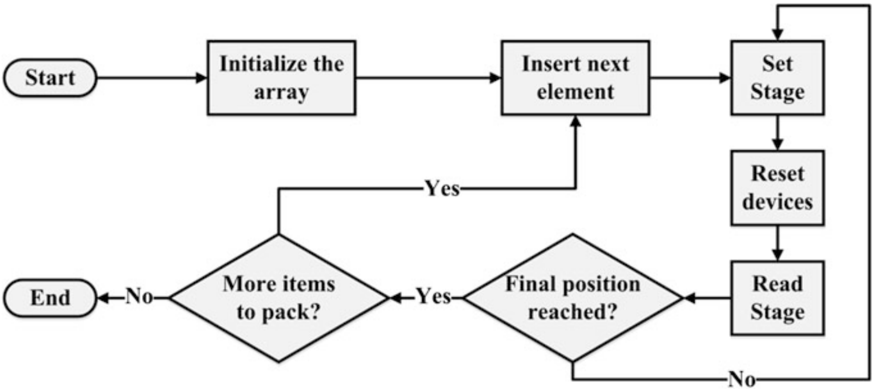


Fig. 8.15 Flow chart describing the algorithm (work flow) during the bin-packing process

updates its *SU* parameter to $SU_i^{t+1} = SU_{i+1}^t + PSoG_i^t$. However, if it does not fit in the bin, i.e. if it is $(SU_{i+1}^t + PSoG_i^t) > bin\_size$, then the packet continues the search in the next available bin. If the packet reaches the last cell of the bin without having encountered any other packets on his way (i.e. if it is the first packet inside the particular bin), is settles there and its *SU* parameter is updated to $SU_i^{t+1} = PSoG_i^t$. All the CA cells, which are located along the bottom boundary of the 2-d array, receive constant inputs $PSoG_{i+1}^t \neq$ ‘0’ and $SU_{i+1}^t =$ ‘0’ from below. Such combination corresponds to a “virtual” existing packet which however does not occupy any of the available space of the bin, i.e. the bin is still considered empty. The update rule for all 1-d CA cells is described in more detail in the following pseudo-code:

```
repeat
if ($PSoG_i^t$==‘0’)                            //the cell is empty
    if ($PSoG_{i-1}^t$==‘0’)                    //previous cell is empty
          do nothing;
    else
          set $PSoG_i^{t+1}$=$PSoG_{i-1}^t$;
    end
else                                            //the cell has a packet
    if ($SW_i^t$==‘1’)                          //flag is up
          reset the cell {$PSoG_i^{t+1}$, $SU_i^{t+1}$, $SW_i^{t+1}$}=‘0’;
          trigger the next bin;
    elsif ($PSoG_{i+1}^t$!=‘0’)                 //next cell has a packet
          if ($SU_i^t$==‘0’)                    //not in final position
                if ($SU_{i+1}^t$+$PSoG_i^t$>$bin\_size$)
                      set $SW_i^{t+1}$=‘1’;
                else
                      $PSoG_i^{t+1}$=$PSoG_i^t$;
                      $SU_i^{t+1}$=$SU_{i+1}^t$+$PSoG_i^t$;
                end
          else                                  //if in final position
                do nothing;
          end
    else                                        //next cell is empty
          reset the cell {$PSoG_i^{t+1}$, $SU_i^{t+1}$, $SW_i^{t+1}$}=‘0’;
    end
end
until ($packets\_left\_for\_packing$ ==‘0’);
```

8.4.4.2 Circuit-Level Implementation

As shown in Fig. 8.15, the update rule for all the 1-d CA cells consists of three discrete consecutive computational stages. The first stage is the SET stage, where the new state of the cell is computed. The second stage is the RESET stage, where the memristors inside the cell are reset (i.e. set to R_{OFF}). The final stage is the READ stage, where the recently calculated state of the cell is stored and the outputs of the cell are defined. For readability reasons we have separated the whole circuit schematic in three simpler schematics corresponding to each one of the aforementioned evolution stages.

More specifically, the dedicated circuit that implements the SET stage is shown in Fig. 8.16. The circuit includes two composite multi-state memristive devices which store the *Packet_Size_on_Grid* ($PSoG_i^t$) and the *Space_Used* (SU_i^t) parameters of the cell. Similar to the sorting CA implementation, which was presented before, such memristive circuit components in part 3 of the general CA cell layout, consist of three parallel memristors that are considered to have different V_{SET} thresholds, namely $V_{\mathrm{SET},1} < V_{\mathrm{SET},2} < V_{\mathrm{SET},3}$, and equal memristance ranges [R_{ON}, R_{OFF}]. So, when a positive voltage is applied to the composite devices, they operate as multi-threshold memristive devices. Assuming a high enough memristance ratio ($R_{\mathrm{OFF}}/R_{\mathrm{ON}}$), the equivalent memristance will take approximately either of the following values: $\{R_{\mathrm{OFF}}/3, R_{\mathrm{ON}}, R_{\mathrm{ON}}/2, R_{\mathrm{ON}}/3\}$, depending on the number of memristors that are set to R_{ON}.

Particularly, a positive voltage whose amplitude falls between $V_{\mathrm{SET},1}$ and $V_{\mathrm{SET},2}$ sets one of the memristors in R_{ON}. Similarly, if the voltage amplitude is between $V_{\mathrm{SET},2}$ and $V_{\mathrm{SET},3}$ it forces two of the memristors to switch their state, whereas a voltage higher than $V_{\mathrm{SET},3}$ causes all of them to switch to R_{ON}. $PSoG_i^t$ and SU_i^t parameters are encoded in the state of these composite devices. Therefore, for the circuit snapshot of Fig. 8.16, the maximum $PSoG_i^t$ and SU_i^t value is three arb. units since three memristors are used. However, this can be adjusted by modifying the number of parallel memristors and their voltage thresholds. Finally, a single memristor is used to hold the state of the SW_i^t flag.

Moreover, in part 4 of the general CA cell layout, there are three current-controlled DC voltage sources which are used to store the next state of the

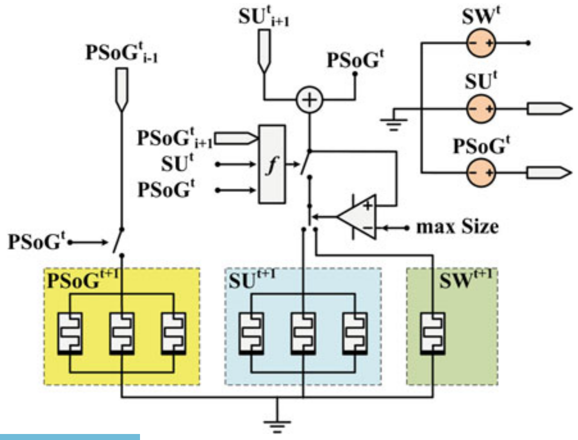


Fig. 8.16 Schematic of the circuit implementing the SET stage of the 1-d CA rule of the bin-packing process

cell, both for internal use as well as to define the output of the cell. In parts 1 and 2 there is a voltage adder and three switches which determine the programming voltage that is applied to every composite memristive device, according to the current cell state and the states of the adjacent cells. Additionally, a comparator is used to check if there is enough space inside the particular bin for the current packet. If there is enough space, then the SU_i^t parameter of the cell is updated. Otherwise, the SW_i^t is set up and in the next time step the packet moves to the next bin.

Similarly, the part of the circuit which is dedicated to reset the memristors is shown in Fig. 8.17. Since gradual resetting is not important for this application, we assume a common negative threshold for all memristors, i.e. V_{RESET}. This stage consists in the “conditional” application of a single negative voltage pulse of appropriate amplitude above $|V_{\mathrm{RESET}}|$, to the multi-threshold devices in order to reset the state of all memristors simultaneously. The application of such voltage is controlled by three switches which are controlled by interfacing circuitry that operates according to the CA update rule.

The last part of the proposed circuit implementation is about the last stage of the CA update rule, i.e. the READ stage, and it is particularly presented in Fig. 8.18. In this stage the state of the composite devices is read by applying a positive voltage V_{READ} of low enough amplitude so that it does not exceed any of the threshold values, thus it does not affect the state of the memristors. For each composite device we include a current-to-voltage (I/V) converter (inverting amplifier) whose external gain is modified according to the composite state of the memristors; the output of the converter will be each time approximately given by $n \times V_{\mathrm{READ}}$, where n is the number of memristors that are in R_{ON}. In part 4 of the CA cell layout, every cell state parameter is related to a corresponding DC voltage source and all of them are adjusted accordingly to hold the next cell state which is also its output. Flag SW_i^{t+1} is read using a voltage divider via a series resistor; its value defines whether or not the $PSoG_i^{t+1}$ value will be passed to the top cell of the next bin.

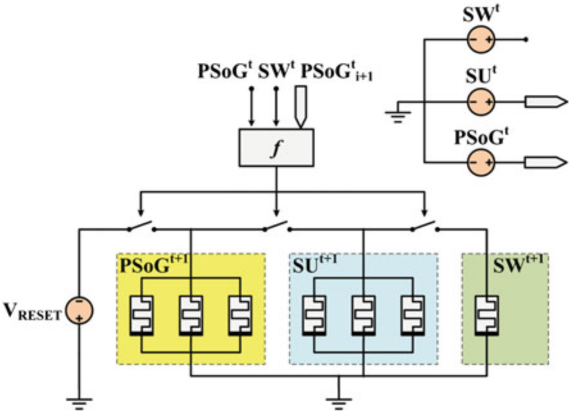


Fig. 8.17 Schematic of the circuit implementing the RESET stage of the 1-d CA rule of the bin-packing process

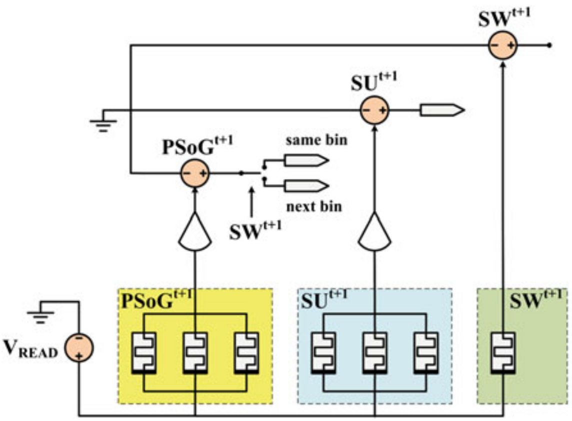


Fig. 8.18 Schematic of the circuit implementing the READ stage of the 1-d CA rule of the bin-packing process

Something important to note here is that, as the size of the problem changes (considering the number of packets to be packed, the number of bins, the packet size, etc.), so does the corresponding circuit for the packing (and the sorting processes). In fact, more memristors need to be included in the composite multi-threshold devices which store the cell state. Also, increasing the number of the packets impacts the total computational time which is relative to the size of the CA array.

More specifically, each new packet that enters the grid passes through all the unoccupied cells of each bin until it settles in its final position. Therefore, every packet needs approximately a total of $m \times n$ computing steps, where m denotes the bins that the packet passes through and n is the number of the cells in each bin. For the algorithm to function correctly, each bin must have [*bin_size*/min(*packet_size*)] + 1 cells. The latter extra cell is required during the navigation of the packets in the case that a bin is filled with packets of minimum size; there must be always an additional cell on the top of the bin for the next arriving packet. Everything considered, eventually the total computational time is proportional to the average required computing steps $x = number\_of\_packets \times \lambda \times bins\_used/2$, where λ is the number of cells per bin. The time factor x varies according to the size of the packets inside the sample. This assumption was made considering that every packet during the search will pass, on average, through half of the bins that will eventually be used. To test this assumption, the time factor was computed for some of the packet instances which were used in simulations and it is discussed in the simulations section. Compared with the true simulation steps, our calculations varied only between 0.23 and 17 %.

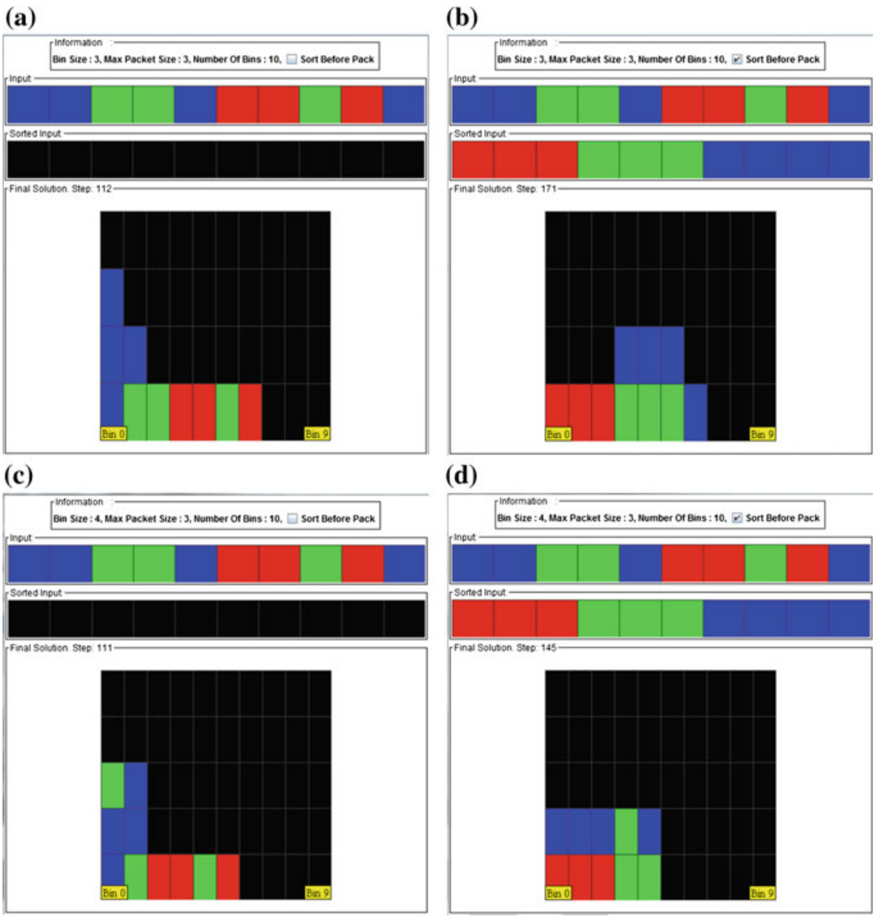
Finally, after experimentation it was figured out that a small improvement made to the circuit which implements the READ stage, could prove advantageous for two main reasons: (i) to improve circuit performance in terms of power by avoiding meaningless READ operations (the cells that contain packets which are in their final position will not change their state); (ii) to avoid possible state-drift of the stored information caused by continuous reading of memristors. Such modification concerns only an additional switch, which is controlled by signal SU_i^t, placed between the

V_{READ} DC voltage source and the memristive components. This way the READ stage becomes selective, meaning that not all cells will pass through it in every evolution step. In fact, the cells which contain packets that are in their final position (i.e. with $SU_i^t \neq 0$) will skip the READ stage and the output voltage sources will simply keep their values.

8.4.4.3 Simulation Results

Using the EJS environment [35] we developed a GUI-based simulation tool where we tested the effectiveness of all presented memristive CA circuit designs. In all conducted simulations the parameters of the memristor model [34] were set to the following common values for all memristors: $\{a_{\mathrm{x}}, b, c, m, f_{\mathrm{o}}, L_{\mathrm{o}}\} = \{5 \times 10^4, 10, 0.1, 82, 310, 5\}$ and $\{r_{\mathrm{MIN}}, r_{\mathrm{MAX}}\} = \{100, 1000\}$, corresponding to $R_{\mathrm{OFF}} \approx 650$ kΩ and $R_{\mathrm{ON}} \approx 2$ kΩ, respectively. Such a high $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ ratio makes it easier to distinguish the different composite states of the multi-state memristive components. The number of parallel memristors in the composite devices was selected in order to have a maximum packet size of three units and a maximum bin size of either three or four units. The voltage thresholds for the memristors forming the composite devices were chosen as $\{V_{\mathrm{SET},1}, V_{\mathrm{SET},2}, V_{\mathrm{SET},3}\} = \{0.5, 1.5, 2.5\}$ V with a common reset threshold $V_{\mathrm{RESET}} = -3$ V. Depending on the state of the memristors, the applied read voltage produces four different voltage levels via the inverting amplifiers, which are approximately equal to {0, 1, 2, 3} V. The duration of the SET and the RESET stage was selected, after experimentation with the memristor model, equal to $\Delta t = 0.5$ s, which is enough for the memristors to completely switch their states.

Simulation results are shown in Fig. 8.18. In the final state of the 2-d array (corresponding to that of Fig. 8.14), every colored cell corresponds to a packet whose color indicates its size. Overall, four simulations took place for a particular set of ten packets of various sizes and ten bins. The first two in Fig. 8.19a, b concern a set of ten bins of capacity equal to three arb. units, where the packets are processed either (a) in arbitrary or (b) in descending order. In this example both results are nearly-optimal giving the same number of used bins. However, starting with the largest objects first leads to better utilization of the available space; six bins are completely filled compared to five when the packets are arbitrarily introduced to the packing system. We repeated the same simulation after having increased the bin capacity to four units. The corresponding results are given in Fig. 8.19c, d. Here the impact of the prior sorting process is evident; one less bin is finally occupied while the total space utilization is much better with 4 out of 5 used bins being completely filled, compared to only 2 out of 6 when no prior sorting takes place. In each simulation scenario we include the duration of computations in time steps; cases (b) and (d) include the steps of the prior sorting process. Hence, increasing the capacity of the bins significantly lowers the necessary computational time when the candidate packets are sorted, whereas it has no significant effect for mixed entries (Fig. 8.19).



(a)

Information :
Bin Size : 3, Max Packet Size : 3, Number Of Bins : 10, □ Sort Before Pack

Final Solution Step: 112

(b)

Information :
Bin Size : 3, Max Packet Size : 3, Number Of Bins : 10, ☑ Sort Before Pack

Final Solution Step: 171

(c)

Information :
Bin Size : 4, Max Packet Size : 3, Number Of Bins : 10, □ Sort Before Pack

Final Solution Step: 111

(d)

Information :
Bin Size : 4, Max Packet Size : 3, Number Of Bins : 10, ☑ Sort Before Pack

Final Solution Step: 145

Fig. 8.19 Simulation results **a**–**d** after the bin packing process. Two shelves of objects appear in the program display. The top shelf shows the candidate objects to be packed, where the sorted entry presents the objects from largest to smallest. Colors {R, G, B} correspond to {3, 2, 1} sizes. The bottom shelf shows the results in the 2-d grid

We conducted an additional set of simulations in order to test both the functionality of the proposed circuits as well as the quality of the provided solutions for significantly larger packet instances. To this end we based our analysis on the benchmark data sets available in [50]. The selected simulation instances had maximum bin capacity = 100 arb. units, maximum packet size = 100 arb. units, and number of packets equal to 50 or 100. The benchmark data include the necessary total bins for each instance, so next we present the simulation results along with the benchmark solution to facilitate comparison between them. The name of each instance is encoded as follows: “NxCyWz”, where $x = 1$ when the number of packets is $n = 50$ or $x = 2$ when $n = 100$, $y = 1$ for bin capacity $c = 100$, and $z = \{1, 2, 4\}$ for a corresponding packet size within the range {[1, 100], [20, 100], [30, 100]} arb. units [50]. For every instance we conducted 20 different simulations named using the capital letters A-T.

According to the simulation results, shown in Tables 8.1, 8.2, 8.3, 8.4, 8.5 and 8.6, the circuit computes the optimal solution in most cases. Only in three instances the given solution is worse by one bin, highlighted in Tables 8.1 and 8.3. This is attributed to the First-Fit decreasing (FFD) algorithm implemented by the circuit, compared to the algorithm used by the corresponding benchmark set. According to [51], the upper bound of FFD is $\mathrm{FFD}(I) \leq (11/9) \times \mathrm{OPT}(I) + (6/9)$, where I is an instance of the problem, FFD(I) is the solution (the number of bins used) and OPT (I) is the optimal solution. This means that the FFD algorithm, in the worst case, it will give a solution equal to (11/9) × OPT(I) + (6/9), compared to the optimal solution OPT(I). In our implementation the FFD algorithm was chosen because it does not require knowledge of the state of all the bins (problem space), but instead it examines only locally every bin in order. This attribute is in line with the 1-d CA definition on which the circuit implementation is based, having no global control or inspection but only local connections.

8.4.5 The Knapsack Problem

The knapsack (or rucksack) problem [52] is an NP-hard problem in combinatorial optimization, defined as follows: given a set of items, each with a size and a value

Table 8.1 Instance N1C1W1

	A	B	C	D	E	F	G	H	I	J
S	25	31	**21**	28	26	27	25	31	25	26
B	25	31	**20**	28	26	27	25	31	25	26
	K	L	M	N	O	P	Q	R	S	T
S	26	33	30	**26**	32	26	28	25	28	28
B	26	33	30	**25**	32	26	28	25	28	28

S Simulation results; *B* Benchmark data

Table 8.2 Instance N1C1W2

	A	B	C	D	E	F	G	H	I	J
S	29	30	33	31	36	30	30	33	35	34
B	29	30	33	31	36	30	30	33	35	34
	K	L	M	N	O	P	Q	R	S	T
S	35	31	30	33	29	33	36	34	37	38
B	35	31	30	33	29	33	36	34	37	38

S Simulation results; *B* Benchmark data

Table 8.3 Instance N1C1W4

	A	B	C	D	E	F	G	H	I	J
S	35	40	36	38	38	32	**38**	40	35	37
B	35	40	36	38	38	32	**37**	40	35	37
	K	L	M	N	O	P	Q	R	S	T
S	41	35	41	39	34	38	34	38	36	42
B	41	35	41	39	34	38	34	38	36	42

S Simulation results; *B* Benchmark data

Table 8.4 Instance N2C1W1

	A	B	C	D	E	F	G	H	I	J
S	48	49	46	50	58	50	60	52	62	59
B	48	49	46	50	58	50	60	52	62	59
	K	L	M	N	O	P	Q	R	S	T
S	55	55	46	48	48	54	46	56	45	52
B	55	55	46	48	48	54	46	56	45	52

S Simulation results; *B* Benchmark data

Table 8.5 Instance N2C1W2

	A	B	C	D	E	F	G	H	I	J
S	64	61	68	74	65	65	73	70	67	67
B	64	61	68	74	65	65	73	70	67	67
	K	L	M	N	O	P	Q	R	S	T
S	72	62	65	64	64	68	65	67	66	66
B	72	62	65	64	64	68	65	67	66	66

S Simulation results; *B* Benchmark data

Table 8.6 Instance N2C1W4

	A	B	C	D	E	F	G	H	I	J
S	73	71	77	82	73	77	71	75	73	74
B	73	71	77	82	73	77	71	75	73	74
	K	L	M	N	O	P	Q	R	S	T
S	70	75	72	71	80	67	75	70	80	70
B	70	75	72	71	80	67	75	70	80	70

S Simulation results; *B* Benchmark data parameter, determine the total number of each item to be included in a collection so that: (i) the total size is less than or equal to a given limit (i.e. the knapsack's capacity); (ii) the total value of all the selected items is the largest possible. The problem name derives from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable of the available items. It is one of the most popular and most needed algorithmic problems; it often arises in resource allocation and in real-world decision-making processes in a wide variety of fields, including computer science, complexity theory, economy, etc.

Several algorithms are available to solve knapsack problems [53]. Among several known versions of this problem, the unbounded knapsack problem (UKP) places no upper bound on the number of copies of each kind of item to be included. George Dantzig proposed a greedy approximation algorithm to solve the UKP in [54]. His approach sorts the items in decreasing order of value per unit of size. Next, it proceeds to insert them into the sack, starting with as many copies as possible of the first kind of item until there is no longer space in the sack for more. However, for the bounded version of the problem (BKP), where the supply of each kind of item is limited, the algorithm may not lead always to optimal solutions.

In this section we address the knapsack problem by proposing a memristive CA circuit-level approach based on the Dantzig's algorithm [54]. Given the fact that the nature of this problem is very similar to that of the bin packing, and that the proposed algorithm requires a sorting procedure prior to further processing of the input data, in this case we will combine the circuit approaches which were proposed for these two problems in an efficient way, compatible with the requirements of the new target problem. The reader is kindly requested to refer to the previous sections of this chapter for relevant information and circuit schematics.

8.4.5.1 Algorithm Description

For the Dantzig's algorithm it was quickly figured out that, under certain assumptions, it could be considered as in line with the CA properties which we used to solve the bin packing problem. In fact, the items (packets) are sorted in descending order according to the value per unit of size, and then they are considered for possible packing. So far, the only notable difference between the First-Fit decreasing bin packing algorithm and Dantzig's algorithm lies in the parameter by which the items are sorted. Therefore, we based our solving approach on the same 2-d lattice of Fig. 8.14, this time representing the knapsack with a column of the 2-d lattice.

The packing procedure is the same with the one used in the bin packing solution, except for the extra parameter *Value_on_Grid* (VoG_i^t), which holds the total value of the items which are packed in a particular knapsack. This fourth parameter was included in the set of parameters which characterizes the state of the *i*th 1-d CA cell at time moment *t*: {*Packet_Size_on_Grid* ($PSoG_i^t$), *Value_on_Grid* (VoG_i^t), *Space_Used* (SU_i^t), *Switch* (SW_i^t)}. The packing process is described in detail in the following pseudo-code:

```
repeat
if ($PSoG_i^t$ ==‘0’)                              //the cell is empty
    if ($PSoG_{i-1}^t$==‘0’)                       //previous cell is empty
          do nothing;
    else
          set $PSoG_i^{t+1}$=$PSoG_{i-1}^t$;
          set $VoG_i^{t+1}$=$VoG_{i-1}^t$;
    end
else                                               //the cell has an item
    if ($SW_i^t$==‘1’)                             //flag is up
          reset the cell {$PSoG_i^{t+1}$, $SU_i^{t+1}$, $SW_i^{t+1}$, $VoG_i^{t+1}$}=‘0’;
          trigger the next knapsack;
    elsif ($PSoG_{i+1}^t$!=‘0’)                    //next cell has an item
          if ($SU_i^t$==‘0’)                       //not in final position
                if ($SU_{i+1}^t$+$PSoG_i^t$>$knapsack\_size$)
                      set $SW_i^{t+1}$=‘1’;
                else
                      $PSoG_i^{t+1}$=$PSoG_i^t$;
                      $VoG_i^{t+1}$=$VoG_i^t$;
                      $SU_i^{t+1}$=$SU_{i+1}^t$+$PSoG_i^t$;
                end
          else                                     //if in final position
                do nothing;
          end
    else                                           //next cell is empty
          reset the cell {$PSoG_i^{t+1}$, $SU_i^{t+1}$, $SW_i^{t+1}$, $VoG_i^{t+1}$}=‘0’;
    end
end
until ($remaining\_items\_for\_packing$==‘0’);
```

An important property of the proposed approach is the fact that, similar to the bin packing problem, it assumes many available identical knapsacks. Therefore, instead of providing a single solution for the items that fit in one knapsack, the solving process continues until there are no more items left, giving this way a set of different solutions (normally) ordered from the best to the worst, since every subsequent solution concerns only the remaining items that were not included in the previous solutions.

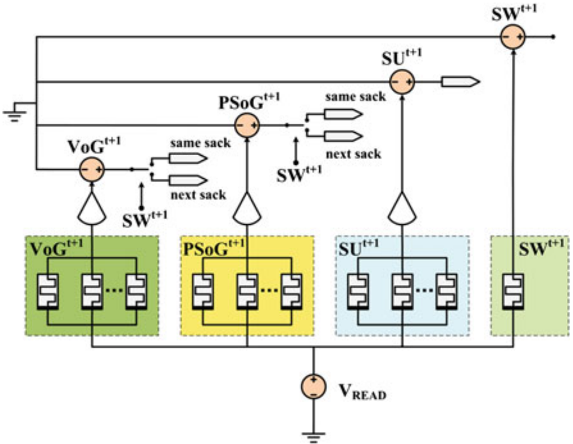
8.4.5.2 Circuit-Level Implementation

In order to implement the knapsack solving algorithm at circuit-level, the sorting and the bin packing circuits were slightly modified to accommodate the extra parameters mentioned above. In fact, compared with the circuits demonstrated in the previous sections, here in both processes there is another composite memristive component which was included in every CA cell to store the $value_i^t$ parameter.

As far as the sorting part is concerned, now there are two memristive multi-state devices (in part 3 of the general CA cell layout) which encode the $value_i^t$ and the $size_i^t$ of each item, which are assumed to be positive integer numbers and are used internally to compute the value per unit of size. The latter controls the position of the switches which determine the input voltage which will be applied to the memristive components. As described previously in the corresponding section, the sorting process is based on 1-d CA and here the CA evolution rule is the same, only that now there are two inputs for the cells and, consequently, two parameters which characterize the state of the CA cells. The interfacing circuitry, in parts 1 and 2 of the general layout, takes into consideration the value per unit of size. During the RESET stage, the state of all the memristors is reset to R_{OFF}, so that the cells are then able to compute their next state during the SET stage. Both the $value_i^t$ and the $size_i^t$ parameter of each CA cell are handled in the same way as shown in Fig. 8.12a. Finally, during the READ stage, the resistive state of every composite multi-state component is decoded and stored in the value of a corresponding DC voltage source, found in part 4 of the cell layout, as shown in Fig. 8.12b.

Regarding the packing part of the solving process, in each cell there is a comparator which is used to check if there is enough space inside the knapsack for the next packet. If there is enough space, then the SU_i^t parameter of the cell gets an appropriate value. Otherwise, the SW_i^t flag is set up. In the next time step, the packet continues the navigation in the next available knapsack, or it is discarded if only one knapsack is activated. In the knapsack solving process, as it is described in the provided pseudo-code, the extra parameter *Value_on_Grid* (VoG_i^t) is used only for the storage of information and has no participation in the computations taking place in the interfacing circuitry (parts 1 and 2 of the general cell layout) which defines the programming signals that are applied to the memristive components. Its value is stored in a corresponding voltage source and it is communicated to the neighboring CA cells. Overall, this parameter is handled exactly as the *Packet_size_on_Grid* ($PSoG_i^t$) in all the three stages of each computing step, as it is demonstrated in the circuit schematic of Fig. 8.20, which corresponds to the READ stage. The inclusion

Fig. 8.20 Schematic of the circuit implementing the READ stage of the 1-d CA update rule of the packing process



of any more illustrative circuit schematics in this section was considered redundant and the reader is kindly requested to refer to the previous section dedicated to the bin packing problem. Of course, likewise mentioned before, the number of necessary memristors used in each multi-state memristive switch varies according to the value range of the corresponding stored parameter. As a consequence, the supported value-ranges are subject to particular restrictions concerning the tolerance of the memristors to the maximum voltage that can be applied to them, as it was explained in detail in Chap. 3.

8.4.5.3 Simulation Results

In order to test the functionality of the memristive CA circuit designs for the knapsack problem, we conducted a number of simulations using the EJS environment [35] with which we developed a GUI-based simulation tool. In all conducted simulations, the parameters of the memristor model [34] were set to the following common values for all memristors: $\{a_{\mathrm{x}}, b, c, m, f_{\mathrm{o}}, L_{\mathrm{o}}\} = \{5 \times 10^4, 10, 0.1, 82, 310, 5\}$ and $\{r_{\mathrm{MIN}}, r_{\mathrm{MAX}}\} = \{100, 1000\}$, corresponding to $R_{\mathrm{OFF}} \approx 650$ kΩ and $R_{\mathrm{ON}} \approx 2$ kΩ, respectively. Such a high $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ ratio makes it easier to distinguish the different composite states of the multi-state memristive components. The number of parallel memristors in the composite devices was selected while taking into account the maximum required values of the parameters. Likewise in the bin packing problem, the duration of the SET and the RESET stage in this case was selected again equal to $\Delta t = 0.5$ s, which is enough time for the memristors to completely switch their states.

We based our analysis on two specific problem data sets which are available online [55]. The first data set concerned 5 items and a knapsack capacity of 26 arb. units. The five items have size = {12, 7, 11, 8, 9} and value = {24, 13, 23, 15, 16}. The optimal solution for this set is the following selection {–, X, X, X, –}, where '–' denotes a not included item and 'X' a selected item in the knapsack. Hence, two of the five available items are discarded. In the given solution the selected items have size = {7, 11, 8} and value = {13, 23, 15}. Therefore, the total accumulated size is 26 arb. units (a full knapsack) and total value equal to 51 arb. units. For this input data set the simulated approach, proposed here, gave the results summarized in Fig. 8.21a. We used two identical knapsacks, i.e. two columns in the 2-d array structure, to include all the available items and thus provide two alternative solutions, where the second one contains the items that did not fit in the first knapsack. Every knapsack consists of three CA cells and, in Fig. 8.21a this property corresponds to the columns of the table. Knapsack #1 contains two items with total (size, value) = (23, 47) arb. units, whereas the knapsack #2 (the data shown in the last two rows of the table) includes the three remaining items with total (size, value) = (24, 44) arb. units. Knapsack #1 is the solution provided by the circuit implementation of Dantzig's algorithm, whereas knapsack #2 is an alternative solution which our circuit approach is able to compute exploiting the entire set of available items. Based on the "value per unit of size" parameter, solution #1 is apparently better.

(a)

	Knapsack #1			SUM
size	0	12	11	23
value	0	24	23	47
	Knapsack #2			**SUM**
size	9	7	8	24
value	16	13	15	44

(b)

	Knapsack #1				SUM
size	3	4	10	31	48
value	5	7	20	70	102
	Knapsack #2				
size	0	6	19	20	45
value	0	10	37	39	86

Fig. 8.21 Simulation results having two available knapsacks for each one of the two **a**, **b** benchmark data sets

However, it is not in accordance with the optimal solution of (size, value) = (26, 51) arb. units, which utilizes the entire available capacity of the knapsack.

The second data-set concerned 7 items and a knapsack capacity of 50 arb. units. The seven items have size = {31, 10, 20, 19, 4, 3, 6} and value = {70, 20, 39, 37, 7, 5, 10}. The optimal solution for this set is the following selection {X, –, –, X, –, –, –}, hence the total accumulated size is 50 arb. units (a full knapsack) and total value equal to 107 arb. units. For this input data set the simulated approach gave the results summarized in Fig. 8.21b. We used again two identical knapsacks comprising 4 CA cells, so there are two different solutions. Knapsack #1 contains four items with total (size, value) = (48, 102) arb. units, whereas the knapsack #2 includes the three remaining items with total (size, value) = (45, 86) arb. units. Likewise before, knapsack #1 is the solution provided by the circuit implementation of Dantzig's algorithm, whereas knapsack #2 is an alternative solution. Based on the "value per unit of size" parameter, solution #1 is again better. However, once again neither of the given solutions is the optimal (size, value) = (50, 107) arb. units.

In conclusion, the simulation results confirmed the correct operation of the circuit-level approach to the knapsack problem. However, the notable weakness of the implemented greedy approximation algorithm to provide the optimal solution for a particular data set, is attributed to the fact that the basic criterion for the item selection is their value per unit of size and not the better utilization of the capacity of the knapsack. Nevertheless, depending on the circuit configuration, the proposed CA-inspired circuit approach has the ability to pack all the available items in separate knapsacks. This way, it provides packing solutions for all the items so that the value per unit of size in every knapsack is the best possible, whereas all the solutions are also sorted according to this ratio.

8.5 Overview and Comparison

The contribution of this chapter consists in the combination of a powerful computational tool with the unique circuit properties of memristors within CA-inspired hardware (HW) implementations of known algorithms for several NP-hard artificial intelligence (AI) problems. The presented approach uses the memristor both for encoding of information and for computing. Memristors are analog devices, so they could be theoretically programmed to any intermediate conductance between the boundary values if accurate programming pulses were applied to them. However, in all the proposed CA circuit designs we use instead composite multi-state memristive components and achieve multiple stable levels of conductance in a more robust manner. Moreover, thanks to the CA-compatibility of the chosen algorithms, the proposed circuits are capable of parallel processing of information. Using memristive components instead of conventional CMOS registers to store the CA cell state values offers the advantage of: (i) potentially smaller circuit area, because the memristors permit higher integration density; (ii) nonvolatile storage of information; (iii) simple circuitry, e.g. using opamps instead of complex digital comparators; (iv) execution of computations in memory.

To the best of our knowledge, we formulated the first general circuit design methodology for memristive CA cells. Based on it, we proposed several CA cell implementations which we then used to design 1-d and 2-d computational structures. For all the target AI problems we presented the fundamental memristive CA cell which implements the corresponding CA rule. The cell designs could be easily modified to support different types of local neighborhoods, even for 3-d CA computational structures. The correct functionality of all the presented designs was verified via system-level simulation using the memristor device model of Chap. 2 and, in most cases, published benchmark data sets for comparison.

We were particularly able to design and simulate a 2-d structurally dynamic memristive CA capable of detecting the nodes of a given mesh belonging to a shortest path solution. Compared to the memristive network-based approach to shortest path computations in a square lattice, presented in Chap. 7, the CA-based memristive circuit approach: (i) has easier initialization process, since the CA requires only a single pulse to simultaneously reset the state of the anti-serially connected memristors in each cell (all cells are simultaneously initialized), whereas networks require all memristors to be accessed and programmed sequentially; (ii) requires access to fewer devices, since the CA comprises $M \times N$ cells whereas the memristive network would contain $(M - 1) \times N + M \times (N - 1)$ memristive connections; (iii) requires common voltage supply, since all the CA cells operate with the same constant voltage, whereas the network requires a variable ramp-waveform voltage to reach a proper amplitude; (iv) supports directed graph-based problems, since the CA cells permit the easy projection of edge directivity patterns whereas the network's homogeneity and regularity impede such property, unless certain modifications are made; (v) has predefined max computation time, since the CA will finish computation within up to $M \times N$ steps, or else there is no solution, whereas a network has no such time limit. Nevertheless, unlike CA, whose operation depends also on conventional electronics, the network's operation is expected to provide computing capabilities at a rate which has absolutely no dependence on the algorithmic complexity.

Most of the aforementioned advantages are generally offered by all the presented CA designs, among which there was an approach to perform sorting of data in a linear array and solving the 1-d bin packing problem. The integration of them permitted the implementation of more advanced algorithms which improved the bin packing solutions and also found application in the solution of the knapsack problem. However, further architectural improvements and modifications to all the implemented algorithms which will facilitate exclusively parallel processing of information, as well as real circuit simulations using environments with integrated-circuit-emphasis (SPICE), will be needed in an attempt to establish such novel CA-based early circuit approaches to the solution of complex NP-hard problems. Furthermore, cost-related issues concerning the implementation of multi-state components consisting of multiple memristors with different switching thresholds in the same dice, will be investigated in depth in relation to real experimental data. Replacement of the composite multi-state components with single memristors, and of the extra computing/interfacing circuitry with memristor-based circuits, will eventually improve density and power consumption of such computing architectures. Currently, there is a growing variety of systems that exhibit memristive behavior. Therefore, it is much expected that experimental implementation of memristive CA computational structures could be done in massively parallel processors, or even in neuromorphic computing architectures of the near future.

References

1. International Technology Roadmap for Semiconductors (ITRS) (2013) (Online), Available: http://www.itrs.net/. Accessed June 2014
2. M. Klimo, O. Such, Memristors can implement fuzzy logic (2011) (Online), Available: http://arxiv.org/abs/1110.2074
3. S. Park, J. Park, S. Kim, W. Lee, B.H. Lee, H. Hwang, Programmable analogue circuits with multilevel memristive device. IET Electron Lett **48**(22), 1415–1417 (2012)
4. V. Ntinas, I. Vourkas, G. C. Sirakoulis, LC filters with enhanced memristive damping, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Lisbon, 2015
5. M. Di Ventra, Y.V. Pershin, The parallel approach. Nat. Phys. **9**, 200–202 (2013)
6. D.B. Strukov, K.K. Likharev, CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. Nanotechnology **16**(6), 888–900 (2005)
7. A.A. El-Slehdar, A.H. Fouad, A.G. Radwan, Memristor based N-bits redundant binary adder. Microelectron. J. **46**(3), 207–213 (2015)
8. Y. Pershin, M. Di Ventra, Practical approach to programmable analog circuits with memristors. IEEE Trans. Circ. Syst. I Regul. Pap. **57**(8), 1857–1864 (2010)
9. S. Shin, K. Kim, S. Kang, Memristor applications for programmable analog ICs. IEEE Trans. Nanotechnol. **10**(2), 266–274 (2011)
10. E.A. Vittoz, Future of analog in the VLSI environment, in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, New Orleans, LA, USA, 1990
11. M. Laiho, E. Lehtonen, Arithmetic Operations within Memristor-Based Analog Memory, in *12th International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, Berkeley, CA, 2010
12. Y.V. Pershin, M. Di Ventra, Solving mazes with memristors: a massively parallel approach. Phys. Rev. E **84**, 046703 (2011)

13. Y. Leblebici, H. Ozdemir, A. Kepkep, U. Cilingiroglu, A compact high-speed (31, 5) parallel counter circuit based on capacitive threshold logic gates. IEEE J. Solid State Circuits **31**(8), 1177–1183 (1996)
14. M. Halbach, R. Hoffmann, Implementing cellular automata in FPGA logic, in *18th International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, New Mexico, 2004
15. I. Vourkas, G.C. Sirakoulis, On the generalization of composite memristive network structures for computational analog/digital circuits and systems. Microelectron. J. **45**(11), 1380–1391 (2014)
16. G. Papandroulidakis, I. Vourkas, N. Vasileiadis, G.C. Sirakoulis, Boolean Logic Operations and Computing Circuits Based on Memristors. IEEE Trans. Circ. Syst. II Express Briefs **61** (12), 972–976 (2014)
17. I. Vourkas, D. Stathis, G.C. Sirakoulis, Massively parallel analog computing: Ariadne's thread was made of memristors. IEEE Trans. Emerg. Top. Comput. (2015 in press). doi: 10.1109/TETC.2015.2420353
18. B. Chopard, Cellular automata modeling of physical systems, in *Computational Complexity*, ed. by R.A. Meyers (Springer International Publishing, New York, NY, 2012), pp. 407–433
19. S. Wolfram, *Cellular Automata and Complexity, Reading* (Addison Wesley, MA, 1994)
20. A.I. Adamatzky, Computation of Shortest path in cellular automata. Math. Comput. Modelling **23**(4), 105–113 (1996)
21. K. Charalampous, A. Amanatiadis, A. Gasteratos, Efficient Robot Path Planning in the presence of dynamically expanding obstacles, in *10th International conference on Cellular Automata for Research and Industry (ACRI)*, Santorini island, Greece, 2012
22. I. Georgoudas, G.C. Sirakoulis, E.M. Skordilis, I. Andreadis, A cellular automaton simulation tool for modelling seismicity in the region of Xanthi. Environ. Model Softw. **22**(10), 1455–1464 (2007)
23. I. Georgoudas, P. Kyriakos, G.C. Sirakoulis, I. Andreadis, An FPGA implemented cellular automaton crowd evacuation model inspired by the electrostatic-induced potential fields. Microprocess. Microsyst. **34**(7–8), 285–300 (2010)
24. I. Karafyllidis, A model for the prediction of oil slick movement and spreading using cellular automata. Environ. Int. **23**(6), 839–850 (1997)
25. I. Karafyllidis, A. Thanailakis, A model for predicting forest fire spreading using cellular automata. Ecol. Model. **99**, 87–97 (1997)
26. C. Mizas, G.C. Sirakoulis, V. Mardiris, I. Karafyllidis, N. Glykos, R. Sandaltzopoulos, Reconstruction of DNA sequences using genetic algorithms and cellular automata: towards mutation prediction? Biosystems **92**(1), 61–68 (2008)
27. G.C. Sirakoulis, I. Karafyllidis, A. Thanailakis, A cellular automaton model for the effect of population movement on epidemic propagation. Ecol. Model. **133**(3), 209–223 (2000)
28. M.-A. Tsompanas, G.C. Sirakoulis, Modeling and hardware implementation of an amoeba-like cellular automaton. Bioinspir. Biomim. **7**, 036013 (2012)
29. I. Vourkas, G.C. Sirakoulis, FPGA based cellular automata for environmental modeling, in *19th IEEE International Conf. Electronics, Circuits, and Systems (ICECS)*, Seville, Spain, 2012
30. P. Progias, G.C. Sirakoulis, An FPGA processor for modelling wildfire spread. Math. Comput. Model. **57**(5–6), 1436–1452 (2013)
31. J. von Neumann, *Theory of self-reproducing automata, Urbana* (University of Illinois, IL, 1966)
32. S. Golzari, M.R. Meybodi, A maze routing algorithm based on two dimensional cellular automata, in *7th International Conference of Cellular Automata for Research and Industry (ACRI)*, Perpignan, France, 2006
33. M. Itoh, L.O. Chua, Memristor cellular automata and memristor discrete-time cellular neural networks. Int. J. Bifurcat. Chaos **19**(11), 3605–3656 (2009)
34. I. Vourkas, G.C. Sirakoulis, A novel design and modeling paradigm for memristor-based crossbar circuits. IEEE Trans. Nanotechnol. **11**(6), 1151–1159 (2012)

35. Easy Java Simulations (EJS) (Online). Available: http://fem.um.es/Ejs/. Accessed 2014
36. D. Stathis, I. Vourkas, G.C. Sirakoulis, Shortest path computing using memristor-based circuits and cellular automata, in *11th International Conference on Cellular Automata for Research and Industry (ACRI)*, Krakow, Poland, 2014
37. J.W. Moon, L. Moser, On cliques in graphs. Isr. J. Math. **3**(1), 23–28 (1965)
38. A. Ben-Dor, R. Shamir, Z. Yakhini, Clustering gene expression patterns. J. Comput. Biol. **6** (3–4), 281–297 (1999)
39. J. Cong, M.L. Smith, A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design, in *30th International Design Automation Conference*, New York, NY, 2003
40. V. Spirin and L.A. Mirny, Protein complexes and functional modules in molecular networks. Proc. Nat. Acad. Sci. U.S.A. **100**(21), 12123–12128 (2003)
41. E. Balas, C.S. Yu, Finding a maximum clique in an arbitrary graph. SIAM J. Comput. **15**(4), 1054–1068 (1986)
42. Y.S. Reddy, Solving max-clique using cellular neural network, in *9th International Workshop on Cellular Neural Networks and Their Applications (CNNA)*, Hsinchu, Taiwan, 2005
43. D.E. Knuth, The art of computer programming, 2nd ed., vol. 3. Sorting and Searching, Reading, MA: Addison-Wesley (1998)
44. J.L. Gordillo, J. V. Luna, Parallel sort on a linear array of cellular automata, in *IEEE International Conference on Systems, Man, and Cybernetics, Humans, Information and Technology*, San Antonio, TX, 1994
45. I. Vourkas, D. Stathis, G.C. Sirakoulis, Memristor-based parallel sorting approach using one-dimensional cellular automata. IET Electron. Lett. **50**(24), 1819–1821 (2014)
46. R. Lewis, A general-purpose hill-climbing method for order independent minimum grouping problems: a case study in graph colouring and bin packing. Comput. Oper. Res. **36**(7), 2295–2310 (2009)
47. E.G. Coffman Jr, J. Csirik, G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: survey and classification, in *Handbook of Combinatorial Optimization*, ed. by P.M. Pardalos, D. Du, R.L. Graham (Springer International Publishing, New York, NY, 2013), pp. 455–531
48. D.S. Johnson, Fast algorithms for bin packing. J. Comput. Syst. Sci. **8**(3), 272–314 (1974)
49. D. Stathis, I. Vourkas, G.C. Sirakoulis, Solving AI problems with memristors: a case study for optimal “bin packing”, in *18th Panhellenic Conference on Informatics (PCI)*, Athens, Greece, 2014
50. A. Scholl, R. Klein, Bin packing, problem description, solving procedures, and benchmark data sets, (Online). Available: http://www.wiwi.uni-jena.de/entscheidung/binpp/index.htm. Accessed 15 Mar 2015
51. G. Dosa, The tight bound of first fit decreasing bin-packing algorithm is FFD(I) $\le$ 11/9OPT (I) + 6/9, in *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies,* ed. B. Chen, M. Paterson, G. Zhang. Lecture Notes in Computer Science, vol. 4614 (Springer, 2007), pp. 1–11
52. H. Kellerer, U. Pferschy, D. Pisinger, in *Knapsack Problems* (Springer, Berlin, 2004)
53. S. Martello, P. Toth, *Knapsack Problems: Algorithms and Computer Implementation, New York* (John Wiley and Sons, NY, 1990)
54. G.B. Dantzig, Discrete-variable extremum problems. Oper. Res. **5**(2), 266–288 (1957)
55. J. Burkardt, sets of data associated with specific problems or subjects, (Online). Available: http://people.sc.fsu.edu/∼jburkardt/datasets. Accessed 1 June 2014