Universitat
de les Illes Balears

DOCTORAL THESIS
2022

ON THE USE OF STOCHASTIC LOGIC FOR NON-LINEAR CIRCUITS AND SYSTEMS

Oscar Vicente Camps Pascual

Universitat
de les Illes Balears

DOCTORAL THESIS
2022

Doctoral Programme in Electronic Engineering

ON THE USE OF STOCHASTIC LOGIC FOR NON-LINEAR CIRCUITS AND SYSTEMS

Oscar Vicente Camps Pascual

Thesis Supervisor: Prof. Rodrigo Picos Gayà
Thesis Supervisor: Prof. Stavros G. Stavrinides
Thesis tutor: Prof. Rodrigo Picos Gayà

Doctor by the Universitat de les Illes Balears

WE, THE UNDERSIGNED, DECLARE:

That the thesis entitled *On the use of stochastic logic for nonlinear circuits and systems*, presented by Oscar Camps Pascual to obtain a doctoral degree the has been completed under the supervision of Prof. Rodrigo Picos (University of Balearic Islands, Spain) and Prof. Stavros Stavrinides (International Hellenic University, Greece).

For all intents and purposes, we hereby sign this document:

MSc Eng. Oscar Vicente Camps Pascual

Prof. Rodrigo Picos

Prof. Stavros Stavrinides

Palma de Mallorca,

Acknowledgement

To my mentor, and however my friend Rodrigo, for being there in the ups and downs. For not giving up. This wouldn't have been possible without you.
To Stavros, for all the improvements and good advices. Meeting you has been one of the most valuable facts of this project.
To my parents Margalida and Vicente, to my chosen family, Victor, Ines and their daughter Victoria, to all my friends, especially Pep and Miquel...I want to thank all of them for their patience, their support, the contributions, the jokes and pranks that have made this journey worthy.
To George and my chosen nephews, Ariadna and Gerard, for lending me their fathers for this enterprise.
To my cat, for the snuggle time to reduce the stress caused by my mentor, my committee, and all my professors.
To all my pets. This wouldn't have been so funny without them.

Outline

The increasing pervasion of devices related to wireless networks, data process and transport and in general the Internet of Things (IoT) has led to a resultant rapid increase of edge devices. The numbers are enormous and the predictions are wondrous [1]; in a few years (by 2025) 180 ZBytes will be the amount of data to be handled (International Data Corporation - IDC). In addition, IoT devices will exceed 150 billions and it is estimated that the data produced by them will be about 70% of the data produced worldwide (IDC) [1]. It is apparent that all the types of centralized processing, even in the form of cloud, cannot properly and efficiently support this new computing landscape; taking also into account the fact that IoT has to go hand by hand with other technologies regarding artificial intelligence, big data, mobile computing etc., which are being referred to as ubiquitous computing platforms. Therefore, edge computing rises in the horizon, calling for data processing at the edge of the network.

All these technologies call for innovative approaches [2, 3] and some of those approaches are approximate computing [4–6], deep learning [7], new post-CMOS devices and architectures [8], or advanced processing techniques [9,10]. One of the fields where more computational power is required, is communications, especially when data privacy protection and security are included. Thus, data encryption emerges as a mandatory element. Nowadays, many techniques and approaches are proposed in the context of the IoT, in all hierarchical levels of data communications, others for ensuring privacy [11] and others for practically ensuring security [12]. As a result, there exist numerous proposals in this sense, usually ubiquitous ones; one such promising option seems to lean towards using chaotic-based encoder-decoder schemes to secure and/or authenticate data transmission in general [13]. There are examples of such secure-communication systems, analog [14, 15], and digital [16,17] ones demonstrating merits like low-cost, circuit simplicity, low-power operation etc. [9, 10, 18, 19].

On the other hand, it seems that approximate computing enables high power savings [6], making this technique a viable candidate for IoT edge devices. This framework offers energy savings by trading accuracy for energy. There are a handful of methods that can successfully implement approximate computing: programming methods and algorithms, hardware imple-

mentations and other ubiquitous solutions. As a curious note, this strongly reminds of the way chaos was encountered by Lorenz, finding different solutions of the homonym set of equations because of truncated number storage [20].

An interesting approach had been already introduced by Von Neumann in 1956 [21], based on a series of lectures given by R.S. Pierce in 1952 at California Institute of Technology. This approach, namely Stochastic Computing (SC) or Stochastic Logic, makes a trade-off between calculation time and accuracy. This approach has been successfully applied in fields as diverse as neural network implementation [22, 23], data mining [24], data compression [25], or mathematical calculations (FFT) [26], control [27], or even A/D conversion [28], among others. An important advantage is that it allows for a high reduction in the number of components, thus reducing the power required to run the circuit. However, this comes to a price, since the time required to perform the operation also increases exponentially with the number of bits.

There are cases where this trade-off plays a crucial role, like in the case of chaotic systems. It is known that key features of all nonlinear, chaotic systems are: long-term bounded aperiodic behavior, enhanced sensitivity to parameters and initial conditions, and fast de-correlation between past and present [29]. These features, especially sensitivity to initial conditions and parameter values, make proper implementation of chaotic systems within approximate computing frameworks, difficult, not impossible though. A successful example is the case of implementing a chaotic oscillator in a purely digital environment [17], but with the cost of creating a much more complicated (higher-dimensional) implementation.

In this thesis, we have focused on the use of Stochastic Computing (SC) applied to the solution of several issues. As a first step, we've evaluated the performance of SC when implementing nonlinear circuits with chaotic behavior. Specifically, in chapter 2 we have implemented the so-called Shimizu-Morioka system in SC. The results, partially published in [30]) have shown that this implementation can be useful, if it's done with a limited number of bits with parallel implementations.

In chapter 3, we present three different implementations of memristors and memristor-based systems based on SC. The first one is a purely digital memristor based on a flux-charge model. This part of the chapter was partially published in [31]. The second proposal, partially published in [32], includes the implementation of a switched capacitor memristive emulator. Finally, at the last part of the chapter regarding the third presented implementation, we propose an improvement to the previous emulator by adding SC. This last part of the chapter was partially published in [33], [34] and [35].

In the next chapter, we propose a few applications developed with memristor emulators and SC. At the first part of the chapter 4, we design and implement a system for solving mazes, using the memristive emulator as a

delay element. We initially checked the system's operation in Matlab and then we exported it to two different FPGAs. This part of the chapter was partially published in [36]. The end of chapter 4 includes a proposal for the use of SC in designing and implementing Cellular Nonlinear Networks (CNN). By combining Matlab and a FPGA, we develop a CNN and apply it to three real-time processes (Store, edge detection and image sharpening) for both grey and color images. This part of the chapter was partially published in [37] and [38].

Finally, the thesis ends with a concluding chapter, where next to briefly describing the presented work, we discuss about the value and The efficiency of using SC in several cases, especially for edge computing applications.

Resumen

La creciente penetración de los dispositivos relacionados con el procesamiento y el transporte de datos, las redes inalámbricas, y, en general, el Internet de las cosas (IoT) ha dado lugar a un rápido aumento de los dispositivos edge. Los números son enormes y las predicciones maravillosas [1]; en unos años (para el 2025) 180 ZBytes será la cantidad de datos a manejar (International Data Corporation - IDC). Además, los dispositivos IoT superarán los 150 mil millones y se estima que los datos producidos por ellos serán alrededor del 70% de los datos producidos a nivel mundial (IDC) [1]. Es evidente que todos los tipos de procesamiento centralizado, incluso en forma de nube, no pueden admitir de manera adecuada y eficiente este nuevo panorama informático; teniendo también en cuenta que IoT tiene que ir de la mano de otras tecnologías en materia de inteligencia artificial, big data, computación móvil, etc., a las que se está denominando plataformas de computación ubicua. Por lo tanto, la computación edge se eleva en el horizonte y exige el procesamiento de datos en el perímetro de la red.

Todas estas tecnologías requieren enfoques innovadores [2,3] y algunos de esos enfoques son computación aproximada [4–6], aprendizaje profundo [7], nuevos dispositivos y arquitecturas post-CMOS [8], o técnicas de procesamiento avanzadas [9, 10]. Uno de los campos donde más potencia computacional se requiere es el de las comunicaciones, especialmente cuando se incluye la protección de la privacidad y la seguridad de los datos. Así, el cifrado de datos surge como un elemento obligatorio. Hoy en día, se proponen muchas técnicas y enfoques en el contexto del IoT, en todos los niveles jerárquicos de las comunicaciones de datos, otros para garantizar la privacidad [11] y otros para garantizar prácticamente la seguridad [12]. En consecuencia, existen numerosas propuestas en este sentido, generalmente ubicuas; una de esas opciones prometedoras parece inclinarse hacia el uso de esquemas de codificador-decodificador basados en caos para asegurar y/o autenticar la transmisión de datos en general [13]. Hay ejemplos de tales sistemas de comunicación segura, analógicos [14, 15] y digitales [16, 17] que demuestran méritos como bajo costo, simplicidad de circuitos, operación de bajo consumo, etc. [9, 10, 18, 19].

Por otro lado, parece que la computación aproximada permite un alto ahorro de energía [6], lo que convierte a esta técnica en un candidato viable

para los dispositivos IoT edge. Este marco ofrece ahorros de energía al intercambiar precisión por energía. Hay un puñado de métodos que pueden implementar con éxito la computación aproximada: métodos y algoritmos de programación, implementaciones de hardware y otras soluciones ubicuas. Como nota curiosa, esto recuerda fuertemente la forma en que Lorenz encontró el caos, encontrando diferentes soluciones al conjunto homónimo de ecuaciones debido al almacenamiento de números truncados [20].

Von Neumann ya había introducido un enfoque interesante en 1956 [21], basado en una serie de conferencias dadas por R.S. Pierce en 1952 en el Instituto de Tecnología de California. Este enfoque, a saber, Computación estocástica (SC) o Lógica estocástica, hace un compromiso entre el tiempo de cálculo y la precisión. Este enfoque se ha aplicado con éxito en campos tan diversos como la implementación de redes neuronales [22, 23], minería de datos [24], compresión de datos [25] o cálculos matemáticos (FFT) [26], control [27], o incluso conversión A/D [28], entre otros. Indicativo de sus ventajas, permite una gran reducción en el número de componentes, reduciendo así la potencia necesaria para hacer funcionar el circuito. Sin embargo, esto tiene un precio, ya que el tiempo necesario para realizar la operación también aumenta exponencialmente con el número de bits.

Hay casos en los que esta compensación juega un papel crucial, como en el caso de los sistemas caóticos. Se sabe que las características clave de todos los sistemas caóticos no lineales son: comportamiento aperiódico acotado a largo plazo, mayor sensibilidad a los parámetros y condiciones iniciales, y rápida descorrelación entre el pasado y el presente [29]. Estas características, especialmente la sensibilidad a las condiciones iniciales y los valores de los parámetros, hacen que la implementación adecuada de sistemas caóticos dentro de marcos informáticos aproximados sea difícil, aunque no imposible. Un ejemplo exitoso es el caso de implementar un oscilador caótico en un entorno puramente digital [17], pero con el costo de crear una implementación mucho más complicada (de mayor dimensión).

En esta tesis, nos hemos centrado en el uso de la computación estocástica (CE) aplicada a la resolución de diferentes problemas. Como primer paso, hemos evaluado la eficacia de la CE a la hora de implementar circuitos no lineales con comportamiento caótico. Específicamente, en el capítulo 2 hemos implementado el llamado sistema de Shimizu-Morioka en CE. Los resultados (publicados en parte en [30]) han mostrado que esta implementación puede ser útil, si se hace usando un número limitado de bits, con implementaciones en paralelo.

En el capítulo 3, hemos presentado tres implementaciones de memristores y sistemas memristivos basados en CE. El primero de los tres es un emulador de memristor digital basado en el modelo de flujo-carga. Esta parte del capítulo se basa en el artículo publicado en [31].

La segunda propuesta, publicada parcialmente en [32], incluye la implementación de un emulador de memristor utilizando capacidades conmutadas.

Finalmente, en la última parte del capítulo con respecto a la tercera implementación presentada, proponemos una mejora al emulador anterior, añadiéndole CE.

Esta última parte del capítulo se publicó parcialmente en [33], [34] y [35].

En el siguiente capítulo, proponemos algunas aplicaciones desarrolladas con emuladores de memristores y CE. En la primera parte del capítulo 4, realizaremos el diseño e implementación de un sistema para resolver laberintos, utilizando el emulador memristivo como elemento de retardo.

Inicialmente comprobamos la operación del sistema en Matlab y a continuación lo exportamos a dos FPGAs diferentes. Esta parte del capítulo fue publicada parcialmente en [36]. El final del capítulo 4 incluye una propuesta para el uso de la CE en el diseño e implementación de Redes Celulares No-Lineales (Cellular Nonlinear Networks - CNN). Combinando Matlab y una FPGA, desarrollaremos una CNN para aplicar a tres procesos (Almacenado, detección de bordes y definición de imagen) en tiempo real a imágenes en escala de grises y en color. Esta parte del capítulo fue publicada parcialmente en [37] y [38].

Finalmente, acabamos la tesis con un capítulo de conclusiones, en el cual discutiremos sobre el valor y eficiencia de utilizar CE para diferentes casos, especialmente para aplicaciones de computación en el perímetro de la red.

Resum

La creixent penetració de dispositius relacionats amb les xarxes sense fils, el processament i el transport de dades i, en general, l'Internet de les coses (*Internet of Things*, IoT) ha donat lloc a un ràpid augment de la quantitat de dispositius de computació en el llindar (edge computing). Els números són enormes i les prediccions són espectaculars [1]; d'aquí a uns anys (el 2025) 180 ZBytes serà la quantitat de dades que s'haurà de gestionar (International Data Corporation - IDC). A més, els dispositius IoT superaran els 150.000 milions i s'estima que les dades produïdes per ells seran al voltant del 70% de les dades produïdes a tot el món (IDC) [1]. És evident que tots els tipus de processament centralitzat, fins i tot en forma de núvol, no poden suportar de manera adequada i eficient aquest nou panorama informàtic; tenint en compte també el fet que l'IoT ha d'anar de la mà d'altres tecnologies en matèria d'intel·ligència artificial, big data, informàtica mòbil, etc. que s'estan denominant plataformes d'informàtica ubiqua. Per tant, la computació edge s'alça a l'horitzó, demanant el processament de dades a la vora de la xarxa.

Totes aquestes tecnologies requereixen enfocaments innovadors [2,3] i alguns d'aquests enfocaments són computació aproximada [4–6], aprenentatge profund [7], nous dispositius i arquitectures post-CMOS [8], o tècniques avançades de processament [9, 10]. Un dels camps on es requereix més potència computacional és el de les comunicacions, especialment quan s'han d'incloure la protecció i la seguretat de la privacitat de les dades. Així, el xifratge de dades sorgeix com un element obligatori. Actualment, en el context de l'IoT es proposen moltes tècniques i enfocaments, en tots els nivells jeràrquics de comunicacions de dades, d'altres per garantir la privacitat [11] i d'altres per garantir pràcticament la seguretat [12]. En conseqüència, existeixen nombroses propostes en aquest sentit, generalment omnipresents; una d'aquestes opcions prometedores sembla inclinar-se cap a l'ús d'esquemes de codificador-descodificador basats en caos per assegurar i/o autenticar la transmissió de dades en general [13]. Hi ha exemples d'aquests sistemes de comunicació segura, [14, 15] i digitals [16, 17] que demostren mèrits com el baix cost, la simplicitat del circuit, el funcionament de baix consum, etc. [9, 10, 18, 19].

D'altra banda, sembla que la informàtica aproximada permet un gran

estalvi d'energia [6], fent d'aquesta tècnica un candidat viable per a dispositius edge IoT. Aquest marc ofereix estalvi energètic mitjançant l'intercanvi de precisió per energia. Hi ha un grapat de mètodes que poden implementar amb èxit la informàtica aproximada: mètodes i algorismes de programació, implementacions de hardware i altres solucions omnipresents. Com a nota curiosa, això recorda fortament la manera com es va trobar el caos per Lorenz, trobant diferents solucions al conjunt homònim d'equacions a causa de l'emmagatzematge de nombres truncats [20].

Un enfocament interessant ja havia estat introduït per Von Neumann el 1956 [21], basat en una sèrie de conferències impartides per R.S. Pierce el 1952 a l'Institut Tecnològic de Califòrnia. Aquest enfocament, és a dir, la informàtica estocàstica (SC) o la lògica estocàstica, fa una compensació entre el temps de càlcul i la precisió. Aquest enfocament s'ha aplicat amb èxit en camps tan diversos com la implementació de xarxes neuronals [22,23], mineria de dades [24], compressió de dades [25] o càlculs matemàtics (FFT) [26], control [27], o fins i tot conversió A/D [28], entre d'altres. Indicatiu dels seus avantatges, permet una gran reducció del nombre de components, reduint així la potència necessària per fer funcionar el circuit. Tanmateix, això té un preu, ja que el temps necessari per realitzar l'operació també augmenta exponencialment amb el nombre de bits.

Hi ha casos en què aquesta compensació juga un paper crucial, com en el cas dels sistemes caòtics. Se sap que les característiques clau de tots els sistemes no lineals i caòtics són: comportament aperiòdic limitat a llarg termini, sensibilitat millorada als paràmetres i condicions inicials i una ràpida descorrelació entre el passat i el present [29]. Aquestes característiques, especialment la sensibilitat a les condicions inicials i els valors dels paràmetres, fan que la implementació adequada de sistemes caòtics dins de marcs informàtics aproximats sigui difícil, però no impossible. Un exemple d'èxit és el cas d'implementar un oscil·lador caòtic en un entorn purament digital [17], però amb el cost de crear una implementació molt més complicada (de dimensions superiors).

En aquesta tesi, ens hem centrat en l'ús de la computació estocàstica (CE) a diferents problemes. Com a primer pas, hem avaluat l'eficàcia de la CE a l'hora d'implementar circuits no lineals amb comportament caòtic. Específicament, al capítol 2 hem implementat l'anomenat sistema de Shimizu-Morioka en CE. Els resultats (publicats en part a [30]) han mostrat que aquesta implementació pot ser útil, si es fa usant un nombre limitat de bits, amb implementacions en paral·lel.

Al capítol 3, hem presentat tres implementacions de memristors i sistemes memristius basats en CE. El primer dels tres és un emulador de memristor digital basat en el model de fluxe-càrrega. Aquesta part del capítol es basa en l'article publicat a [31]. La segona proposta, publicada parcialment a [32], inclou la implementació d'un emulador de memristor usant capacitats commutades.

Finalment, a la darrera part del capítol pel que fa a la tercera implementació presentada, proposam una millora a l'emulador anterior, afegint-hi CE. Aquesta darrera part del capítol es va publicar parcialment a [33], [34] i [35].

Al següent capítol, proposam algunes aplicacions desenvolupades amb emuladors de memristors i CE. A la primera part del capítol 4, dissenyarem i implementarem un sistema per resoldre laberints, fent servir l'emulador memristiu com a element de retard. Vàrem començar comprovant l'operació del sistema amb Matlab i a continuació el vàrem exportar a dues FPGAs diferents. Aquesta part del capítol va ser publicada parcialment a [36]. El final del capítol 4 inclou una proposta per l'ús de la CE per dissenyar i implementar Xarxes Cel·lulars No-lineals (Cellular Nonlinear Networks - CNN). Combinant Matlab i una FPGA, desenvoluparem una CNN per aplicar a tres processos (Enmagatzament, detecció de llindars i definició d'imatge) en temps real a imatges en escala de grisos i en color. Aquesta part del capítol va ser publicada parcialment a [37] i [38].

Finalment, acabam la tesi amb un capítol de conclusions, on discutim sobre el valor i l'eficiència d'usar CE per diferents casos, especialment per aplicacions de computació al perímetre de la xarxa.

Acronyms

The following abbreviations are used in this thesis:

AI :	Artificial Intelligence
B2S :	Binary to Stochastic
BEN :	Binary Encoded Number(s)
DRM :	Dynamic Route Map
ENB :	Effective Number of Bits
FFT :	Fast Fourier Transform
FPGA :	Field Programmable Gate Array
IoT :	Internet of Things
JTAG :	Joint Test Action Group
LSB :	Least Significant Bit
LUT :	Look Up Table
MSB :	Most Significant Bit
NF :	Noise Figure
ODE :	Ordinary Differential Equation
PDF :	Probability Density Function
RBG :	Random Bit Generator
RNG :	Random Number Generator
SC :	Stochastic Computing
SCN :	Stochastic Computing Number
SEN :	Stochastic Encoded Number(s)
SL :	Stochastic Logic
UDP :	User Datagram Protocol
UM :	Universal Machines

Contents

List of Publications

List of publications related to the thesis:

Journals:

1. **O. Camps**, M. M. Al Chawa, S. G. Stavrinides, and R. Picos, "Stochastic computing emulation of memristor cellular nonlinear networks," Micromachines, vol. 13, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2072-666X/13/1/67

2. C. de Benito, **O. Camps**, M. M. Al Chawa, S. G. Stavrinides, and R. Picos, "A switched capacitor memristor emulator using stochastic computing," Technologies, vol. 10, no. 2, p. 39, 2022.

3. P. Dopazo, C. de Benito, **O. Camps**, S. G. Stavrinides, and R. Picos, "Gerard: General rapid resolution of digital mazes using a memristor emulator," Physics, vol. 4, no. 1, pp. 1–11, 2021.

4. **O. Camps**, S. G. Stavrinides, and R. Picos, "Stochastic computing implementation of chaotic systems," Mathematics, vol. 9, no. 4, 2021. [Online]. Available: https://www.mdpi.com/2227-7390/9/4/375

5. G. Svetoslavov, **O. Camps**, S. G. Stavrinides, and R. Picos, "A switched capacitor memristive emulator," IEEE Transactions on Circuits and Systems II: Express Briefs, 2020.

Conferences:

1. C. de Benito, **O. Camps**, M. Al Chawa, S. Stavrinides, and R. Picos, "A stochastic switched capacitor memristor emulator," in 2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST). IEEE, 2021, pp. 1–4.

2. **O. Camps**, S. G. Stavrinides, and R. Picos, "Efficient implementation of memristor cellular nonlinear networks using stochastic computing,"

in 2020 European Conference on Circuit Theory and Design (ECCTD). IEEE, 2020, pp. 1–4

3. **O. Camps**, R. Picos, C. de Benito, M. M. Al Chawa, and S. G. Stavrinides, "Effective accuracy estimation and representation error reduction for stochastic logic operations," in 2018 7th Int. Conf. on Modern Circuits and Systems Technologies (MOCAST). IEEE, 2018

4. **O. Camps**, M. M. Al Chawa, C. de Benito, M. Roca, S. G. Stavrinides, R. Picos, and L. O. Chua, "A purely digital memristor emulator based on a flux-charge model," in 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS). IEEE, 2018, pp. 565–568

5. **O. Camps**, R. Picos, C. de Benito, M. M. Al Chawa, and S. G. Stavrinides, "Emulating memristors in a digital environment using stochastic logic," in 2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST). IEEE, 2018, pp. 1–4.

Chapter 1

Introduction

1.1 Background

The increasing pervasion of devices related to wireless networks, data process and transport and in general the Internet of Things (IoT) has led to a resultant rapid increase of edge devices. The numbers are enormous and the predictions are wondrous [1]; in a few years (by 2025) 180 ZBytes will be the amount of data to be handled (International Data Corporation - IDC). In addition, IoT devices will exceed 150 billions and it is estimated that the data produced by them will be about 70% of the data produced worldwide (IDC) [1]. It is apparent that all the types of centralized processing, even in the form of cloud, cannot properly and efficiently support this new computing landscape; taking also into account the fact that IoT has to go hand by hand with other technologies regarding artificial intelligence, big data, mobile computing etc. which are being referred to as ubiquitous computing platforms. Therefore, edge computing rises in the horizon, calling for data processing at the edge of the network.

All these technologies call for innovative approaches [2, 3] and some of those approaches are approximate computing [4–6], deep learning [7], new post-CMOS devices and architectures [8], or advanced processing techniques [9,10]. One of the fields where more computational power is required, is communications, especially when data privacy protection and security are included. Thus, data encryption emerges as a mandatory element. Nowadays, many techniques and approaches are proposed in the context of the IoT, in all hierarchical levels of data communications, others for ensuring privacy [11] and others for practically ensuring security [12]. As a result, there exist numerous proposals in this sense, usually ubiquitous ones; one such promising option seems to lean towards using chaotic-based encoder-decoder schemes to secure and/or authenticate data transmission in general [13, 39–41]. There are examples of such secure-communication systems, analog [14, 15], and digital [16, 17] ones demonstrating merits like low-cost,

circuit simplicity, low-power operation etc. [9, 10, 18, 19].

On the other hand, it seems that approximate computing enables high power savings [6], making this technique a viable candidate for IoT edge devices. This framework offers energy savings by trading accuracy for energy. There are a handful of methods that can successfully implement approximate computing: programming methods and algorithms, hardware implementations and other ubiquitous solutions. As a curious side note, this strongly reminds of the way chaos was encountered by Lorenz, finding different solutions of his equations because of truncated number storage [20]. In fact, as the legend goes, Lorenz found that saving the state of a simulation and using it as the new initial condition, resulted in a very different simulation than simply continuing this without saving. After a lot of analysis effort, it was found that results were stored with less resolution than was used internally to perform the computations. This difference in the number of bits caused minuscule differences that led to divergent simulations because of the chaotic nature of the system. Thus, it is quite important to analyze if the system to be implemented allows the use of approximate computing.

An interesting approach had been already introduced by Von Neumann in 1956 [21], based on a series of lectures given by R.S. Pierce in 1952 at California Institute of Technology. This approach, namely Stochastic Computing (SC) or Stochastic Logic, makes a trade off between calculation time and accuracy. This approach has been successfully applied in fields as diverse as neural network implementation [22, 23], data mining [24], data compression [25], or mathematical calculations (FFT) [26], control [27], or even A/D conversion [28], among others. Indicative of its advantages [42], it allows for a high reduction in the number of components, thus reducing the power required to run the circuit. However, this comes to a price, since the time required to perform the operation also increases exponentially with the number of bits. This, in turn, may increase the total energy consumption when the number of bits exceeds 16-17 [43, 44]. There are, however, techniques that allow this problem to be alleviated, and make this competitive even for higher bit-numbers [44]. Notice, that stochasticity is entering only in the proper (for calculation) number generation, and has nothing to do with the systems implemented within this framework. In fact, it could be considered as an equivalent of typical noisy system.

There are cases where this trade-off plays crucial role, like in the case of chaotic systems. It is known that key features of all nonlinear, chaotic systems are: long-term bounded aperiodic behavior, enhanced sensitivity to parameters and initial conditions, and fast de-correlation between past and present [29]. These features, especially sensitivity to initial conditions and parameter values, make proper implementation of chaotic systems within approximate computing frameworks, difficult, not impossible though. A successful example is the case of implementing a chaotic oscillator in a purely digital environment [17], but with the cost of creating a much more compli-

cated (higher-dimensional) implementation.

In this chapter, we will rate the goodness of this trade off. In section 1.4 we will explain some fundamentals about errors calculations and propagation of errors in stochastic computing. Later, in section 1.4.2 we will make an approach to effective accuracy estimation. Finally, in section 1.4.3, we will discuss about ways to reduce the error in stochastic computing.

1.2 Approximate computing

If we stop for a moment, we will realize that we are surrounded by approximations in many fields. In engineering, we can find approximate signal processing in audio and video. In communications systems, network protocols like UDP over IP are based on a best effort policy. In mathematics, we can usually find function approximations, and in computer science, we use very often approximate string matching and approximation algorithms. The two main motivations to bet for an AC are the power and the reliability, and we can not ignore any of both. To be honest, most of the computation we use nowadays are being performed either on portable devices (Mostly mobile phones) or in massive data centers. In the case of mobile phones, it would be interesting if we can make the battery last longer and, in the case of data centers, power consumption is the main operational cost.

AC is really suitable when we can take advantage of applications with high embedded error tolerance. This tolerance is really high in situations in which users can accept a lack of accuracy, as the phone example mentioned before, or in situations in which the average human senses are limited, as in multimedia applications, like raw image format versus .jpg format, or flac file sound format versus mp3 files. Finally, error tolerance is very high when the golden result does not exist, it's not possible, or it's too expensive, like in data mining.

And with respect to applications, AC is interesting for those iterative and convergent ones that need to treat massive data and the number of iterations marks the quality of the results. AC is also very effective for applications with more than a right answer, like machine learning and web search. Finally, we consider that AC is appropriated for the applications that operate in noisy data with analog inputs and, as told, for applications with analog outputs for human users.

1.3 Stochastic Computing

1.3.1 Basic operations

The SC approach adopts real number representation by strings of N random binary numbers b_i. The probability of "1" bits to appear within the bit-

string is proportional to the number to be operated [45]:

$$p = \frac{1}{N}\sum_{i}^{N} b_i \tag{1.1}$$

These strings are called Stochastic Computing numbers (SCN) or Stochastic Encoded Numbers (SEN). In this thesis, we will opt for the second, using also the term Binary Encoded Numbers (BEN) for those encoded as classical binary numbers. Notice that the number of bits that can be encoded in a chain of length N is $log_2 N$, since the relevant information is just the number of "1"s . For instance, the chains[1] "0000 0000 0001 1111 0000 1100" and "0100 0000 0111 1001 0000 0001" would both encode the information that there are 7 ones in a chain which is 24 bits long.

There are two main ways to generate a map between a SCN and real numbers: first, we can map the desired range of real numbers to the real domain [0..1]; second, we can map them to the interval [-1..1]. Depending on which mapping is to be implemented, many different mathematical operations can then be done using simple logic gates or simple sequential circuits.

As an example, multiplication of SCN is performed using a simple AND gate when using the [0..1] domain. Alternatively, considering the [-1..1] domain, the same multiplicative operation requires the use of an XNOR gate, as shown in Fig. 1.1(a).

Since we cannot represent any SC number as a probability higher than one, the case of addition becomes slightly more complex, since 1+1=2. Thus, the operation that should be implemented is $(x + y)/2$, which would always return a maximum value of 1. This operation is usually implemented using a multiplexer, as shown in Fig. 1.1(b), where the p(0.5) means a signal with a probability of 50% to be '1' or '0'. This necessary input signal is generated using one of the bits generated in the RNG, so no additional circuitry is needed. It is worth pointing out that this gate is the same in both the [0..1] and the [-1..1] domains. Other more complex operations (division [27], square roots [27], reversible gates [46], etc...) are also discussed in the literature, though not presented in this thesis.

Another important point is the conversion from BEN to SEN. This is usually achieved by using a scheme similar to that in Fig. 1.2, where an N-bit random number is generated by utilizing a random number generator (RNG) and compared to the value of the N-bit BEN. If the output of the RNG is below the BEN, the converter's output would be bit "1", bit "0" otherwise. In the opposite operation, converting SEN back to its BEN representation, the number of 1's included in the signal needs to be calculated; something that can be straightforwardly achieved by a simple counter.

[1] In the above chains, the spaces are added just to facilitate reading.

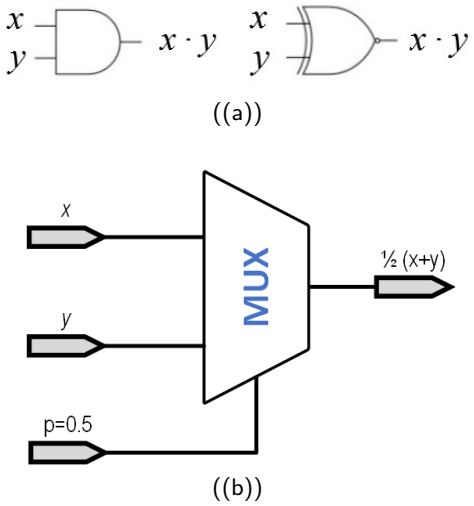


Figure 1.1: Basic implementation of basic operation in SC. (a) Basic implementation scheme of a SC multiplier in the (0..1) range (AND gate, left) and in the (−1..1) range (XNOR gate, right). (b) Basic implementation scheme of a SC adder using a multiplexer.

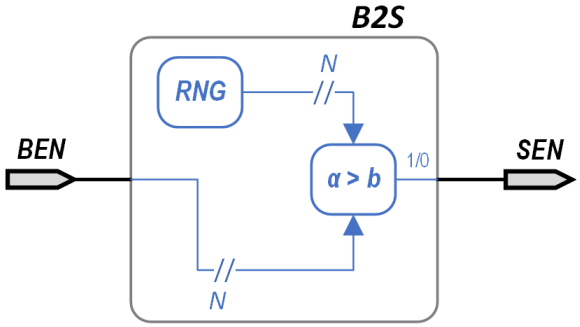


Figure 1.2: Basic implementation scheme of a Binary Encoded Number (BEN) to a Stochastic Encoded Number (SEN), using a Random Number Generator (RNG).

1.3.2 Division and Square Root Configurations

In this section we will discuss the implementation proposed by Gaines in the 1960 [47], since it forms the basis for all the posterior proposals (for instance, [48, 49]).

In this approach, a single JK flip-flop is used to generate an approximation to the division result. As it is known, when the two inputs of the JK are 01 or 10, the state of the flip-flop (Q^+) is set to 0 and 1, respectively. J=0 and K=0 doesn't change the state ($Q^+ = Q$) and J=1 and K=1 toggles

the state of the JK ($Q^+ = \bar{Q}$). If we define p_{ij} as the probabilities of JK=ij, we can see that the probability of the state p_{Q^+} is:

$$p_{Q^+} = p_{00} \cdot p_Q + p_{10} + p_{11} \cdot (1 - p_Q) \tag{1.2}$$

If the flip-flop is in steady-state, $p_Z = p_{Q^+} = p_Q$, so $p_Z = \frac{p_{10}+p_{11}}{1-p_{00}+p_{11}}$. Where

$$\begin{aligned} p_{X_1} = p_J = p_{10} + p_{11} \\ p_{X_2} = p_K = p_{01} + p_{11} \\ p_{00} + p_{01} + p_{10} + p_{11} = 1 \end{aligned} \tag{1.3}$$

Thus, the probability of the output Z being 1 is: $p_Z = \frac{p_{X_1}}{p_{X_1}+p_{X_2}}$

This way, if the dividend p_{X_1} is small, we can approximate the division operation $p_Z = \frac{p_{X_1}}{p_{X_2}}$. If the divisor p_{X_2} is small, the result becomes very inaccurate. Gaines proposed adding a sequential component, called ADDIE (ADaptative DIgital Element), in order to implement $p_Z = \frac{p_{X_1}}{p_{X_2}}$. The ADDIE can be implemented by using an up-down counter with feedback and makes an estimation of $\hat{p_Z}$ in binary form. To generate the stochastic form of $\hat{p_Z}$, it uses a SNG. As we know, $\hat{p_Z} \cong p_Z = \frac{p_{X_1}}{p_{X_2}}$, so $p_{X_1} = p_{X_2} \cdot \hat{p_Z}$.

In Fig. 1.3 we show the scheme of divider based on ADDIE for the [0..1] domain, which can also be named as the unipolar version. This one calculates $\hat{p_Z}$ in a dynamic way. If p_{X_1} is lower than $p_{X_2} \cdot \hat{p_Z}$, then p_z is greater than $\hat{p_Z}$ and the counter gets increased. If $p_{X_2} \cdot \hat{p_Z}$ is greater than p_{X_1}, then $\frac{p_{X_1}}{p_{X_2}}$ is lower than $\hat{p_Z}$ and the counter gets decreased.

The output of the counter is connected to an AND gate in order to perform $p_{X_2} \cdot \hat{p_Z}$. The result of this AND gate is connected to the Down input of the counter, feeding it back. The binary form of p_{X_2} (Dividend) is connected to the Up input of the counter.

The division $\hat{p_Z} = \frac{p_{x_1}}{p_{x_2}}$ is completed once the system is in equilibrium, which is when the counter output p_z is such that $p_{X_1} = p_{X_2} \cdot \hat{p_Z}$, which means that, in average, the output of the counter will not change.

The variance of the output of the counter (p_z or, in its stochastic representation, $\hat{p}_z$) was noted by Gaines to be inversely proportional to the the number of possible states of the counter (2^k). The reason for this is that p_Z changes value by $\Delta p_z = \frac{1}{2^k}$ when the Up and Down signals are different. Thus, this Δp_z defines an error boundary for his proposed divider.

In addition, since the variance due to the stochastic noise in a combinational SC circuits is $\sigma = \frac{p_Z \cdot (1-p_Z)}{N}$ (where N is the bit-stream length [50]), Z can present an arbitrarily small variance given an arbitrarily long bit-stream and an arbitrary length counter.

In figure 1.4 we can see the design of the ADDIE-based divider for the [-1..1] domain, also called the bipolar version. A few variations must be

made to Fig. 1.3 to get $2p_Z - 1 = \frac{2p_{X_1}}{2p_{X_2}}$. We must note that the bipolar stochastic numbers may be negative, so $X_1 > X_2 \cdot \hat{Z}$ does not guarantee that $\hat{Z} < X_1/X_2$. The comparison between $\hat{Z}$ X_1/X_2 is affected by the sign of X_2. On the same way, $X_1 < X_2 \cdot \hat{Z}$ does not imply that $\hat{Z} > X_1/X_2$. By comparing $X_1 \cdot X_2$ and $X_2^2 \cdot \hat{Z}$ instead of X_1 and $X_2 \cdot \hat{Z}$, we can decrease the uncertainty that is caused by the signed numbers. If $X_2 \neq 0$, $\rightarrow X_2^2 > 0$, so $X_1 \cdot X_2 > X_2^2 \cdot \hat{Z} \rightarrow \hat{Z} < X_1/X_2$ and $\hat{Z} > X_1/X_2$ if $X_1 \cdot X_2 < X_2^2 \cdot \hat{Z}$

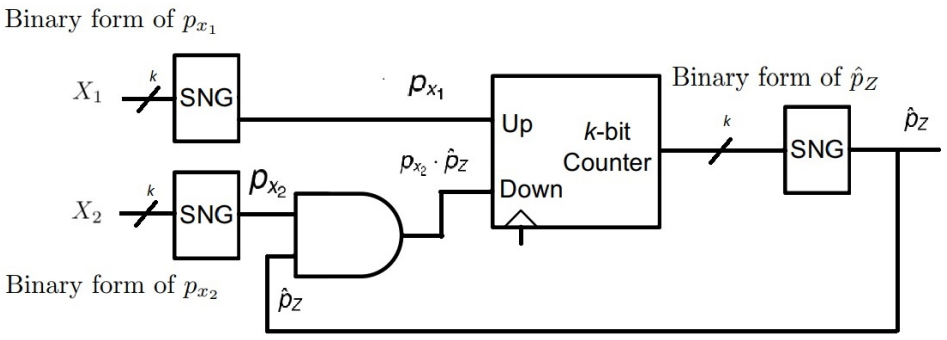


Figure 1.3: Division scheme for the [0..1] domain, as in [47].

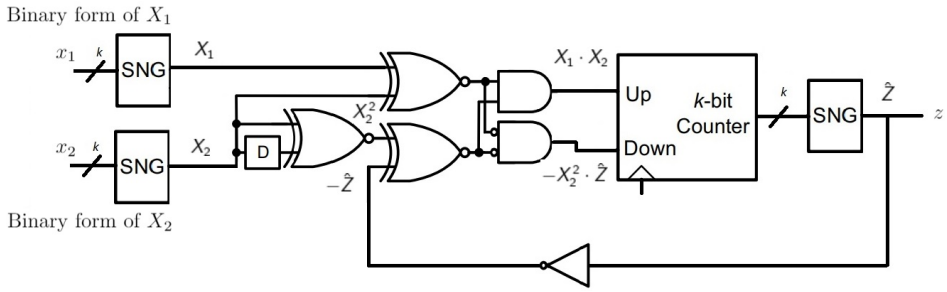


Figure 1.4: Division scheme for the [-1..1] domain, as in [47].

The information in this section has been gathered from [51], [48] and [49].

1.3.3 Arbitrary Function Implementation

A very interesting approach to approximate arbitrary functions using Stochastic computing was proposed in [52]. To achieve this target, they propose the use of the Taylor expansion, after dividing the function input range in a series of segments.

A few previous assumptions are needed. The first one is that $f(x)$ is a function continuosly differentiable for an input range [a,b]. The range [a,b] can be split into N sub-intervals of the same length. This way, every length

of the sub-intervals will be $h = |a - b|/N$. Each interval can be denoted as $[a_{i-1}, a_i]$, with $i = 1, ..., N$ as the index of each one of them.

When using the unipolar format, both the input and output range in stochastic logic are [0,1]. The existing method only discuss functions with these range's conditions. The main issue in this method is to approach every output segment as a linear function $y = a_i x + b_i$, with i as every segment's identifier.

By using the Taylor series expansion, $f(x)$ can expanded at $x = x_0$ this way:

$$f(x) = f(x_0) + f'(x_0)\Delta x + \frac{f''(x_0)}{2}(\Delta x)^2 + O((\Delta x)^3) \tag{1.4}$$

with $\Delta x = x - x_0$.

For an arbitrary input x, as long as the interval number (index i) is determined, Eq. (1.4) with only the linear term can be used for approximating $f(x)$. The corresponding value of index i can be determined by (1.5)

$$i = \lfloor x/h \rfloor + 1 \tag{1.5}$$

$$q = \frac{x}{h} - p \tag{1.6}$$

where $\lfloor x/h \rfloor$ is the integer part of x/h and is denoted by p.

1.3.3.1 Linear Approximation

The above Taylor approximation has been used in [52] by truncating it to the first order, thus obtaining a linear approximation. This way, (1.4) can be expressed by

$$f^*(x) = \lambda_i + \mu_i \cdot q \tag{1.7}$$

where the values of λ_i, q and $|\mu_i|$ belong to $[0, 1]$. In addition, λ_i and $|\mu_i|$ depend on the specific interval i. Eq. (1.7) is a basic approximation equation that cannot be implemented with unipolar stochastic logic directly, because the value of μ can be negative when $f((p + 1)h) < f(ph)$. Thus, the proposed method requires a specific study on the monotonicity of $f(x)$ in $[0, 1]$ to determine whether the value of μ is positive or not. Thus, some implementations of a full system according to the monotonicity of the target function in $[0, 1]$ are proposed by [52], and shown in Figures 1.5 and 1.6, where $\lambda - GU$ and $\phi - GU$ are the λ and ϕ generating unit, respectively.

1.3.3.2 Proposed Quadratic Approximation Scheme

The above system, though effective, seems to be fairly complex due to the use of unipolar logic, which requires a quite large overhead in terms of control

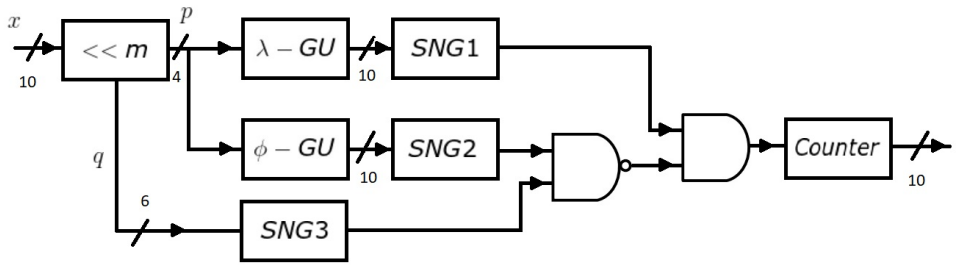


Figure 1.5: Existing scheme [52] for monotonically increasing functions.

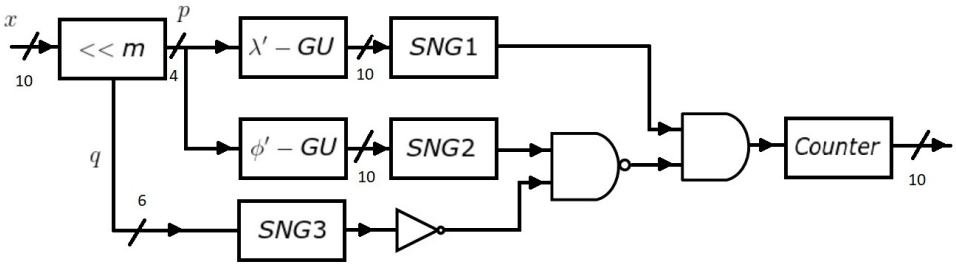


Figure 1.6: Existing scheme [52] for monotonically decreasing functions.

logic. It also presents the additional problem of needing to find the points where the function can change slope (minima and maxima), since there a linear approximation is bound to be inaccurate.

Both these problems can be solved by using the bipolar representation (x ∈ [-1,1]), and using a quadratic approximation, as in Eq. (1.4). Using the same notation than before, we can write:

$$f^*(x) = a_0 + a_1 \cdot q + a_2 \cdot q^2 \tag{1.8}$$

The overhead for this implementation is nearly null, since it requires only a delay element (a D register, for instance), two additional multiplications (AND gates), and an additional addition (OR gate plus a multiplexer).

In order to remove the selectors required to determine the current segment, we assume that we are using $N = 2^{N_S}$ segments. This way, the values of the different coefficients can be stored in a memory, which is addressed using the N_S most significant bits of the input x, which are also used to calculate q.

The proposed scheme is shown in Fig. 1.7 where a_0, a_1 and a_2 are the corresponding values of λ_i, μ_i and η_i at each interval. The notation $\hat{a}$ is used for the stochastic numbers representation, while a is the digital value stored into the memory cells.

It has to be noted that the criterion to define the range's approximation

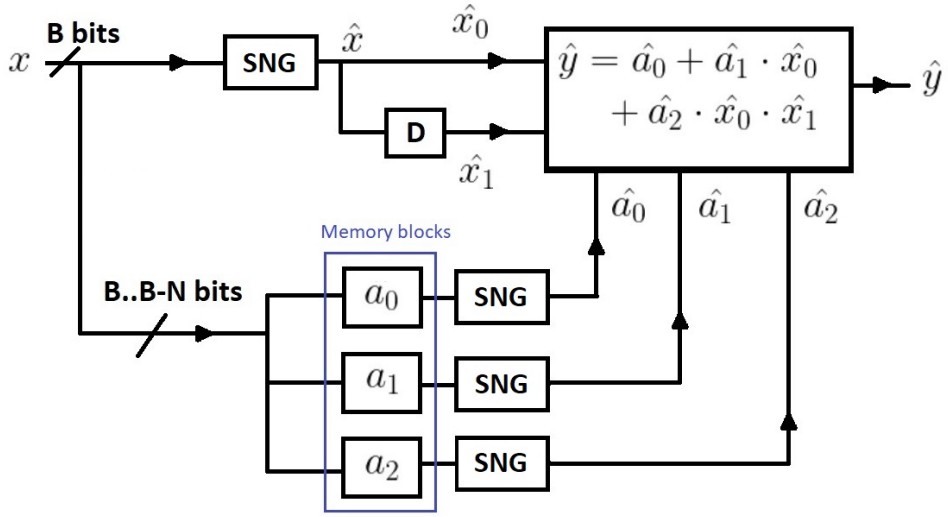


Figure 1.7: Proposed scheme with memory blocs. The actual implementation of the additions and multiplication inside the box is not explicitly shown, since they are basic stochastic operations.

is that the slope of the first and second derivative must be between -1 and 1. The method implementation is as follows. In an off-line, one-time process, we split the interval [-1..1] in N segments, where $N = 2^{N_S}$ as said above. Then, we approach the arbitrary function to a quadratic function $y = a_0 + a_1 * x + a_2 * x^2$. The coefficients a_0, a_1, a_2 may be different for each interval, and they must be calculated beforehand and stored. Since N is a power of two, these coefficient values are stored into a memory using the N most significant bits of 'x' as the address, thus significantly simplifying the recovery process compared to previous proposals, since there is no need to make a comparison to determine the corresponding segment. Notice that this forces the number of bits B of the input to be higher than N_S.

To estimate the error, we evaluate the difference between the approached value and the real value and we perform the RMS on that difference. It is clear that, depending on the specific function to be approximated, the number of segments, and the number of bits, the error will be dominated either by the intrinsic stochastic error or by the goodness of the approximation using a quadratic function inside the given segments.

To check the effectiveness of the proposed algorithm, we have tested it with many different functions. In Table 1.1 we have calculated the RMS error between the real function and the approximation made by using 32 segments and 20 bits for some of the proposed functions.

As an example, figure 1.9 shows function $y = 0.1 \cdot cos(6 \cdot x) \cdot log(x + 2)$ when using 20 and 22 bits for 32 segments. The RMS errors were 1.80E-10 and 1.79E-10, respectively.

Function	Error	Figure
$y = 0.1 \cdot sin(18 \cdot x) \cdot x^2$	3.49E-09	Fig. 1.8(a)
$y = exp(x)$	3.44E-12	Fig. 1.8(b)
$y = 0.1 \cdot exp(x/4) \cdot cos(12 \cdot x) \cdot x^2$	5.77E-09	Fig. 1.8(c)
$y = -0.25 + 1/(6 + 4 \cdot x)$	2.48E-11	Fig. 1.8(d)
$y = 0.1 \cdot sin(\pi \cdot x)$	6.86E-12	Fig. 1.8(e)
$y = 0.4 \cdot x^2 - 0.25$	1.38E-12	Fig. 1.8(f)
$y = 0.1 + 0.5/(2 + x) - 0.2 \cdot x$	2.01E-12	Fig. 1.8(g)
$y = cos(x)$	1.47E-12	Fig. 1.8(h)
$y = 0.25 \cdot (x - 0.2) \cdot (x + 0.5) \cdot x$	4.27E-12	Fig. 1.8(i)

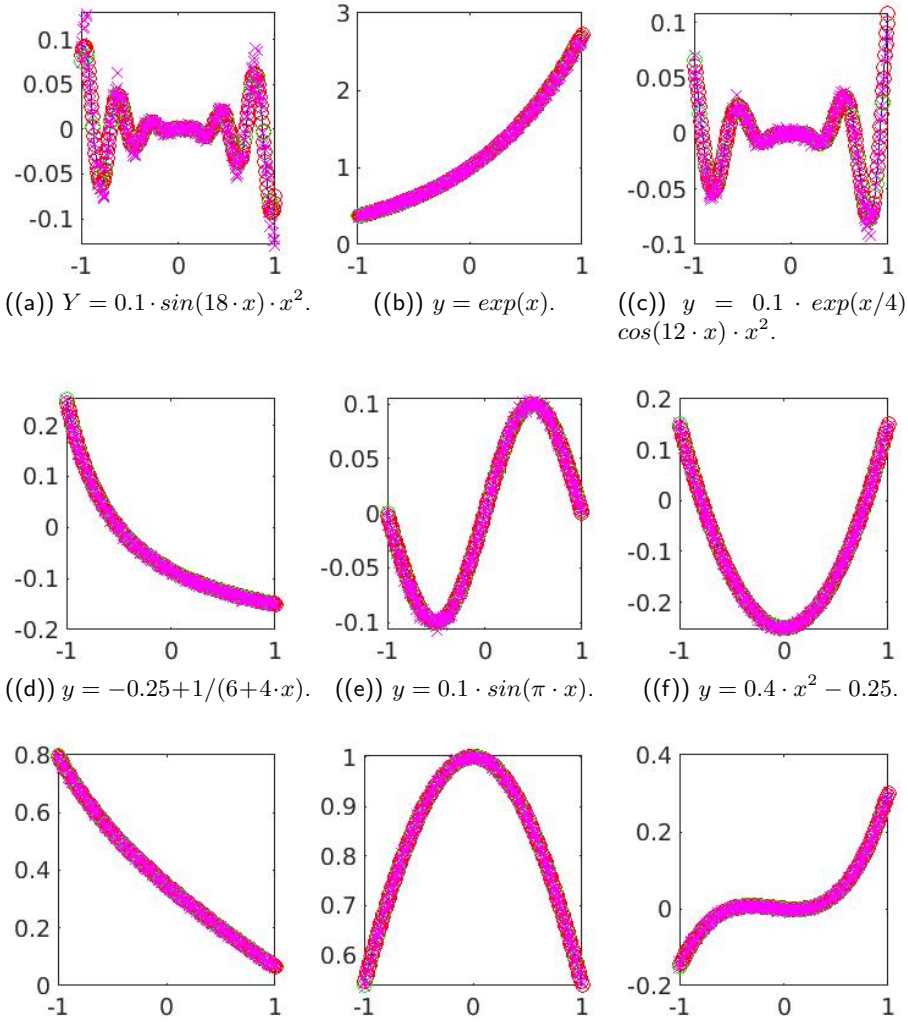
Table 1.1: Arbitrary function examples and RMS error in the interval x ∈ [-1,1] using 20 bits and 32 segments.

Number of bits	RMS error FN_A	RMS error FN_B
8	2.08E-06	1.76E-06
10	1.28E-07	1.23E-07
12	8.57E-09	1.05E-08
14	9.93E-10	3.90E-09
18	5.33E-10	3.50E-09
20	5.30E-10	3.49E-09
22	5.30E-10	3.49E-09
24	5.30E-10	3.49E-09

Table 1.2: RMS errors for function $FN_A = 0.1 \cdot exp(x/4) \cdot cos(12 \cdot x) \cdot x^2$ and $FN_B = 0.1 \cdot sin(18 \cdot x) \cdot x^2$ for several number of bits and 32 segments. The symbols correspond to the calculated values of the function. Boundaries between segments are marked in green.

Number of segments	Error FN_A	Error FN_B
8	8.85E-06	1.35E-04
16	3.64E-07	1.92E-06
32	5.77E-09	3.80E-08
64	9.16E-11	6.25E-10

Table 1.3: RMS errors for function $FN_A = 0.1 \cdot exp(x/4) \cdot cos(12 \cdot x) \cdot x^2$ and $FN_B = 0.1 \cdot sin(18 \cdot x) \cdot x^2$ according to 20 bits and several number of segments.



((a)) $Y = 0.1 \cdot sin(18 \cdot x) \cdot x^2$.

((b)) $y = exp(x)$.

((c)) $y = 0.1 \cdot exp(x/4) \cdot cos(12 \cdot x) \cdot x^2$.

((d)) $y = -0.25+1/(6+4 \cdot x)$.

((e)) $y = 0.1 \cdot sin(\pi \cdot x)$.

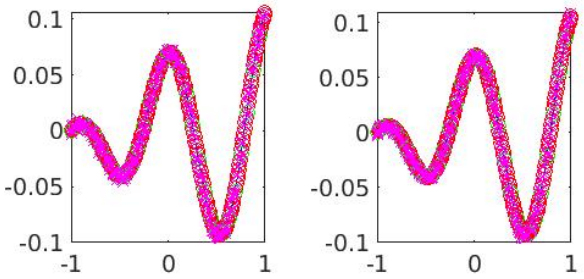
((f)) $y = 0.4 \cdot x^2 - 0.25$.

((g)) $y = 0.1 + 0.5/(2 + x) - 0.2 \cdot x$.

((h)) $y = cos(x)$.

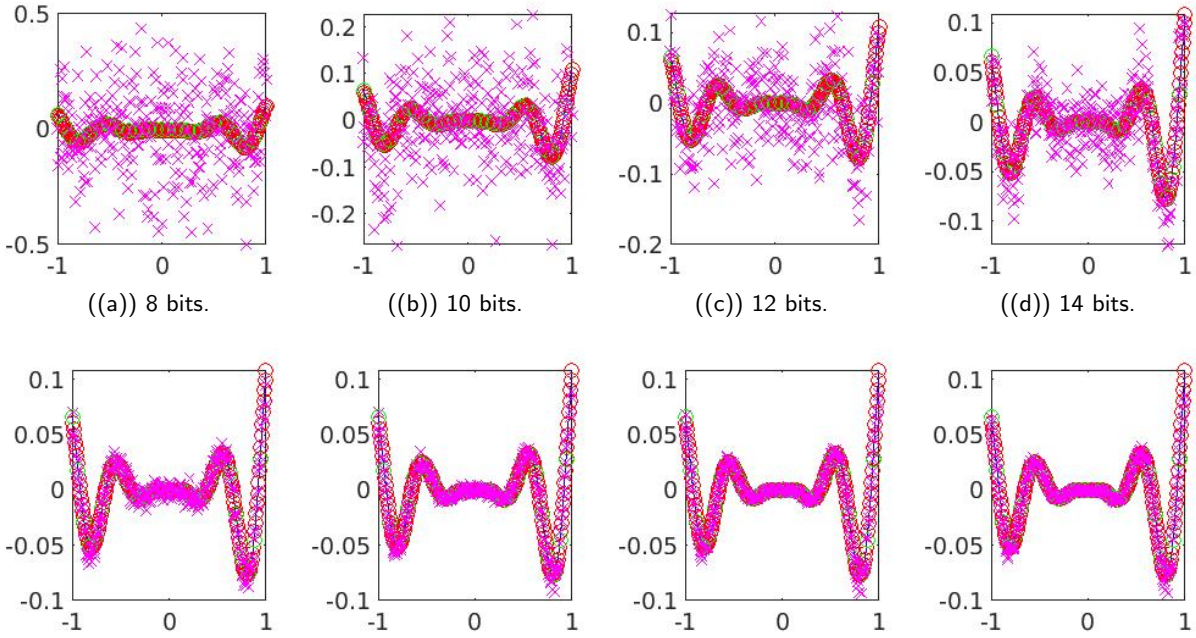
((i)) $y = 0.25 \cdot (x - 0.2) \cdot (x + 0.5) \cdot x$.

Figure 1.8: Arbitrary functions with 32 segments and 20 bits. The symbols correspond to the calculated values of the function. Boundaries between segments are marked in green.



((a)) 20 bits. ((b)) 22 bits.

Figure 1.9: Comparison between using 20 and 22 bits, respectively for function $y = 0.1 \cdot cos(6 \cdot x) \cdot log(x + 2)$ and 32 segments. The symbols correspond to the calculated values of the function. Boundaries between segments are marked in green.



((a)) 8 bits. ((b)) 10 bits. ((c)) 12 bits. ((d)) 14 bits.

((e)) 18 bits. ((f)) 20 bits. ((g)) 22 bits. ((h)) 24 bits.

Figure 1.10: Comparison of the effect of the used number of bits for function $y = 0.1 \cdot exp(x/4) \cdot cos(12 \cdot x) \cdot x^2$ for 32 segments. The symbols correspond to the calculated values of the function. Boundaries between segments are marked in green.

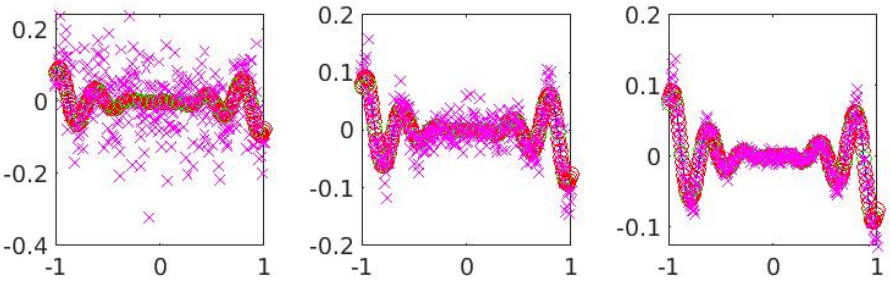


Figure 1.11: Comparison of the effect of changing the number of bits for function $Y = 0.1 \cdot sin(18 \cdot x) \cdot x^2$ for 32 segments. The symbols correspond to the calculated values of the function. Boundaries between segments are marked in green.

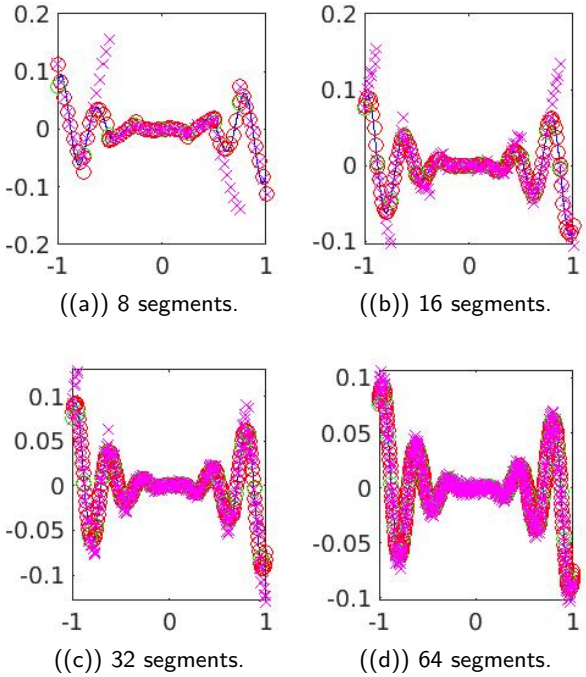


Figure 1.12: Comparison of the effect of changing the number of segments for function $y = 0.1 \cdot sin(18 \cdot x) \cdot x^2$ using N=20 bits. The symbols correspond to the calculated values of the function. Boundaries between segments are marked in green.

A more extended example is shown in Figure 1.10, where $y = 0.1 \cdot exp(x/4) \cdot cos(12 \cdot x) \cdot x^2$ is plot along its approximation by using several number of bits (from 8 to 24 bits). Table 1.2 shows the corresponding RMS errors according to each number of bits. The same evaluation is performed for $y = 0.1 * sin(18 * x) * x^2$ in figure 1.11. Table 1.2 also shows the RMS errors according to each number of bits. The behavior of the RMS error for both function approximations is similar, decreasing exponentially with an increasing number of bits, as expected, until a plateau is reached. This plateau is related to the error intrinsic to the stochastic computing method. It propagates to the final value in a way related to the specific function, which can be calculated using standard error propagation techniques [53].

Those same two functions were also used to evaluate the effect of changing the number of segments. Figure 1.12 depicts the function $y = 0.1 \cdot sin(18 \cdot x) \cdot x^2$ along with the approximations using several number of segments. Table 1.3 shows the RMS errors according to each number of segments. In this case, the behavior is similar to the effect of the number of bits, in the sense that the RMS error seems to decrease with the exponential of the logarithm of the number of segments. In this case, and due to computational limitations, we have not reached a plateau, even if we can theorize there may exist one.

1.3.4 Intrinsic Noise Estimation

Another important point is the conversion from BEN to SEN. This is usually achieved by using a scheme similar to that in Figure 1.2 (b), where an N-bit random number is generated by utilizing a random number generator (RNG) and compared to the value of the N-bit BEN. If the output of the RNG is below the BEN, the converter's output would be a 1-bit "1", or a 1-bit "0" otherwise. In the opposite operation, converting SEN back to its BEN representation, the number of 1's included in the signal needs to be calculated; something that can be achieved by a simple counter.

It is apparent that the error in the approximation of the SEN to its actual value is equivalent to the error provided by a random walk process of length n, and thus proportional to $\sqrt{n}$, as it has been discussed in the literature [50]. Therefore, using N bits, we may consider that all the noise caused by the process is included in the lowest $N/2$ bits. This way, the noise figure NF for a signal of power S_p with noise power N_p caused by the use of the SCN is:

$$NF = 10log_{10}\left(\frac{S_p}{N_p}\right) = 10log_{10}\left(\frac{2^N}{2^{N/2}}\right) \approx 3.01N/2 \; dB \qquad (1.9)$$

Notice that for this equation we have considered the maximum possible amplitude for the input signal. In order to consider the minimum amplitude

over the noise, we consider we are 1 bit over the noise $(N/2+1)$. In this case, we obtain:

$$NF = 10log_{10}\left(\frac{S_p}{N_p}\right) = 10log_{10}\left(\frac{2^{N/2+1}}{2^{N/2}}\right) \approx 3.01\ dB \tag{1.10}$$

Thus, the system is expected to have a NF between 3dB and 3 $(N/2+1)$dB, assuming as above that we use more than 1 bit. This NF sets the required number of bits, which is related to the sensitivity of the equation system to noise. Empirically, we have seen that linear equations allow for a low N, while nonlinear systems call for higher values. Notice that a value of the $NF = 20$ dB calls for N=12 bits, while N=32 bits would provide $NF = 54$ dB.

1.4 Intrinsic Error in Stochastic Computing

As it is known, in digital logic there are only two different possible values (zero and one), at any node, at any time. Thus, when one considers random processes in digital logic, it's only natural to use the Bernoulli formalism. Considering the case of a random experiment/process with two possible outputs α_1 and α_2, with respective probabilities p and q, belonging in (0, 1): the α_1-result can be tagged as a success while the α_2-result as a failure. Then, a random variable X is defined as one:

$$X : \Omega \to \mathbb{R} \tag{1.11}$$

where Ω is the sample space, given as $X(\alpha_1) = 1$ and $X(\alpha_1) = 0$. In this case, the probability function of X will be $p(X = 1) = p$, $p(X = 0) = p$, and we can tag X as Bernoulli's random variable with p-parameter. The expected or mean value of X is:

$$E(X) = p \tag{1.12}$$

and its variance $\sigma(X)$ is:

$$\sigma(X) = p \cdot q \tag{1.13}$$

If one considers n-independent-repetitions of a Bernoulli's type experiment, then the random variable X will be called a Binomial Random Variable with parameters n and p. In this case the expected or mean value is:

$$E(X) = n \cdot p \tag{1.14}$$

and its variance :

$$\sigma(X) = n \cdot p \cdot q \tag{1.15}$$

1.4.1 Error Propagation

By utilizing the calculations in equations (1.14) and (1.15) the error introduced by performing certain operations is discussed. It is apparent that the intrinsic randomness of the representation method leads to a certain numerical inaccuracy, that can be interpreted as noise. In specific, the error performed when doing an addition and a multiplication will be estimated, in the lines that follow.

1.4.1.1 Expected value and Variance of the sum

If X and Y are two uncorrelated random variables, then the expected value of their summation $Z = X + Y$ will be:

$$\begin{aligned} E(Z) = E(X+Y) = \int\int (x'+y') f_{X,Y}(x',y') dx'dy' = \\ = \int x' f_X(x') dx' + \int y' f_Y(y') dy' = \\ = E[X] + E[Y] \end{aligned} \tag{1.16}$$

Where function f is the Probability Density Function (PDF) of the random variables.

According to equation (1.16), if $Z = X+Y$ then $E[Z] = E[X+Y])E[X]+E[Y]$ and the corresponding variance σ of Z is calculated according to (1.17):

$$\begin{aligned} \sigma(Z) = E[(Z - E[Z])^2] = E[(X + Y - E[X] - E[Y])^2] = \\ = E[\{(X - E[X])^2 + (Y - E[Y])^2\}] = \\ = \sigma[X] + \sigma[Y] \end{aligned} \tag{1.17}$$

1.4.1.2 Expected value and Variance of the product

If X and Y are two independent random variables, the expected value of their product $E[Z] = E[X \cdot Y]$ will be:

$$\begin{aligned} E[Z] = E[X \cdot Y] = \int\int x'y' f_X(x') f_Y(y') dx'dy' = \\ = \left\{ \int x' f_X(x') dx' \right\} \cdot \left\{ \int y' f_Y(y') dy' \right\} = \\ = E[X] \cdot E[Y] \end{aligned} \tag{1.18}$$

Like in the case of addition, according to equation (1.18), if $Z = X * Y$ then $E[Z] = E[X \cdot Y] = E[X] \cdot E[Y]$ and the corresponding variance σ of Z is calculated according to (1.19):

$$\begin{aligned} \sigma(Z) = E[X]^2 \cdot \sigma[Y] + E[Y]^2 \cdot \sigma[X] + \sigma[X] \cdot \sigma[Y] = \\ = E[X^2] \cdot E[Y^2] - E[X]^2 \cdot E[Y]^2 \end{aligned} \tag{1.19}$$

Details on how to calculate the above derivations can be found in [47], [54] and [55].

1.4.2 Effective accuracy estimation

To calculate the expected results of these operations and the expected deviation, we can assume that we are operating with a binomial distribution. For such a distribution, the probability of an '1s' is considered to be p, and the probability of '0s' is considered to be q ($q = 1 - p$). Then we can calculate the mean (μ) and the standard deviation (σ) as:

$$\mu = n \cdot p \tag{1.20}$$

$$\sigma = \sqrt{n \cdot p \cdot (1 - p)} \tag{1.21}$$

According to the definition of relative error ϵ (epsilon), this is equal to the ratio between the error of the actual value, and the expected value. If one wishes to have a probability of being correct higher than a 99.99%, then the error could be bounded by 4σ; thus the relative error is:

$$\epsilon(\%) = \frac{4\sigma}{\mu} = \frac{4\sqrt{n \cdot p \cdot (1-p)}}{n \cdot p} = \sqrt{\frac{1}{p} - 1} \cdot \frac{4}{\sqrt{n}} \tag{1.22}$$

And by assuming that p=0.5, then this relative error could be written as:

$$\epsilon(\%) \simeq \frac{4}{\sqrt{n}} \tag{1.23}$$

Defining the *Effective Number of Bits* (ENB) as the number of the binary digits demanded for the representation of a binary number, whose Less Significant Bit (LSB) is above the value of 4σ then the representation error of the number is below the quantification error ($p > 0.9999$) using the number of bits defined by the ENB. If we want to calculate the ENB b of a Stochastic Computing Number (SCN) of a length of n bits, we can say that:

$$n = 2^{x+b} \tag{1.24}$$

where x is the number of bits needed to represent the error:

$$2^{-x} = \epsilon = \frac{4\sigma}{\mu} = \frac{4\sqrt{n \cdot p \cdot (1-p)}}{n \cdot p} = \sqrt{\frac{1}{p} - 1} \cdot \frac{4}{\sqrt{n}} \tag{1.25}$$

Assuming again that that p=0.5, we end to the following:

$$2^{-x} = \frac{4}{\sqrt{n}} \tag{1.26}$$

and then, simple arithmetic operations lead to fact that:

$$b = x + 2 \tag{1.27}$$

Thus, the number of the needed bits to represent a SCN number with a resolution better than b bits is $b + x = 2b - 2$. As a result, according to the number of digits needed to represent the error, the ENB can vary accordingly, leading to reduced number of digits needed for the specific accuracy. For instance, according to the empirical rule described above, if an error less than 1% is demanded, then:

$$\epsilon = 0.01 = 2^{-x} = \frac{4}{\sqrt{n}} \rightarrow n = 400 < 512 = 2^9 \rightarrow x = 9 \tag{1.28}$$

According to equation (1.27) and in order to get this accuracy ($< 1\%$), the ENB demanded is calculated to be 20 bits, thus we would need a SCN with a length of 2^{20} bits.

1.4.3 Reducing error in stochastic number representation

It is imperative that the numbers used when applying the scheme of stochastic computing, must be uncorrelated one to the other. To comply with this demand, random bit generators (RBG) are utilized both in the initial stage of any mathematical chain process and the intermediate stages of stochastic computing function processes. In this chapter, partially published on [50], we propose a new method for implementing a RBG with improved characteristics, in terms of error suppression. The idea behind this new method is based on the principal of feedback. In specific, we add a feedback path on the sequence of bits, produced in the RBGs output. This feedback considers the average of the probabilities obtained until the present instance and properly applies it to the input, practically leading to an improved error performance. In 1.13, we show the block diagram of the proposed method.

Comparative simulations of the operation of a classic (RBG) and one utilizing this new method (Improved RBG), for producing uncorrelated numbers, suitable for stochastic computing operations, were run in Matlab environment. Since in stochastic computing all numbers represent probabilities, a sweep for two different probabilities (namely p=0.15 and p=0.50) was executed and the Probability Density Function (PDF) was calculated; providing

us with a measure of the error suppression achieved in the case of the *Improved RGB*. The results appear in figure 1.14, where the blue line (lower) distributions regard the case of a typical RBG and the red line (upper) distributions the case of the *Improved RBG* utilizing the principle feedback. In the cases of both probabilities the calculated PDF peak appeared to be significantly higher, when the proposed hereby new method was used. In 1.4, the calculated mean value in these cases appear, further confirming the *Improved RBG's* performance.

We ran another comparative test, aiming to provide with clues of the proposed RBG's operational capabilities, according to the number of bits of the input number in the simulation environment. In specific, we fed the generator with samples of binary words comprising of 2^6, 2^8, 2^{10}, 2^{12} and 2^{14} bits, all of them representing an expected value (mean value) of p=0.35. The corresponding results appear in figure 1.15 and in table 1.5. We can easily observe that it comes a time in which the peak grows slowly until no significant increase is observed.

Table 1.4: Calculated means in the case presented in fig 1.14

Mean	Without feedback	With feedback
$p = 0.15$	0.1562	0.1550
$p = 0.50$	0.4981	0.4969

Table 1.5: Calculated means in the case presented in fig 1.15 (Proposed method)

Samples	Mean	With feedback	Color
2^6	$p = 0.35$	0.3521	red
2^8	$p = 0.35$	0.3505	green
2^{10}	$p = 0.35$	0.3505	blue
2^{12}	$p = 0.35$	0.3498	black
2^{14}	$p = 0.35$	0.3502	yellow

1.5 Objectives and Outline

The main objective of the thesis is to study the possible application of stochastic computing (SC) to different problems, including chaotic equations and memristive systems.

Toward this, we have evaluated the performance of SC when implementing nonlinear circuits with chaotic behavior. Thus, we have started by analyzing a simple (chapter 2) , showing that it may behave correctly. After that, we have used the same circuit to create an oscillator and we have

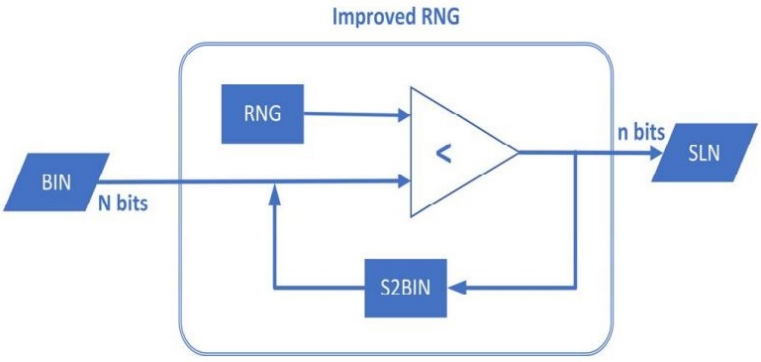


Figure 1.13: The proposed, feedback-based *Improved Random Bit Generator*, with reduced error representation.

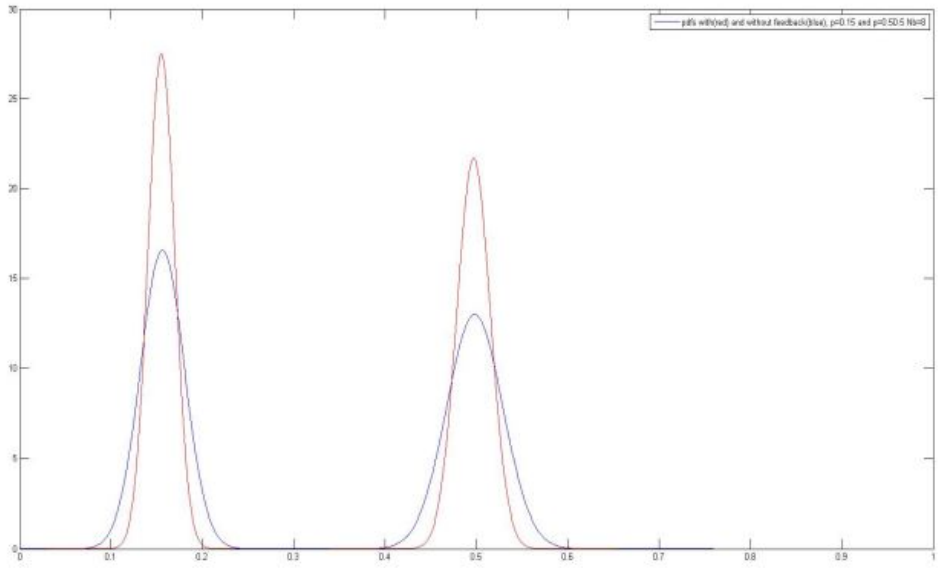


Figure 1.14: PDFs for several probabilities using the standard method without feedback (blue line) and the proposed method with feedback (red line).

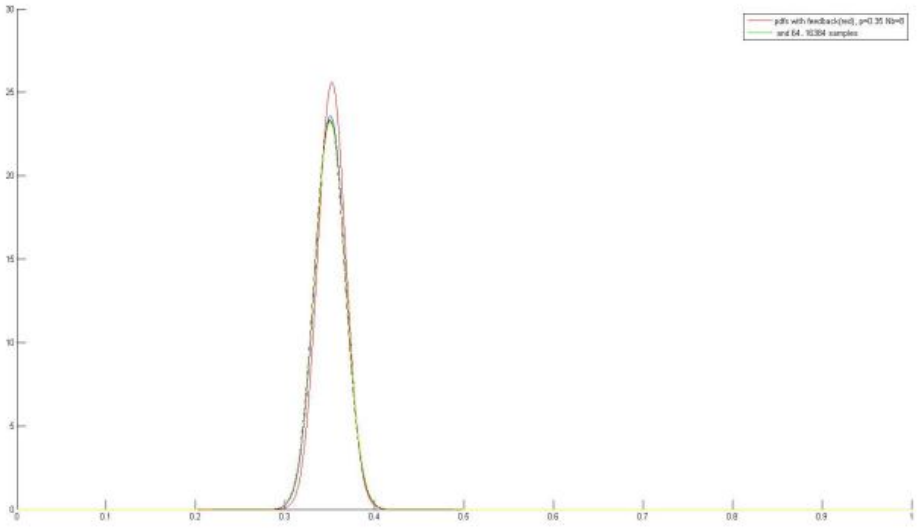


Figure 1.15: PDFs for one probability using the proposed method (with feedback) for several number of samples: 2^6, 2^8, 2^{10}, 2^{12} and 2^{14}.

checked that it also works as expected. Finally, we have implemented the so-called Shimizu-Morioka nonlinear system in SC. The results, partially published in [30]) have shown that this implementation can be useful, if it's done with a limited number of bits and a parallel implementation.

In chapter 3, we present three different implementations of memristors and memristor-based systems based on SC. The first one is a purely digital memristor based on a flux-charge model. Then, we present the implementation of a switched capacitor memristive emulator; and finally, we present the third implementation, as we propose an improvement to the previous emulator, by adding SC.

At the next chapter 4, we've treated a few applications to be developed with memristors within a SC environment. Initially, we designed and implemented a system to solve mazes, using the memristive emulator as a delay element. Before the FPGA implementation the system was checked in Matlab. Concluding chapter 4, we propose the use of SC for the design and implementation of Cellular Nonlinear Networks (CNN). By combining Matlab and a FPGA, we developed a CNN and utilized it (as a proof of concept) to three real-time processes (Store, edge detection and image sharpening) for gray and color images.

Chapter 2

Implementation of Differential Equation Systems

2.1 Introduction

In this chapter, having in focus possible utilization of chaotic systems for edge computing applications, like the IoT, an unconventional approach to the computation of nonlinear dynamical systems is investigated. In specific, we show how the stochastic computing (SC) framework, an option for approximate computing, can be utilized to properly implement nonlinear, chaotic operating systems. It is apparent that approximate computing methods for implementing systems sensitive to initial conditions and system parameters is an important issue that needs thorough investigation.

In Section 2.2 an introduction to Stochastic Computing is apposed, presenting some very basic examples; Section 2.3 presents a method in the form of an example, on how to implement a nonlinear system using both this framework and a classical integration, as well. Then, established nonlinear dynamics tools and metrics like fractal dimension, Kolmogorov-Sinai entropy etc. are utilized to estimate how well the described SC implementation behaves, compared to the conventional approach. Finally, Section 2.4 concludes the chapter.

2.2 Stochastic Computing Implementation of Analog Systems

2.2.1 Implementation of basic differential equations

Implementation of differential equations using SC is similar to the case of implementing them in discrete form. The process involves, mainly, rewriting

the equation in a specific way so they can be integrated using SC. This requires three different transformations:

1. Rewriting the equation in a form suited to SC. In the most basic case, this implies replacing all the additions by half additions: $a + b \rightarrow 2(a/2 + b/2)$. Other, more complex operations may require a harder reworking of the equations to ensure all the operations can be implemented in SC in the [-1,1] or [0,1] range. For instance, in the case of implementing a division, one has to ensure that the result is always going to be in range, which may require an additional scaling and shifting of the variables involved.

2. Scaling all the variables into the [-1..1] or [0..1] domains, since those are the values that can be dealt with in SC.

3. Then, a final transformation ensures that all the modules of the coefficients in the equations are lower than 1. This is equivalent to a time scaling.

Once these three transformations are performed, the equations may be processed as a SC system. It is apparent that in this procedure multiplications are implemented by the gates appearing in Figure 1.1(a), and additions by the half-additions implemented by the gate in Figure 1.1(b).

In order to implement the integrator the scheme appearing in Figure 2.1 is utilized. This figure shows the symbol to be used in (a), while the scheme is shown in (b). It performs a continuous integration, by counting up or down depending on whether the input is 1 or 0 and comparing the output of the counter with a random number generator RNG to create a SCN. The implementation of the integrator and the RNG implies making a decision on the effective number of bits demanded in order to define the size of the counter. Notice that this is equivalent to determining the precision of the integrator, since numbers are represented between [0..1] (or [-1..1]) with N bits of resolution and a noise figure provided by Eq. (1.9). Related to the number of RNG needed to implement this scheme, it has been shown in [56] that using the same RNG in both inputs actually improves the accuracy, thus simplifying the design.

Related to this number of bits is the issue of way time is mapped to the number of iterations, in the relevant SC equations. This is an issue strongly dependent on the integration method; assuming that a simple first order rectangular integration, with no explicit time dependence is utilized, then, it is apparent that the following equation ((2.1)) holds valid:

$$\begin{aligned} \dot{x} &= f(x) \\ \dot{x} &\approx \frac{\Delta x}{\Delta t} = f(x) \rightarrow \Delta x = f(x)\Delta t \end{aligned} \tag{2.1}$$

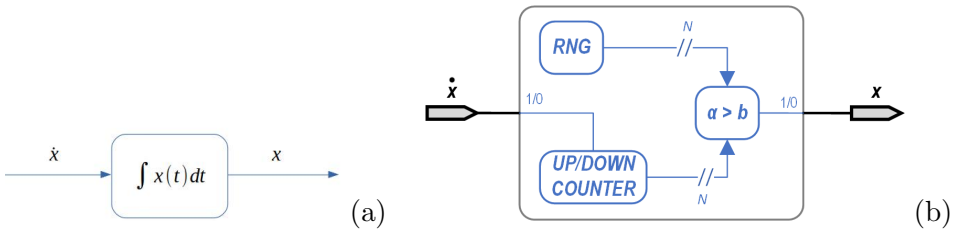


Figure 2.1: (a) Symbol for the integrator block; (b) Basic implementation scheme of a SC integrator. Notice that both the input $\dot{x}(t)$ and the output $x(t)$ are SEN numbers.

Using the integrator scheme presented in Figure 2.1, we see that allowing for a high enough number of N_{acc} iterations, the output at the counter $Int(x(t))$ would be:

$$Int(x) = \frac{N_{acc} \cdot p(1)}{2^N} \tag{2.2}$$

where $p(1)$ is the probability of having a 1 at the input. It has to be noted that $p(1)$ corresponds to the actual value to be represented in the SCN, and N_{acc} is the number of "ones" the accumulator has counted. Thus, equating Eq. (2.1) with Eq. (2.2), we find that the effective time step is given in Eq. (2.3).

$$\Delta t = \frac{N_{acc}}{2^N} \tag{2.3}$$

Notice that the integrator depicted in Figure 2.1 also includes a binary to stochastic (B2S) converter, which is simply a random number generator (RNG), plus a comparator. This way, the output of the integrator is already also a SC number that can be used further in the circuit.

2.2.2 Basic examples

As a proof of concept two basic examples are presented: a system that integrates twice a constant, and a simple oscillator. We have implemented these systems into a DE2-70 FPGA by Altera, using Quartus-II to compile it. The resulting circuits were implemented using less than 500 gates, or less than a 2% of the available number of gates. Communication with the computer was implemented using the JTAG interface, controlled within Matlab.

2.2.2.1 Integrating a constant (twice)

As a first example, we have designed the system shown in Figure 2.2 and Eq. (2.4) and initial conditions $(\ddot{x}, \dot{x}, x) = (k_2, k_1, k_0)$. The system then integrates the equations over time.

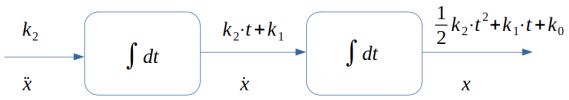


Figure 2.2: Basic implementation scheme of a second order ODE with constant input and initial conditions $(\ddot{x}, \dot{x}, x) = (k_2, k_1, k_0)$. The analytical solution is also provided at the end of each stage.

$$\begin{aligned} \dot{x} &= y \quad (x(t=0) = k_0) \\ \dot{y} &= z \quad (y(t=0) = k_1) \\ \dot{z} &= 0 \quad (z(t=0) = k_2) \end{aligned} \tag{2.4}$$

The analytical solution of the system above, is provided in Figure 2.2 and the results for a specific set of initial conditions are depicted in Figure 2.3 (for initial conditions $k_0 = k_1 = 0$, and $k_2 = 0.08$). In this figure, we have used $N_{acc} = 5000$ iterations and $N = 21$ bits, resulting into a time-step $\Delta t = \frac{5000}{2^{21}} \approx 2.384\ ms$. Using this relation, we expect the line corresponding to $\dot{x}$ reaching $\ddot{x}$ (k_2) at the 420^{th} iteration, and crossing the parabola generated by x at the 839^{th} iteration. The parabola is expected to cross k_2 at iteration 593. Actually, all these crossings emerged exactly at the expected points, as shown in Figure 2.3.

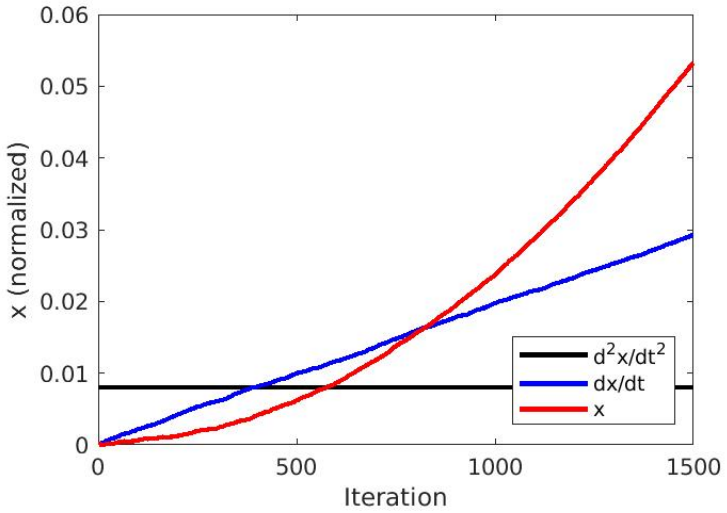


Figure 2.3: Output of the scheme in Figure 2.2, showing x, $\dot{x}$ and $\ddot{x}$. The values of the initial conditions are $k_2 = 0.08, k_1 = 0.0, k_0 = 0.0$, and $N_{acc} = 5000$, $N = 21$ bits, providing a timestep $\Delta t \approx 2.384\ ms$.

2.2.2.2 A Simple Oscillator

The simplest possible autonomous system to be implemented was that of a self-oscillating system; this is a simple coupled system described in Eq.((2.5)) implementing a simple oscillator of frequency one ($\omega = 1$), corresponding to the system illustrated in Figure 2.4.

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -x \end{aligned} \tag{2.5}$$

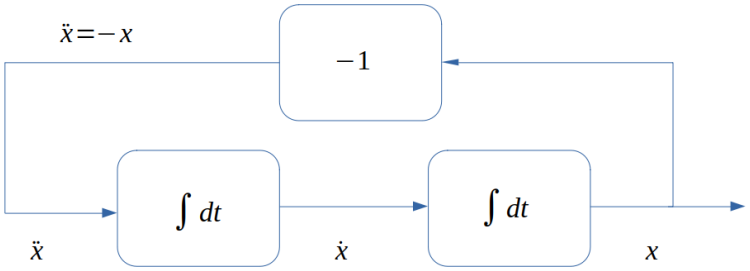


Figure 2.4: Basic implementation scheme of a coupled second order ODE, as in Eq. (2.5).

In this case, the implementation is direct, by simply replacing the blocks in Figure 2.4, with those discussed above. The resulting waveforms of both x and y variables are shown in Figure 2.5. In this figure, we have used $N_{acc} = 2^4 = 16$ iterations and $N = 18$ bits. This provides a $\Delta t = \frac{2^4}{2^{18}} = \frac{1}{8192}$ s. Using this relation, the period is expected to be $2\pi/\Delta t \approx 51497$ iterations, as it actually happens in Figure 2.5.

2.3 Implementing Chaotic Systems in SC

Implementation of chaotic systems using stochastic computing can be achieved by using the same three step process presented above. The main issue in this case regards the sensitivity to noise, for such systems. Thus, an adequate NF will require a high number of bits. To compare the results coming from integrating within the proposed SC environment, to those utilizing the conventional methods, the Shimizu-Morioka system [57] was utilized as a paradigm. Therefore, a comparison between these two implementations of some typical nonlinear time series analysis estimators (Kolmogorov entropy, correlation dimension etc.) were performed; documenting an evaluation of the effectiveness of the SC implementation. It should be mentioned that the classical integration was performed using Matlab, while the *SCN* was performed using the same FPGA than in the previous section.

Regarding the RNG, we have used a single 64-bit implementation of the Mersenne algorithm, which provides a period long enough for our purposes.

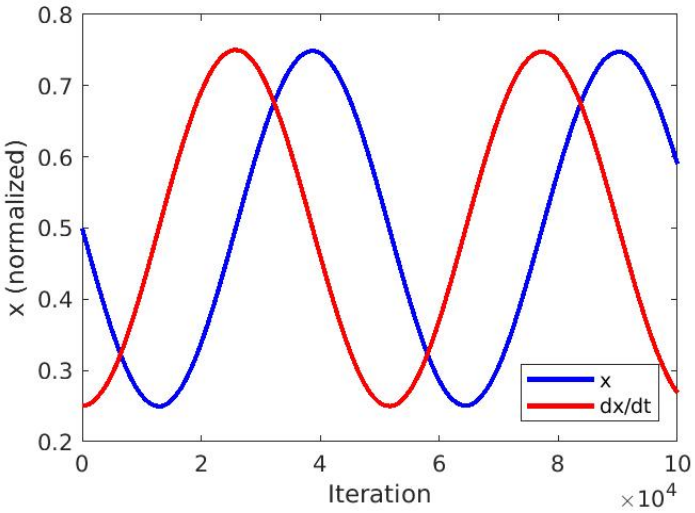


Figure 2.5: Output of the scheme in Figure 2.4, showing both x and $\dot{x}$. In this case, $N_{acc} = 16$, $N = 18$ *bits*, resulting in a $\Delta t = 1/8192$ s.

Notice that most of the RNG we need are only 1-bit long, so we can extract all of them from the same RNG. In addition, we have used the technique proposed in [58], which allows using a single generator to create different random numbers to the integrators and constants to the generators. The scaling constants are stored as fixed registers, which are then used to generate the random sequence through a B2S converter. It is worth noticing that the RNG is actually responsible of a large part of the total energy consumption, due to the large number of computations needed by the algorithm. This energy consumption could be addressed in the future by using memristors to generate the random bits, as proposed in [59–61].

2.3.1 The Shimizu-Morioka System

A system algebraically simpler than the Lorenz system has been proposed by Shimizu and Morioka [57] in 1980. The original equation formulation appears below:

$$\left.\begin{aligned} \dot{x} &= y \\ \dot{y} &= x - \mu \cdot y - x \cdot z \\ \dot{z} &= -\alpha \cdot z - x^2 \end{aligned}\right\} \quad \text{Shimizu Morioka System} \tag{2.6}$$

A usual set of parameter values, leading to the emergence of chaos in this system, is $\mu = 0.81$ and $\alpha = 0.375$. By applying this set, the system is operating in a deterministic chaotic mode, demonstrating an elegant chaotic attractor in the corresponding 3D phase space. As expected for determin-

istic chaotic systems, the trajectories comprising this strange attractor are bounded, while the whole system behavior appears to be bounded within a box (the phase space), the limits of which are explicitly presented in relations ((2.7)).

$$\left.\begin{aligned} x \,\epsilon\, (-1.5, 1.5) \\ y \,\epsilon\, (-1, 1) \\ z \,\epsilon\, (-2.5, 2.5) \end{aligned}\right\} \quad \text{Variable Boundaries} \tag{2.7}$$

2.3.2 Equation preparation

It is apparent that the next step in implementing equation set ((2.6)) in SC environment, is to get through a normalization procedure, as this has been discussed above. This would ensure that all three variables (x, y, z), would be within the SC working interval. In our case, we opted for normalizing all the variables to be within the [0,1] range. In order to achieve this, the following variable transform ((2.8)) was applied, as a first step:

$$\begin{aligned} X = -\frac{x+2}{4} \rightarrow x = 4X - 2 \\ Y = -\frac{y+2}{4} \rightarrow y = 4Y - 2 \\ Z = -\frac{z+3}{6} \rightarrow z = 6Z - 3 \end{aligned} \tag{2.8}$$

It is apparent that the proposed change of variables, transformed equation set ((2.6)) into the form appearing in equation set ((2.10)), after getting through the relations in ((2.9)).

$$\begin{aligned} \dot{x} = 4\dot{X} = y \\ \dot{y} = 4\dot{Y} = x - \mu y - xz \\ \dot{z} = 6\dot{Z} = -\alpha z + x^2 \end{aligned} \tag{2.9}$$

$$\begin{aligned} \dot{X} = Y - \frac{1}{2} \\ \dot{Y} = 4X - 2 + \frac{1}{2}\mu - \mu Y + 3Z - 6XZ \\ \dot{Z} = -\alpha Z + \frac{\alpha}{2} + \frac{8}{3}X^2 + \frac{2}{3} + \frac{8}{3}X \end{aligned} \tag{2.10}$$

Thus, the resulting equation system in ((2.10)) has all its (X, Y, Z) variables oscillating within the proper range for stochastic logic, but the equations are not yet ready to be implemented in a SC environment, This is due to the fact that some of the system-parameters are outside the SC range (in this case higher than one). To solve this, we divided all the terms by a number c_r higher than the highest coefficient appearing in the equations.

Notice that this is equivalent to a scaling of time of a magnitude $t' = c_r t$, thus no qualitative change emerges. In this specific case, using the value $c_r = 32$, the equation set ((2.10)) became as follows:

$$\begin{aligned} \dot{X} &= \frac{Y}{32} - \frac{1}{64} \\ \dot{Y} &= \frac{X}{8} - \frac{1}{16} + \frac{\mu}{64} + \frac{3Z}{32} - \frac{3XZ}{16} - \frac{\mu Y}{32} \\ \dot{Z} &= -\frac{2X}{24} - \frac{\alpha Z}{32} + \frac{\alpha}{64} + \frac{2X^2}{24} + \frac{1}{48} \end{aligned} \tag{2.11}$$

However, it has to be noted that, even if all the parameters and variables are within the proper range, the operations are not in a form ensuring that their results would fall within the SC range. In specific, the additions must be in the form suggested in Eq. ((2.12)).

$$a + b = \frac{1}{2}(2a + 2b) \tag{2.12}$$

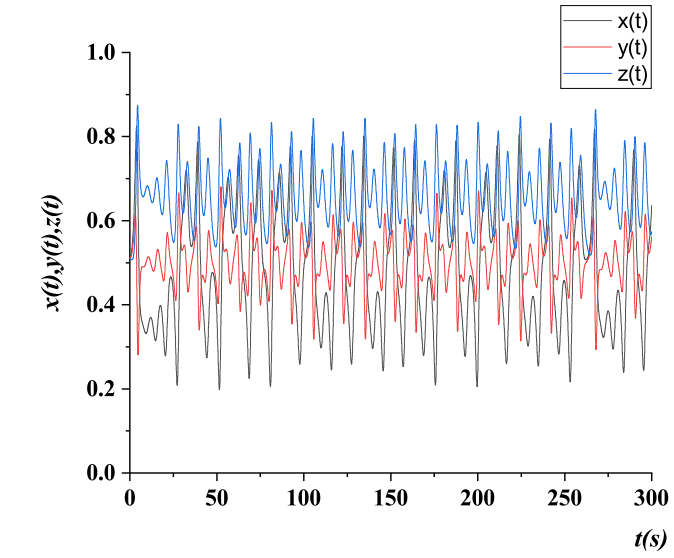
Rewriting all the equations according to this requirement, the equation set ((2.11)) transforms into in the one appearing in ((2.13)), which is the final form of the equations to be implemented in the SC environment.

$$\begin{aligned} \dot{X} &= \frac{1}{2}(\frac{Y}{16} - \frac{1}{32}) \\ \dot{Y} &= \frac{1}{2}\left[\frac{1}{2}[\frac{1}{2}(X - \frac{1}{2}) + \frac{1}{2}(\frac{\mu}{8} + \frac{3Z}{4}]] - \frac{1}{2} - \frac{1}{2}(\frac{3XZ}{4} + \frac{\mu Y}{8})\right] \\ \dot{Z} &= \frac{1}{2}\left[\frac{1}{2}\left[-\frac{1}{2}\left(\frac{2X}{3} + \frac{\alpha Z}{4}\right) + \frac{1}{2}\left[\frac{\alpha}{8} + \frac{2X^2}{3}\right]\right] + \frac{1}{24}\right] \end{aligned} \tag{2.13}$$

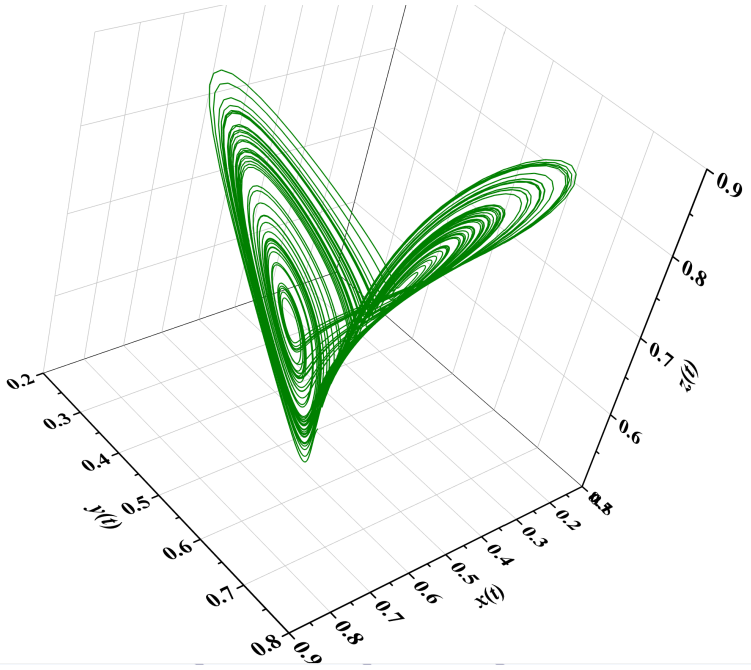
2.3.3 Implementation

The equation set ((2.13)) was implemented in a Matlab environment, being integrated in a conventional way by utilizing its built-in functions and the ode45 solver with variable time step. The solution of the system for all three system-variables in the time domain appears in Figure 2.6 (a), for specific initial conditions. In Figure 2.6 (b) the corresponding attractor of the system, in its 3-dimensional phase space, is apposed.

The same system (beginning from the same initial conditions) has also been implemented in a SC environment using the previously discussed basic gates and it is presented in Figure 2.7. In this figure, the x_i variable corresponds to any of the signals x, delayed by i clock cycles, an essential approach that improves decorrelation. Note that the delay element is not shown, but it is simply implemented by a shift register, taking into account that the variables are only 1 bit long. The number of bits used in this implementation was $N = 22$, with $N_{acc} = 2^{12}$ iterations. This would be



(a)



(b)

Figure 2.6: For the normalized Shimizu-Morioka system, beginning from initial conditions (x, y, z)=(0.51,0.51,0.51), (a) the nonlinear time series of the normalized Shimizu-Morioka system and (b) the corresponding attractor in a 3D phase space, are presented.

equivalent to a conventional system using 17-bit binary arithmetic, once the noise has been taken into account.

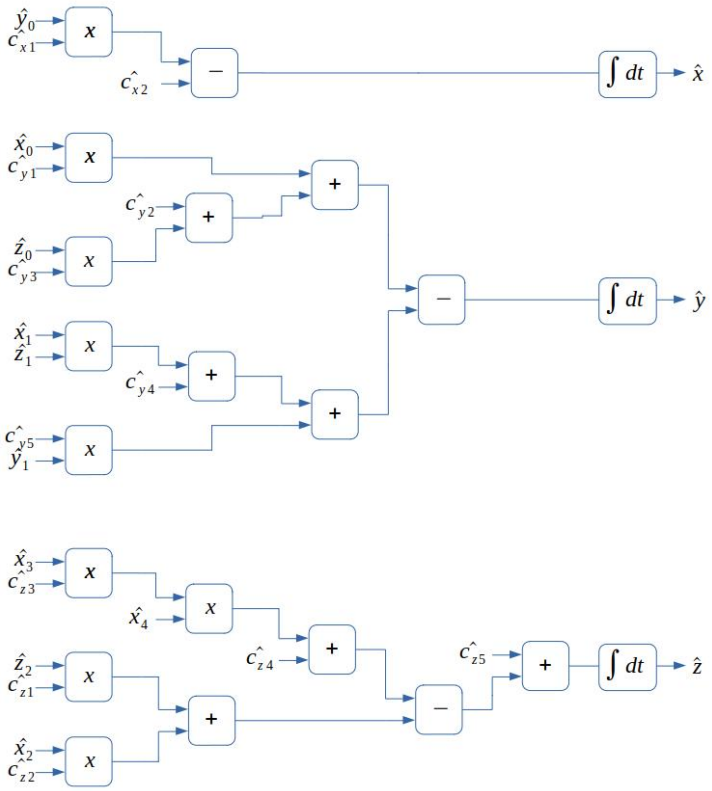


Figure 2.7: Implementation of the Shimizu-Morioka equations using SC. The sub-index in the variables means a delay equivalent to the number used to decorrelate them. The constants c_i are those corresponding to Eq. ((2.13)).

The results corresponding to this integration are presented in the form of time series (in the time-domain) in Figure 2.8 (a), where all three variables X, Y, Z appear. The corresponding attractor, embedded in a 3D phase space, appears in Figure 2.8 (b). Comparing these two figures to the corresponding Figs. 2.6 (a) and 2.6 (b) of the classical implementation, one gets the subjective perception that the two implementations evolve in time in a quite similar way, not the same though. Indeed, although in both implementations the systems begin evolving in time from the same initial conditions, their evolution is not identical, due to the approximate computing approach implemented in SC environment. However, the emerging attractors are in both cases demonstrating almost identical structure in their embedding phase space (3D in this case). A very draft remark is that the time series and the resulting attractor appears to be slightly more noisy, in

the case of SC implementation, something expected because of the approximate nature of calculations in the case of SL.

2.3.4 Chaotic Evaluation

Due to the nature of deterministic chaotic systems and in order to perform a more objective investigation of the fidelity of the dynamics demonstrated by the Shimizu-Morioka equation system, in the stochastic computing environment, a procedure including a variety of relevant metrics was applied.

Initially, the power spectrum of one of the state variables, namely $Z(t)$, was calculated in both cases. It is presented in Figure 2.9, where the red line regards the classical integration method and the green line the SC method. The similarity between them is obvious, including the minimum at around 3.5e-4 Hz and the maximum at 5e-3 Hz. The most apparent difference appears in the higher frequencies, which can be attributed to both the error in number quantization [62] and the noise from a random walk [50] as expressed in equations ((1.9)) and ((1.10)).

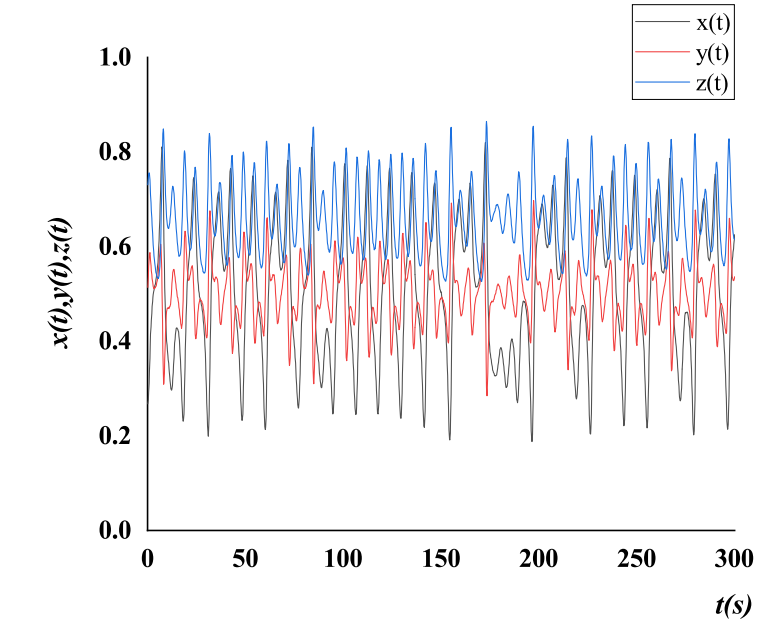
Further investigation and analysis of the demonstrated chaotic behavior included calculation of correlation dimension, Kolmogorov entropy, as well as Lyapunov exponents. To this direction, the $z(t)$ and $Z(t)$ variable from the three time series appearing in Figure 2.6 (a) and 2.8 (a) correspondingly, were considered.

Applying the Takens theory [63] to the $Z(t)$ time series, the topologically equivalent attractor was reconstructed, in the proper phase space. It is noted that according to this theory, the technique of displaced vectors is applied. So initially, the appropriate time delay τ, needed for the reconstruction of the phase space, was calculated by both the mutual information's first local minimum and the autocorrelation function's first zero. In both cases, the conventional and the SC solution, the value emerging for τ appeared to have a significantly lower value when calculated through the mutual information, and therefore this was the one adopted [20]; for the specific set of parameters and initial conditions: τ=16 for the conventionally calculated solution and τ=8 for the SC solution (in both cases we refer to measurement points).

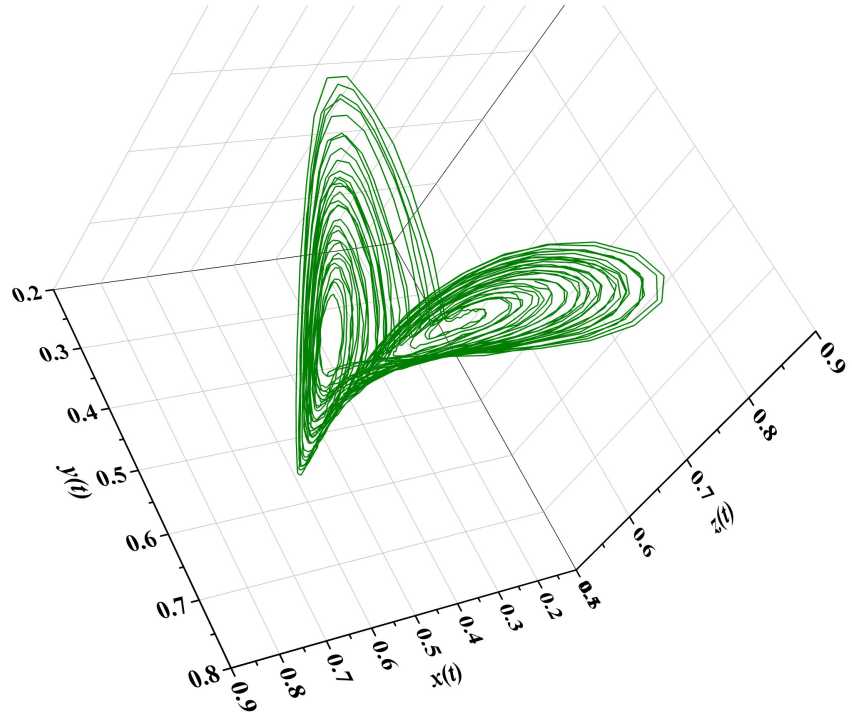
The correlation integrals $C(2,\ell)$, for different embedding dimensions, have been numerically calculated, according to the Grassberger-Procaccia method [64–66], Eq. ((2.14)):

$$C_m(2,\ell) = \sum p_i^2 = \lim_{N\to\infty} \frac{1}{N^2} \sum_{\substack{\mathrm{i,j}=1 \\ \mathrm{i}\neq\mathrm{j}}}^{N} \Theta\left(\ell - \sqrt{\sum_{k=1}^{m} |X_{j+k} - X_{i+k}|^2}\right) \quad (2.14)$$

where parameter ℓ is the hyper-cube dimension in the hyper-phase space for a series of specific embedding dimensions m, and Θ the Heaviside function.



(a)



(b)

Figure 2.8: (a) The nonlinear time series of the Shimizu-Morioka system, as this was calculated using SC, beginning from initial conditions (X, Y, Z)=(0.51,0.51,0.51) and $N = 22$ bits, $N_{acc} = 2^{12}$ iterations. (b) The corresponding attractor.

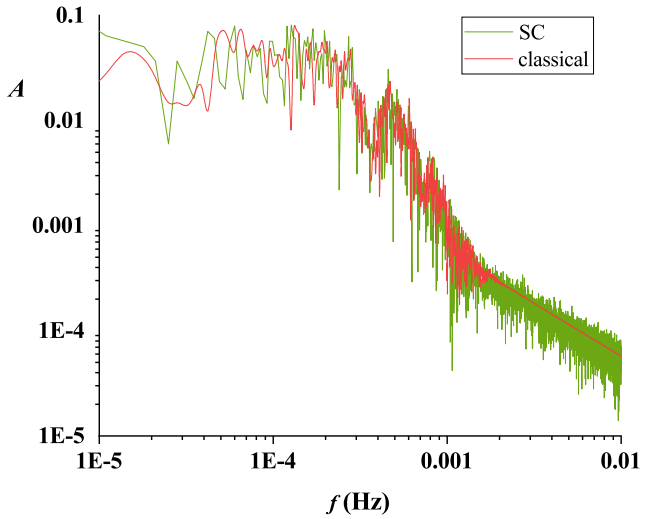


Figure 2.9: Power Spectra obtained from the $Z(t)$ time series using the SC (green) and classical (red) integration methods.

These integrals provided information characterizing the attractor and were calculated for embedding dimension up to $m = 6$, for both the conventional and the SC solutions of $Z(t)$. In both cases, the integrals appeared to almost parallelize for embedding dimensions $m = 3$ and above. The slopes of the linear parts of the correlation integrals (in a double logarithmic scaling) were determined for each embedding dimension (v vs m) and their values appear in Figure 2.10. In this plot, the black line refers to conventional solution, while the red one to SC. In both cases a saturation plateau appears for $m \geq 3$, thus, this is the sufficient phase space dimension, necessary to host the system's global dynamics under any circumstances [20].

In the case of the conventional solution (black line in Figure 2.10), the saturation tends to the non-integer value of $v = 2.10$, which is the correlation dimension of the attractor under these circumstances, further proving the deterministic chaotic nature of the studied time series (and the corresponding system). The closest (to the correlation dimension) higher, integer value defines the minimum embedding dimension, which in this case appears to be $m_{min} = 3$, as expected for a three state-variable system. It is apparent that for the Shimizu Morioka system, the calculated minimum embedding dimension coincides with the minimum sufficient phase space dimension $m_{min} = m_{suff} = 3$.

In the case of the SC solution (red line in Figure 2.10) the relation of the correlation integral slope, for all the embedding dimensions (v vs m) is again saturated after $m = 3$, but this time to a slightly higher non-integer value;

thus the correlation dimension in this case is $v = 2.30$. This difference is something expected and fully explainable, since the approximate computing nature of SC is introducing a specific level of noise, which is expected (and hereby verified) to increase the attractor's dimensionality. However, this increase is within the system's minimum dimensions $m_{min} = m_{suff} = 3$.

In order to investigate the global chaotic dynamics, the Kolmogorov-Sinai Entropy and the Lyapunov exponents were calculated. The Kolmogorov Entropy appears in Figure 2.11, in both cases. It clearly possesses positive value, $K_2 = 0.44$ bits/τ for the conventional solution, and $K_2 = 0.54$ bits/τ for the SC solution. These values are almost the same depicting similar rate of loss of information of the past state of the system, therefore a similar deterministic chaotic nature for both implementations.

Table 2.1: Values of the first three Lyapunov exponents for the 'Z' variable in the cases of classical integration and SC integration.

Order	Classical	SC
λ_1	0.02112	0.03774
λ_2	-0.00487	-0.00462
λ_3	-0.31142	-0.29139

Finally, the three (as expected for a 3-dimensional system) corresponding average local Lyapunov exponents, as they were calculated by the observed $Z(t)$ time series [67], are presented in Table 2.1. The maximal exponent provides with a measure of the predictability of system producing the studied time series [67, 68]. If at least one of them is positive then the system is chaotic. In both investigated cases, the system demonstrates one positive exponent, one nearly zero (due to the finite time series and the approximation introduced by the calculating method) and one negative, hinting for a simple chaotic system. Moreover, in both implementations the values were close one to the other. The higher value of λ_1 in the case of the SC solutions, is again expected and due to the noise introduced by the approximate calculations taking place in the case of the SC implementation [20], [29]. Additionally, the second exponent λ_2 is zero in both cases as expected, and the third ones λ_3 are also close in value. It is noted that in this case the value of the maximal exponent (the only positive) also provides the lower bound of the Kolmogorov-Sinai entropy.

All the above calculations of nonlinear dynamics established metrics, prove and confirm the ability to implement the chaotic dynamics of Shimizu-Morioka system in SC, indifferent to the approximate nature of this kind of calculations.

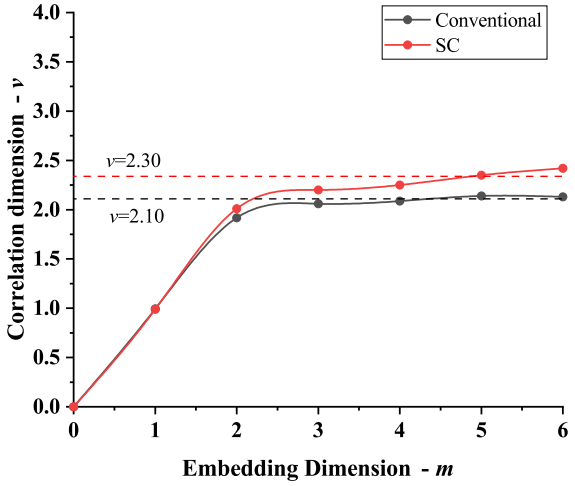


Figure 2.10: Correlation Dimension for the conventionally calculated $Z(t)$ time series (black line) and the one calculated in the SC environment (red line).

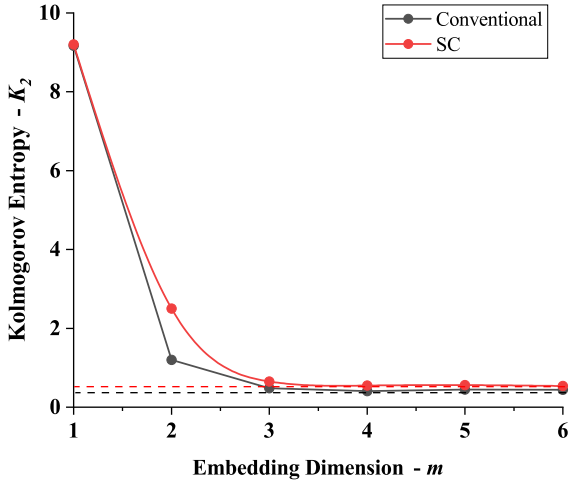


Figure 2.11: Kolmogorov-Sinai Entropy for the conventionally calculated $Z(t)$ time series (black line) and the one calculated in the SC environment (red line).

2.4 Discussion

Having in focus edge computing and data trafficking in the frame of the IoT, approximate computing emerges as an essential technological option, being able to provide low cost in both area and energy for (relatively) low precision calculations. To this direction, in this work we have presented a detailed procedure to implement nonlinear equations using stochastic computing (a kind of an approximate computing method). This procedure involves a renormalization of the equations in three phases: first, it sets the possible values of the variables inside the values [-1,1]; secondly, all the constants inside the equations are recalculated so that they would also lay within the same range, which is equivalent to a scaling of the time variable; finally, all the operations are rewritten in a form compatible to the available operations in stochastic computing.

We have also discussed the effect of changing the number of bits used to represent the variables, showing that this number is related to the signal-to-noise ratio that the system can withstand. Specifically, since the process is a random walk, the noise figure is expected to grow according to the bit-string length, as $\sqrt{N}$.

The main advantage of the presented approach is the low number of components needed to implement the equations, which is far below the required number demanded in the classical approach, due to the method of implementing SC calculations. This makes the specific approach useful for small (lightweight) systems that require to make complex calculations with a low energy consumption, mainly if the needed number of bits is not large. In addition, the robustness of nonlinear systems in the frame of SC was also shown. This is specially relevant, since nonlinear systems are highly sensitive to initial conditions and parameter variations, which may greatly change their long-term behaviour. In our case, we have shown that this long-term behaviour is kept, even when all the constants and parameters have been defined as SCN.

As examples of application, we have implemented three different systems representing as many differential equation systems: a double integrator, a simple oscillator, and a nonlinear system. As already discussed above, the results from the first two cases appear to be exactly the same than the conventional system implementations. In the case of the Shimizu-Morioka system, which has been used as the toy model for nonlinear system implementation, the results obtained using SC are consistent with a conventional implementation. The metrics used for comparing the implementation in the SC versus the conventional one were three: the correlation dimension (2.10 for the conventional case, 2.30 for the SC); the Kolmogorov entropy (0.44 bits/τ for the conventional, 0.54 bits/τ for the SC); and the Lyapunov exponents, which were also found to be very similar.

Related to the performance of the system in stochastic computing, it is

clear that its worse aspect is the time needed to perform the calculations. In the proposed FPGA implementation, the throughput of the system is one bit per clock cycle, and it needs 2^N cycles to perform a single δt iteration. This total time can be reduced by parallelization. Thus, using M identical blocks in parallel as proposed in [44], the total time needed to perform the iteration is reduced accordingly to be t_{SC} =2^N/M clock cycles. On the other hand, when considering conventional binary operations implemented sequentially, the total time would be given by:

$$t_{conv} = \sum_x N_x \tau_x \tag{2.15}$$

where τ_x is the number of cycles needed to implement the arithmetic operation x, which appears N_x times in the circuit. In our case, we only have used multiplication (mul), addition (add), subtraction (sub), and integration (int). Of those, multiplication needs usually between 3 and 7 clock cycles, and the others only 1, since the integral is just another addition. Thus, for our system, we need a total of clock cycles between 40 and 76, with an expected value around 67 cycles. Thus, for a 16 bit implementation with M parallel SC branches, the ratio between t_{conv} and t_{SC} is:

$$\frac{t_{conv}}{t_{SC}} = M\frac{\sum_x N_x \tau_x}{2^N} = M\frac{67}{65536} \approx M/1000 \tag{2.16}$$

Thus, for a simple implementation with 32 parallel branches, the conventional arithmetic would be 32 times faster than a Stochastic Computing equivalent. This, as has been discussed above, leads to a trade-off between speed, precision, and power consumption. Notice, however, that for an application requiring a lower number of bits (as, for instance, in [38], where only 10 bits are needed), this is changed to $\approx M/16$ and an SC implementation could be actually faster and simpler than a conventional one.

Related to the other performance parameters of the system, we have to note that in this thesis we are focusing only on the possibility of implementing such systems, with no emphasis on the energy consumption or area optimization. These two aspects are still under consideration for our proposal, since the literature seems to be unclear in this aspect. However, as an example, related to the area optimization, we can compare the difference between the number of logic elements needed to implement a stochastic computing (always a simple gate) against a usual digital implementation of a vedic multiplication [69] into a certain FPGA, as in [70]. These results are compiled in Table 2.2, and show a clear advantage of SC over conventional implementation. Related to energy consumption, it is known [44] that for short bit-streams (less than 16-17 bits) SC performs better than conventional. Notice, in any case, that the energy consumption needed to generate the random bits is not taken into account, since they can be generated in multiple ways. For instance, a 64-bit Mersenne RNG needs a lot of power to

Table 2.2: Comparison of the number of FPGA parts used for implementation of the vedic multiplication algorithm [70] and the SC multiplication. (*)The number of LUTs used in SC is considered to be 1/3 of a 6-input LUT, as those in the FPGA used in [70].

Algorithm	bits	Slices	LUTs
Vedic	4	19	33
SC	6	1	1/3*
Vedic	16	346	622
SC	22	1	1/3*
Vedic	32	1427	2566
SC	32	1	1/3*

perform all the calculations that lead to the pseudo-random sequence of bits, but if we can use memristors for this task [59–61] this energy is drastically reduced, as well as the needed area. Additionally, a final ASIC implementation of the design could utilize some improvements, as extensively discussed in [44], that allow for exponential improvement of consumption.

The results emerging from the SC implementation are thus showing that this technique is capable to implement complex nonlinear systems, in an area-efficient way, and with small loss of precision, equivalent to an analog circuit implementation. Taking into account their possible application for secure data transmission, they are one possible alternative to be considered in IoT, or Edge computing systems, paving the way for an even wider spread of IoT.

It has to be noted that parts of this chapter were published in [30].

Chapter 3

Implementation of Memristive Systems Emulators

3.1 Introduction

Memristors are the latest fundamental breakthrough in circuit theory and their applications are going to be an important, key-factor in electronic circuit design. They already appear in many forms, like PCA, ReRAM, etc., to mention just a few. However, due to the fact that memristors have appeared quite recently, technology is not mature enough to provide with readily available, off-the-shelf components. As a result, developing and testing new concepts or design architectures based on memristors, are accomplished mainly by using numerical simulation.

To this direction, many good memristor-models have been proposed, either using the classical approach, which utilizes current and voltage [71–73], or the alternative approach that studies and models memristors withing the charge and flux domain [74–76]. However, most of these models appear to demonstrate drawbacks in terms of simulation effectiveness, something that makes simulation of large circuits rather difficult or even impractical [77].

Emulators reproduce the operating characteristics of the memristor by eliminating the aforementioned problems, therefore allowing for the development of more complex and reliable systems [78]. The memristor behavior which is imitated can be an ideal memristor or actual device, depending on the implementation. If we are focused on their field of application, emulators have different characteristics, although there are two main lines of study: analog emulators and digital emulators. The first approach regards analog circuits mimicking this behavior, and a variety of such circuit topologies have been proposed ([79–84], etc...). Most of them ([82–84] etc.), use active

analog blocks (op-amps, OTAs, or advanced elements as current conveyors), resulting into complex and power consuming implementations. Some others use passive electronic elements [79–81], but then they are usually limited to short-term memory (volatility). A second quite important approach pursues the implementation of memristor models onto FPGAs (or ASICs) [85–87]. These are digital circuits providing with good simulation times. This approach is bulky and requires complex implementations, notably in designs with a high number of digital elements. An advantage is the exact control of the emulator behavior, since all the equations are user-defined and explicit.

Many works develop fully analog emulators; for example, in [79], a memristive system was implemented and results demonstrated that it was very easy to fabricate in academic laboratories through classical electrical components from circuit theory. In [81], an emulator is implemented with transistors, resistors and diodes, and it operates in passive mode. Other examples like [88, 89] use amplifiers in their models. In general, analog systems need more power consumption. The volatility of the system is worse than in the digital case, but they present a good implementation of the variable resistance with which the emulator memristance is described.

On the other hand, digital memristor systems emulators can be implemented in FPGAs (or ASICS) [85–87]. Their main advantages are that they present short simulation times and better control of the behaviour of the emulator. However, their variable resistance implementation poses a problem. In digital emulators, it is much easier to define the model, but precision is lost (limited number of bits), and there is usually a need for more computational power than the analog equivalent. For a review of different state-of-the-art emulators, the interested reader can see, for instance, [78] or [90].

In this chapter, we present different implementations of memristors and memristor-based systems based on stochastic computing. Specifically, we present three different implementations, based on part of our published work:

1. Purely digital memristor emulator based on a flux-charge model [31].
2. Switched capacitor memristive emulator [32].
3. Stochastic switched capacitor memristor emulator [30].

3.2 Memristor Modelling Framework

A memristor is a two-terminal device whose resistance (conductance) can change its value when a voltage or current signal is applied. In addition, the value of the resistance (conductance) of the device also depends on its past history and is named memristance (M) (memconductance (G)). The concept of the memristor was extended by Chua in 1976 to memristive systems to explain the behavior of observed systems [91], for instance, in

nature. Nowadays the classification of memristors includes ideal, generic and extended memristor [92].

The most general class is the extended memristor, which includes the others. The dynamic of this class is described using internal variables that determine the internal state of the memristor; these variables, can be for example, temperature or geometrical parameters, depending on the system. The memristor can be voltage- or current-controlled, depending on the input source. On the other hand, in [93], Corinto et al. proposed a mathematical description in the charge flux domain instead of the voltage and current domain. We use for our emulator the equations describing a voltage-controlled extended memristor in the charge flux domain. These are:

$$i = G(\varphi, v, \mathbf{x}) \cdot v \tag{3.1}$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{g}(\varphi, v, \mathbf{x}) \tag{3.2}$$

$$\frac{d\varphi}{dt} = v \tag{3.3}$$

The memconductance (G), which can be nonlinear, is the inverse of the memristance of the device M, v is the voltage between its terminals, i is the current, ϕ is the flux (i.e. the first momentum of voltage), and $\mathbf{x}$ represents other possible state variables.

Finally, it is also important to mention that the memristors present some characteristic fingerprints distinguishing those of other dynamic systems [93, 94]:

1. As Leon Chua noted in [95]: "If it's NOT pinched, it's NOT a memristor". The i-v curve obtained when a periodic signal with zero DC component (voltage or current) is applied to the memristor shows a pinched (at the (v=0,i=0) point) hysteresis loop;

2. The area of the hysteresis loop should tend to zero for higher frequencies, as noted in [93]. The behavior at low frequencies depends on the specifics of the memristor, and there may even exist a frequency where the loop area is maximum [78].

On the other hand, the emulator function must be to mimic the memristor behavior; this is to show its fingerprints. The emulator can be implemented in analog, digital or mixed formats. It is crucial that the circuit implements, among others, the internal state variables, (vector $\mathbf{x}$) in Equations (3.1) and (3.2). These internal variables must be included as electrical

variables in the emulator and are assumed to be isolated from a direct interaction with the outside. They are used, together with the electrical variables (i.e. voltage and flux), to calculate the value of the equivalent memconductance (G) or memristance (M).

Notice that we have implemented the ideal definition of a memristor, with a simple relationship between memristance and flux. Other models, even those oriented to the simulation of actual physical systems as, for instance, in [96–98], could also be implemented. The main difference of this case with the one presented here would be the implementation of the non-basic mathematical operations. This could be done using, for instance, the different circuits proposed in [48,49,51] for division and the associated square root calculation, or in [52] for arbitrary function approximation.

3.3 A Memristor Emulator based on a Flux-Charge Model

In this section, we propose an alternative approach to emulate a memristive system. We propound the design of a purely digital system, using stochastic computing. The advantages of such an approach for memristor emulation are evident: it allows all the possibilities of a digital system to be implemented in FPGAs, while at the same time conserving the properties that make memristors useful.

The emulator is based on a flux-charge description of the memristor, as proposed in the literature. It has been implemented in a commercial FPGA, and it has been shown that its behaviour mimics that of a circuit using memristors. Specifically, we show that it reproduces the fingerprints of a memristor, both in frequency and memory capability. Finally, we also reproduce an IMPLY gate, showing that the behaviour is correctly reproduced.

3.3.1 Digital Implementation of a Memristor Model

A very good theoretical description of memristors has been given by Corinto et al. in [93]. This framework has been already used to successfully model different kinds of memristive systems. For instance, in [74] which presented a semi-empirical model for unipolar ReRAMs as memristors, or [75] where a model for phase change memories is presented.

As we have discussed in the above references, describing a memristor using charge or flux as the electrical variables may be advantageous over using voltage or current. In the case where we wish to implement a digital-only model for a memristor, we have to decide which are our variables, and how are we observing them. First of all, we settle for an ideal memristor. That is, we will have a relation between the flux and the charge, with no

state variables, simplifying the previously mentioned equations, as shown in Eq. ((3.4)) and Eq. ((3.5)):

$$\phi = f(Q) \tag{3.4}$$

$$Q = \int i(t)dt \tag{3.5}$$

In a similar way as was done in [99], we can approximate the conductivity to two different states: high resistance (R_H) and low resistance (R_L) (usually called HRS and LRS, respectively).

$$\phi = \begin{cases} R_L \cdot Q, & Q > Q_{th} \\ \\ R_H \cdot Q, & Q < Q_{th} \end{cases} \tag{3.6}$$

where Q_{th} denotes a prescribed threshold. Notice that we can rewrite Eq. (3.6) as an equation for R plus another equation similar to Ohm's law for the charge and flux:

$$R(Q) = \begin{cases} R_L & Q > Q_{th} \\ \\ R_H & Q < Q_{th} \end{cases} \tag{3.7}$$

$$\phi = R(Q) \cdot Q \tag{3.8}$$

We can implement the above equations in a purely digital circuit using some approximations. First, we calculate the current flowing through the device as:

$$i(t) = \frac{V^+ - V^-}{R(Q)} \tag{3.9}$$

Thus the charge can be calculated as:

$$Q(t) = \int i(t)dt = \int \frac{V^+ - V^-}{R(Q(t))}dt \tag{3.10}$$

If we approximate the integral by a summation, Eq. (3.10) converts to:

$$\frac{Q(t)}{\Delta t} = \sum \frac{V^+ - V^-}{R(Q(t))} = \sum \frac{\Delta V}{R(Q(t))} \tag{3.11}$$

Assuming, without loose of generality, that $\Delta t = 1$, we can approximate Eq. (3.11) as:

$$Q(t) = Q(t-1) + \left(1 + \left(\frac{R_H}{R_L} - 1\right) g(Q(t-1))\right) \frac{\Delta V}{R_H} \tag{3.12}$$

where $g(Q)$ is defined as:

$$g(Q) = \begin{cases} 0, & Q < Q_{th} \\ \\ 1, & Q > Q_{th} \end{cases} \tag{3.13}$$

Notice that R_H/R_L is usually a fairly big number, and can be considered to be an integer. Thus, Eq. (3.12) can be easily implemented using a digital circuit, if we assume, again ,without lose of generality, for a digital implementation, that we choose a reference system where the value of R_H is taken as 1 and V is either 0 or 1. An implementation of such a circuit is proposed in Fig. 3.1. For this implementation, we assume that the counter has a maximum value and a minimum value, and, for simplicity, that its initial value is zero.

We have implemented this emulator into a DE2-70 FPGA by Altera, using QuartusII-32bits to compile it. The resulting circuit is implemented using less than 100 gates, or less than a 1% of the available number of gates. More modern FPGAs, with up to 5.5 million gates, could then possibly implement complex circuits with more than 50k memristors.

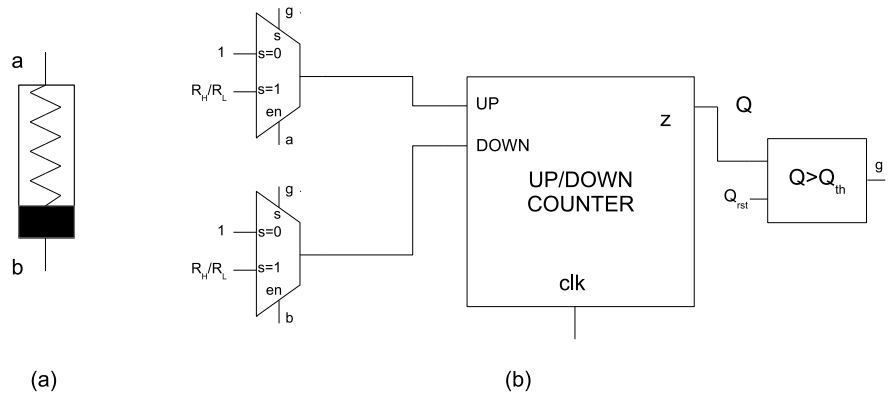


Figure 3.1: Schematic implementation of the digital memristor. (a) Symbol for a memristor. (b) Proposed implementation. The inputs $a = V^+$ and $b = V^-$ can be only 0 or 1, since this is a pure digital circuit. We also assume that the counter has a maximum and a minimum. The initial state is chosen to be zero.

3.3.2 Results and Discussion

In this section we first test the proposed circuit to check that it behaves like a memristor. Once this is established, we apply the emulator to a classical memristor circuit: an IMPLY logic gate. Notice that, as it will be discussed later, the IMPLY logic gate is the structure we get when connecting in parallel two memristors. From these basic blocs, more complex structures can be constructed.

3.3.2.1 Memristive Behavior of the Emulator

As has been stated extensively in the literature (see, for instance, [94]), ideal memristors have two very well defined fingerprints: (1) at zero volts, there is no current, and (2) at high frequencies the hysteresis loop tends to be a straight line through the origin.

In the flux-charge space, these conditions translate into: (1) at constant flux, charge is constant. This condition is the equivalent to condition (1) for the voltage and current, while its second condition naturally arises from Eq. (3.6), but would arise also from any similar description [93]. This last paper demonstrates this rigorously, but the intuition behind is that it is so because for high frequencies there is not time enough for the charge to reach Q_{th} and, thus, remains in its original state.

In order to test that the proposed circuit behaves as expected, we force it with a pulse series at the positive and negative terminals and we monitor the resistance state (HRS or LRS), as well as the internal counter we are using as the charge.

We have tested two different frequencies: low and high, defined as those that allow reaching or not the charge threshold, respectively. The result from low frequency is shown in Fig. 3.2. In this picture, we can see clearly that, when a is higher than b, the charge is increasing slowly until it reaches the threshold value (Q_{th}). Then, the state changes (from HRS to LRS), and the charge increases faster, as defined in Eq. (3.12). Notice that when both inputs are zero, the charge is kept constant, thus obtaining one of the two desired fingerprints. For b higher than a, the charge decreases with the same kind of dual behaviour.

On the other hand, Fig. 3.3 shows the results at high frequency. In this figure it is clear that the charge does not reach the threshold and, as a consequence, the state remains also unchanged. Then, considering both examples at low and high frequency, it is also clear that this circuit presents memristive behavior, since it positively shows the two basic fingerprints of a memristor. It is worth mentioning that even if the memristor state change can be emulated with simpler circuits, emulating the behaviour with frequency requires this kind of digital architecture.

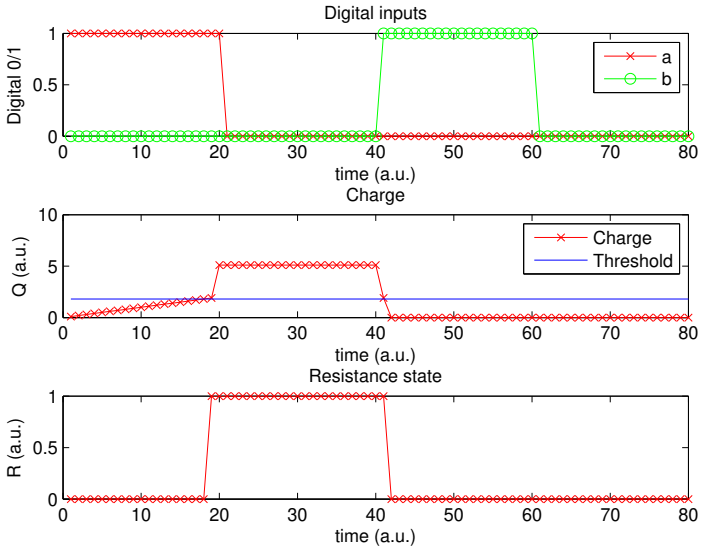


Figure 3.2: Simulation of the behavior at low frequency. The upper graph shows the digital inputs a and b (positive and negative, respectively). The middle section shows the charge, as counted by the emulator. The bottom graph shows the resistance state, where 0 corresponds to the HRS and 1 to the LRS.

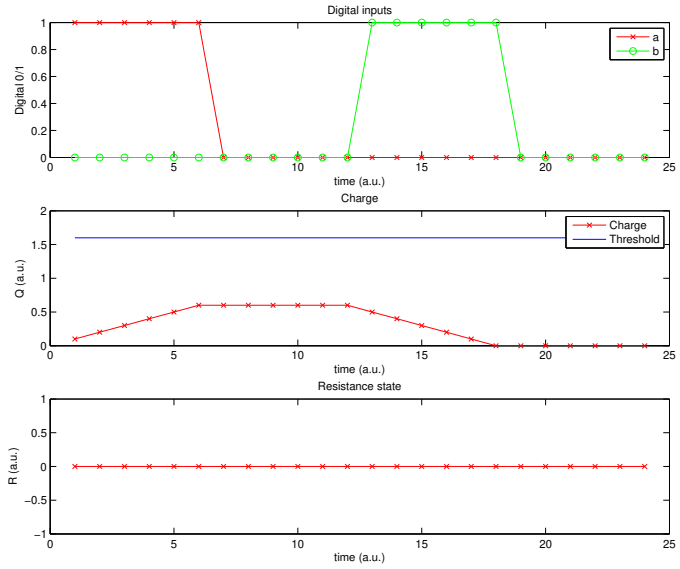


Figure 3.3: Simulation of the behavior at high frequency. The upper graph shows the digital inputs a and b (positive and negative, respectively). The middle section shows the charge, as counted by the emulator. The bottom graph shows the resistance state, where 0 corresponds to the HRS and 1 to the LRS.

3.3.2.2 IMPLY Logic Gate

The most natural way to implement logic functions using memristors seems to be the use of IMPLY logic (see [100] or [101] for instance). We show a classical implementation of a two-inputs IMPLY gate in Fig. 3.4, and its truth table is shown in table 3.1. The operation is based on a two-step process. In the first step, we set the corresponding input state to the memristors. The second step is where the calculation is performed, and uses two different voltage levels into P and Q, with V_P not higher than V_Q. If P is in LRS, then the voltage at node z will be, approximately, V_P. Thus, a voltage across Q is created, but is not sufficient to change the state during its application. In the case of P and Q being both in HRS, thanks to R_G, the voltage at z will be, approximately, V_Q and Q will go to LRS. In the case of P in HRS and Q in LRS, the output is dominated by Q which will not change its state.

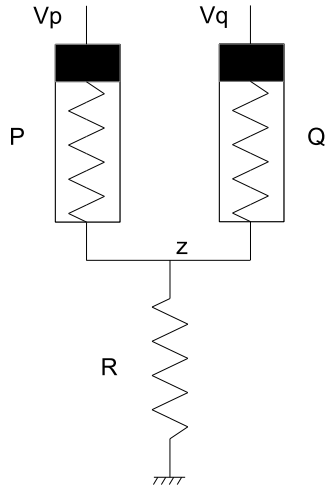


Figure 3.4: IMPLY gate. A classical two-inputs IMPLY gate.

Table 3.1: Truth table of imply function.

Case	p	q	p → q
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

The problem of using our emulator to implement this kind of scheme is that we have to devise some way to combine two emulators to conform

Table 3.2: Sequence for setting up the input values of the memristors. Notice that we are using two clock cycles for the process, so AB means applying A during the first cycle and B during the second one. During this process, the reset signal is high and the output z follows a '10' sequence, forced by signal b. The set process is selected by signal $s = 0$, while calculation is performed at $s = 1$.

Case	V_P,V_Q	r_p,r_q
1	00	1
2	11	0

the circuit. Conceptually, this is solved by stating that the output will be dominated by the input corresponding to the memristor with the lowest resistance. This being said, we can then calculate the output (z) of the gate in terms of the inputs ($V_p = b_1$ and $V_q = b_2$) and the memristor states (r_1 and r_2, respectively, where $r_x = 0$ is the HRS).

$$z = b_1 \cdot r_1 + b_2 \cdot r_2 \tag{3.14}$$

In this Eq. (3.14), the multiplications and the additions are AND and OR operators, respectively. Notice that z corresponds to the value of the positive terminal of the memristor. This equation can, in general, be used also whenever two memristors are connected through a common terminal (z). As said above, it is a winner-takes-all equation that states that the common node follows the signal connected to the memristor with the lowest resistance. It is important to point out that the case where p and q are equal makes this case equivalent to two memristors in parallel.

In order to allow the first step of setting up the initial values of the memristors we need to define a new signal v (a $RESET$ signal) that will force the required values. For the sake of simplicity, we have chosen the combination of values shown in Table 3.2. Notice that it takes two clock steps to set up the process, but it allows us to use the same signal for both memristors. In addition, we define a signal s that determines the state of calculation (s=0) or value setting (s=1). Thus, we get the final equation we use for our gate:

$$z = (b_1 \cdot r_1 + b_2 \cdot r_2) \cdot \overline{s} + s \cdot v \tag{3.15}$$

Using the simple operation in Eq.(3.15), we can combine two of our memristor emulators and make an IMPLY gate. Results are shown in Fig. 3.5. In order to interpret the results, the variable we are considering as the output is the state of memristor Q during the evaluation part of the calculation. This figure shows the four possible combinations of inputs, each

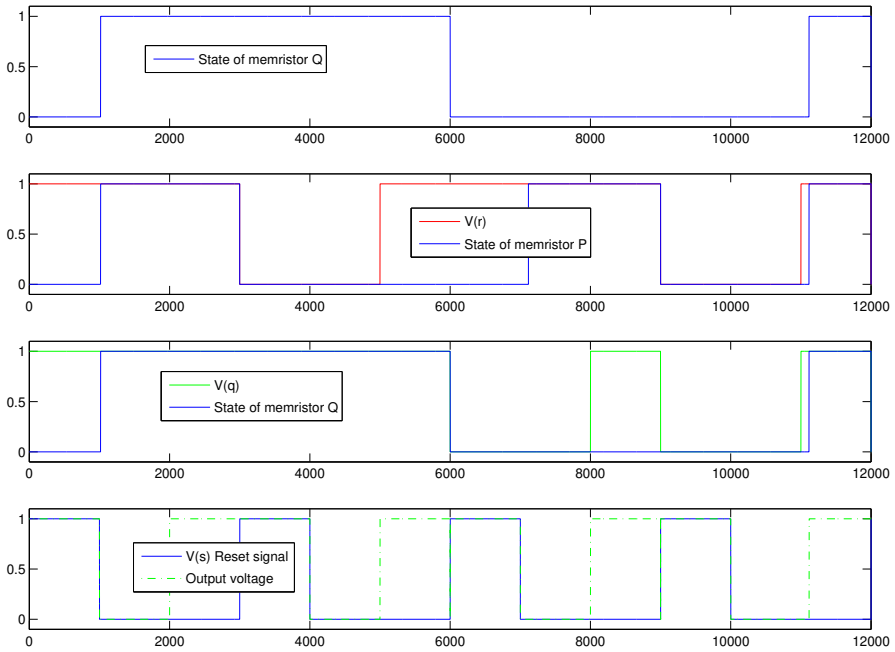


Figure 3.5: IMPLY gate simulation. Results from the simulation of a two-inputs IMPLY gate.

3.4 A switched capacitor memristor emulator

In this section, based on our work [32], we propose a novel, mixed-signal circuit for emulating memristive behaviors. The well-known switched capacitor (SwC) technique is utilized in order to implement a variable driven resistor, necessary for the implementation of the emulators memristance. We perform the control of this circuit using pulse width modulation, which can be adapted very fast to changes in the values, and is much simpler than a controlled pure resistance. This way the problems described above are indeed alleviated. Additionally, the proposed novel emulator provides with a higher linearity than that of a CMOS equivalent, and a fine control that depends only on clock cycle and not on any analog voltage value.

3.4.1 Switched Capacitor emulator

A block diagram describing the proposed mixed-signal memristor emulating system, appears in Fig. 3.6. The illustrated emulator implements the eqs. ((3.1)) to ((3.3)). As already mentioned above, the option of utilizing a switched capacitor module as the controlled resistance, is evident.

In Fig 3.7 we present the circuit of the SwC module (a typical one). Assuming that the two control signals Φ_1 and Φ_2 are equal, but with a 180

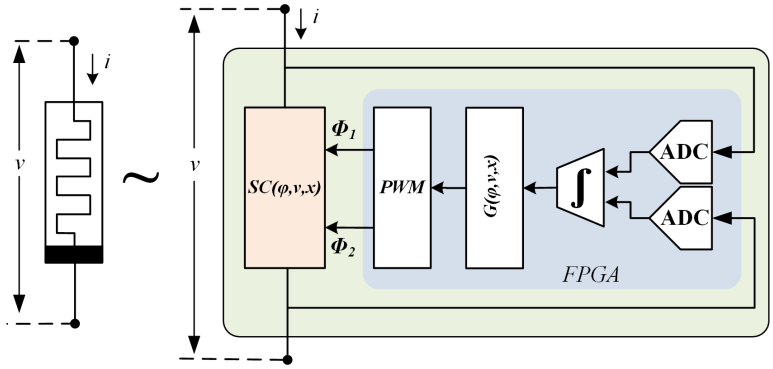


Figure 3.6: Block diagram of the switched capacitor circuit.

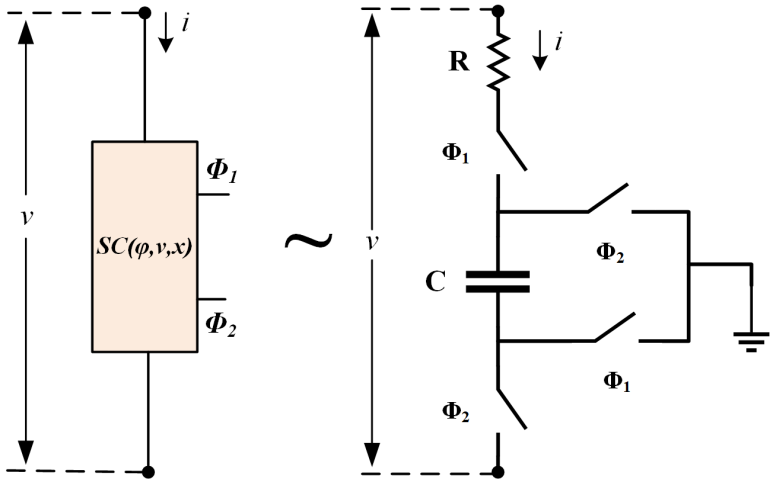


Figure 3.7: Schematic of the used switched capacitor circuit. Resistor R includes the shunt and parasitic resistances. Signals Φ_1 and Φ_2 must not overlap.

deg. lag, then the equivalent resistance R_{eq} of such a circuit is described by [102]:

$$R_{eq} = R_0 + \frac{1}{f_S C} \frac{1 + \exp\left(\frac{D\ T}{\tau}\right)}{1 - \exp\left(\frac{D\ T}{\tau}\right)} \tag{3.16}$$

In the previous equation, D stands for the duty cycle ($0 < D < 1$), $T = 1/f_S$ is the control signal period, f_C is the input signal frequency, C is the value of the used capacitor, and τ is the time constant, obtained by $\tau = C\ R_{total}$. In the later expression, R_{total} also includes the contribution from all the existent parasitic resistances.

3.4.2 Implementation and Results

The proposed emulator has been tested against the well-known signatures of memristor [94], as these were presented above. The main fingerprints are the *pinched* (zero voltage for zero current and vice verso), a hysteresis i-v loop, the area of which tends to zero (it becomes a simple line, without any hysteresis) at higher driving frequencies, tending to an ohmic behavior.

The system proposed in Fig. 3.6 was implemented using a DE0-Nano FPGA running at 25Mhz, with an internal clock divider to get the program running at 12.5 MHz. We used two of its 10-bit ADC to read the input voltage, and two of its output digital ports to drive the control signals of the analog switch that was used to implement the SC resistor. The integrator was implemented as an accumulator. At each clock loop, the accumulator incremented by the difference of the positive input (defined in Fig. 3.6 as the port where the current enters) minus the negative input.

The governing equation was a simple relation between charge and flux:

$$Q = M\phi^2 \tag{3.17}$$

where M is a constant. Thus, the resistance can be calculated as:

$$i = 2M\phi v \implies R = \frac{1}{2M\phi} \tag{3.18}$$

This resistance was then mapped linearly to a value between 0 and 1023, which was considered as the duty cycle D of the PWM. We designed the PWM using a cyclic counter T from 0 to 1023 and a comparator, according to the following:

$$PWM = \begin{cases} 0, & \text{if } T \geq D \\ 1, & \text{otherwise} \end{cases} \tag{3.19}$$

The SwC circuit in Fig. 3.7 was implemented on a prototyping board (Fig. 3.8) using the analog switch HCF4066FE, with a working voltage between -0.5 V to 22 V, and a maximum frequency switch-response of 25 kHz at 3.3V. The control signals Φ_1 and Φ_2 were generated by the DE0-Nano through two 3.3V digital output pins. We used a 1kΩ shunt resistor, and a 15μF capacitor. The value of the equivalent resistance of this system as a function of the input frequency and the duty cycle, was experimentally characterized, and is presented in Fig. 3.9.

We produced the input signal using an AFG320 arbitrary signal generator, and the system was monitored using two oscilloscopes. The first oscilloscope was used to monitor the control signals of the HCF4066FE, as seen in Fig. 3.10. In this figure, it can be clearly seen that both signals are complementary. A second oscilloscope was used to monitor the current, using the shunt resistance of 1kΩ, as well as the input voltage. Fig. 3.11

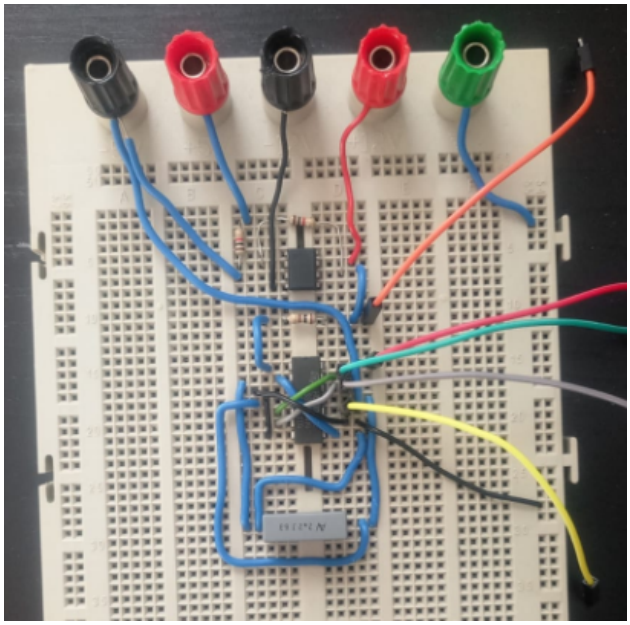


Figure 3.8: Physical implementation of the circuit on a prototyping board.

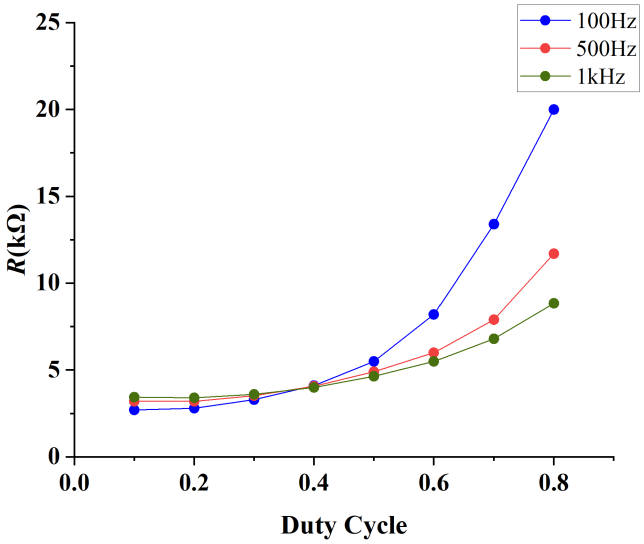


Figure 3.9: Equivalent resistance for various frequencies of the input signal, for $f_S = 12MHz$.

Figure 3.10: Snapshot of the oscilloscope showing the control signals of the analog switch. The upper signal is Φ_1, while Φ_2 is the lower signal. Notice that both signals are complementary and non-overlapping.

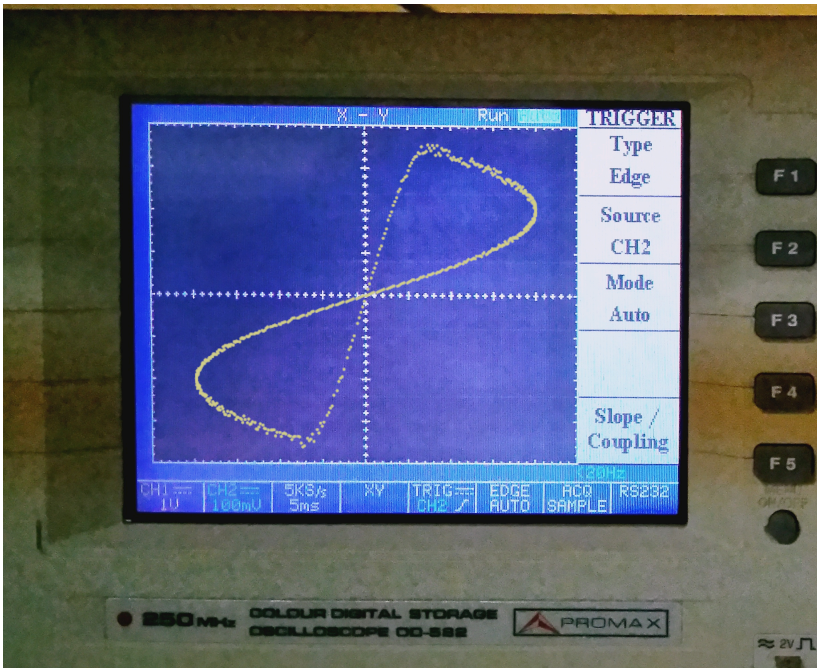


Figure 3.11: Snapshot of the oscilloscope showing system's I-V response to a 50 Hz input signal. The current was calculated as the voltage drop in a 1kΩ shunt resistor.

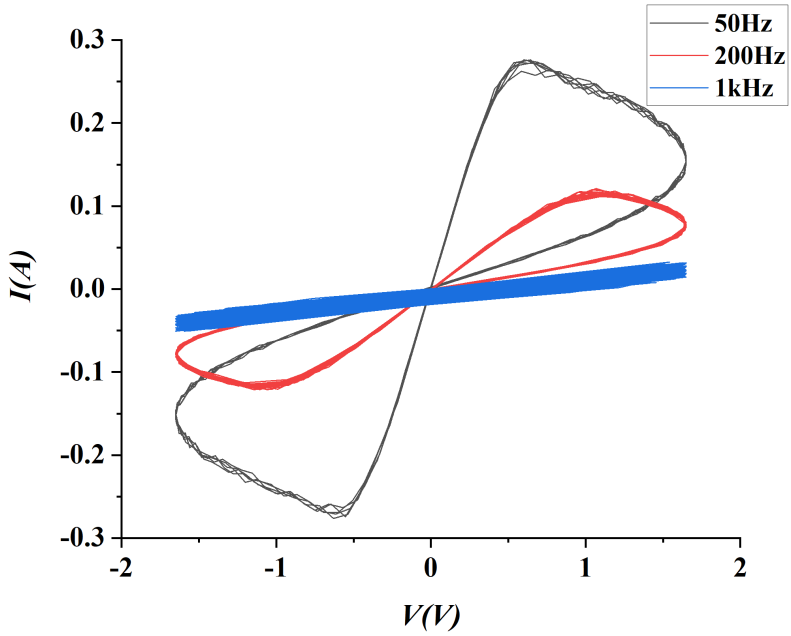


Figure 3.12: Response of the oscilloscope showing the I-V response to three different frequencies input signals.

shows an example of a low frequency input signal (50 Hz) response of the system.

The response of the system to three different frequency sinusoidal input signals is plotted in Fig. 3.12. From this figure becomes apparent that all three curves are passing through the origin (0 V, 0 A), thus they are being pinched. Also, the lobes become narrower with higher driving signal frequency, as expected for memristors [93,94]. As a comment, it can be seen that the signal at 1kHz already shows a lot of noise, which is caused by the closeness of the signal to the switching frequency.

Thus, this emulator exhibits a frequency-span up to some hundreds of Hz. This span is due to both the resolution of the PWM signal (set for this experiment at 10 bits) and the clock frequency of the FPGA (set at 12.5 MHz). Using 10 bits for the PWM signal means needing 1024 clock cycles for each PWM pulse, for a commutation frequency of 12 kHZ, which is a half to the maximum of the used switched. Thus, using a frequency of the input signal of 1 kHz is already too close to the limit.

3.5 A Stochastic Switched Capacitor Memristor Emulator

In this section, partially published in [35], we design, simulate and implement a mixed-signal memristor emulator, improving the versions presented in the previous section and in [33] and [34]. The proposed emulator consists of two blocks, taking advantage of the best features of each design part. In the analog block, a switched capacitor is used to implement a variable resistor, and in the digital one, that is, the control block, we use stochastic computation. The simulation is done with Matlab to implement the functionality of both the analog block, similar to that used in [32], and of the control block. For the experimental implementation, we have reused the one in the previous section, with the quadruple analog switch HCF4066FE and a DE0-Nano FPGA.

3.5.1 Theoretical Design

As mentioned above, our system has been implemented in two parts [32,34]. First, we implemented an analog system including the switched capacitor module (SC), as shown in Figure 3.7, whose equivalent resistance R_{eq} is described by Equation (3.16). In this case, both control external signals S_1 and S_2 are equal, with a lag of 180 deg. [103]. The second part is a digital module implementing the control part in stochastic logic, as will be discussed below.

For our design, in the charge flux domain, it is necessary to calculate the flux from the voltage of the terminals of the SC as a first step. Once this is done, then the relationships between flux and charge are used to obtain the duty cycle (D) that varies the equivalent resistance of the SC. The digital block is the responsible for all these steps.

For this purpose, a series of approximations shall be done to Equation (3.16). The conductance (G) ($G = 1/R_{eq}$) can be rewritten as:

$$G = f_S C \frac{e^{(-\frac{x}{2})} - e^{(\frac{x}{2})}}{e^{(-\frac{x}{2})} + e^{(\frac{x}{2})}} = f_S \cdot C \cdot tanh(\frac{x}{2}) \tag{3.20}$$

where x=DT/τ.

The use of a first order Taylor expansion of $tanh(x/2)$ allows us to get a simpler expression. For this, it is necessary to take into account that the decay time of the system is much longer than the control signal period. Thus, we can obtain a simpler equation describing the conductance G:

$$G = f_S C \frac{DT}{2\tau} \tag{3.21}$$

It is important to notice that this last equation implies that conductance is linearly dependent on the duty cycle D.

To calculate the flux, the digital block converts each voltage terminal of the SC (v_a and v_b in Figure 3.7) to non-correlated random values. Then, the corresponding value $v_a - v_b$ is accumulated into a counter, which acts as the integrator. Notice that since we are using stochastic computing, this up/down counter needs to count only one up ($v_a > v_b$), one down ($v_a < v_b$), or remain the same ($v_a = v_b$). To implement the memristor device, it is necessary to use an equation to describe the relationship between flux and charge. In this section, we use again the simplest relation proposed in Section 3.4:

$$Q = M\phi^2 \tag{3.22}$$

where M is a constant. This equation does not include any internal variables. Applying the fourth derivative of the equation, the conductance is:

$$i = 2M\phi\frac{d\phi}{dt} = 2M\phi v \implies G = 2M\phi \tag{3.23}$$

Matching Eq. ((3.21)) and Eq. ((3.23)), the relation between the duty cycle and flux is:

$$D = \frac{4M\tau}{f_S CT}\phi = K\phi \tag{3.24}$$

where K is therefore a constant value, depending on the specific system used.

To control the analog block, we use the switched capacitors; therefore, the duty cycle (D) must be used. The duty cycle is calculated by the digital block from ϕ according to Equation (3.24) as a stochastic value. To use it, the average value of D is calculated to determine R_{eq} with Equation (3.16).

The emulator block design scheme including the two parts of the design, analog and digital, is shown in Figure 3.13. The part corresponding to the digital block implemented in stochastic computing is shown as a circuit in Figure 3.14.

3.5.2 Simulation Results

In order to be considered as a memristor, the emulator must present two characteristic fingerprints [78, 94, 95]: (1) a pinched loop (2) whose area changes with frequency.

Figure 3.15 presents the $i - v$ curve of the emulator under inputs of different frequency using 16 bits for the stochastic representation. It is apparent from this figure that the curves are pinched at the origin and that the loop area changes with frequency. Thus, we can consider that the two fingerprints are present.

Because of the way it is constructed, the emulator reaches a saturation for the conductance. This is due to the maximum value of $D = 1$, and

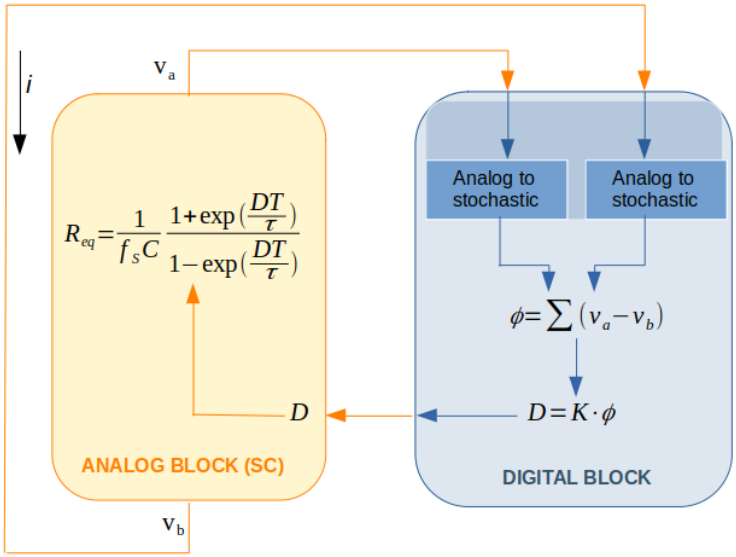


Figure 3.13: Swiched capacitor memristor emulator (SCME) block diagram.

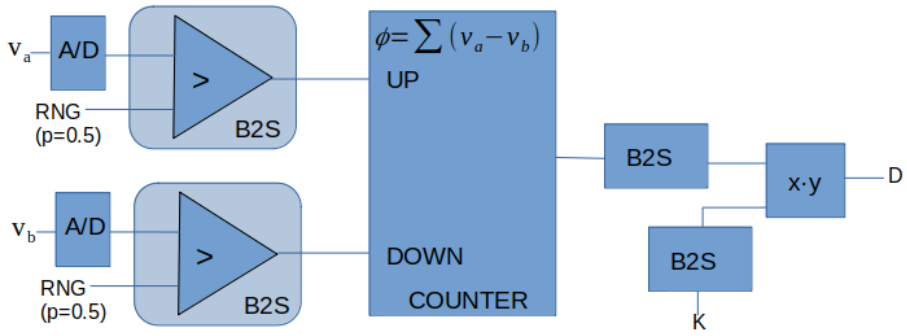


Figure 3.14: Control block implementation using stochastic computing.

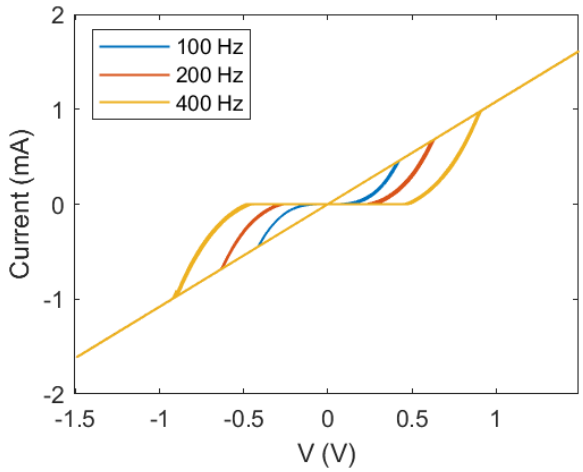


Figure 3.15: Simulated $i - v$ characteristic curves of the memristor implemented using Figure 3.13. Three different frequencies are shown in different colors.

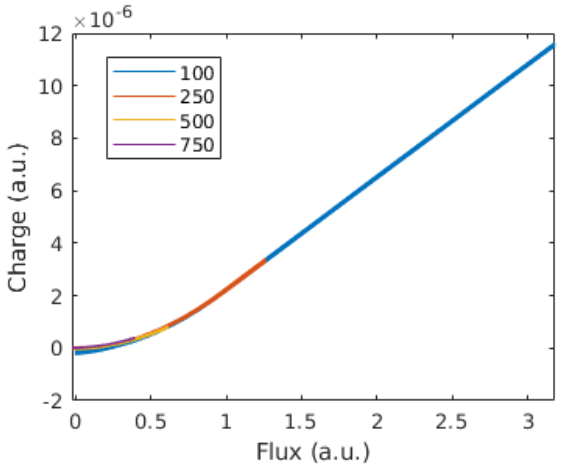


Figure 3.16: $Q - \phi$ characteristics of the memristor implemented using Figure 3.13. The different frequencies (in arbitrary units) are shown in different colors.

can be clearly seen at low frequencies, where the maximum value of flux is reached faster. This may also be seen in Figure 3.16, where the behavior of the Q versus ϕ near the origin is quadratic, as can be expected from (3.22), but it is also seen that its behavior changes to linear after a maximum value for $D = 1$ is reached.

It has to be noted that there is a small noise present caused by the stochastic nature of the system, as discussed above. This noise nearly disappears in the saturation, since the counter is practically constant, even though a small ripple is present caused by the stochastic internal behavior. This noise is greatly reduced in the charge and flux domain (Figure 3.16, because of the integration.

Finally, the current signal for different frequencies is shown in Figure 3.17. As can be seen there, the maximum conductance (related to the maximum value of the current) is lower for higher values of frequency, as expected.

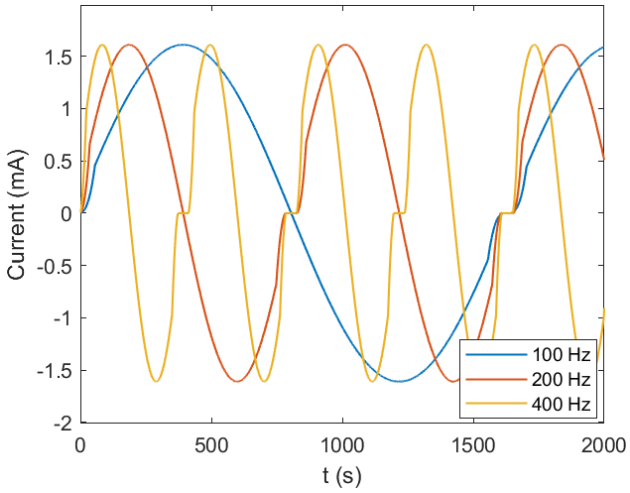


Figure 3.17: Current signal (response) for different frequencies, as obtained from the simulation. The three different frequencies are shown in different colors and correspond to the ones shown in Figure 3.15.

3.5.3 Experimental Setup

The implemented circuit is the same shown in Figure 3.8, reprogramming the FPGA used in Section 3.4. The conversion from analog to stochastic was performed by first converting from analog to digital using two of the on-board available A/D and then converting this digital value into stochastic, as described above. We have used 16 bits for the stochastic representation, and the needed random numbers were created using a public implementation of the Mersenne twister algorithm [104].

An AFG320 arbitrary signal generator was used to generate the input signal, while two oscilloscopes were used to monitor the full system. An oscilloscope monitored the control signals of the HCF4066FE, while the other oscilloscope was used to monitor the voltage through the shunt resistance of 1kΩ to obtain the current and also the input voltage, defined as the difference between the two input terminals.

3.5.4 Experimental Results

We have reused the same experimental system than in section 3.4. The system has been tested using different input frequencies: 100, 200 and 400 Hz. The internal behaviour of the circuit is depicted in Figures 3.18 and 3.19, which depict, respectively, the control signals S_1 and S_2 in one of these cases and the waveform corresponding to the three least significant bits of the counter.

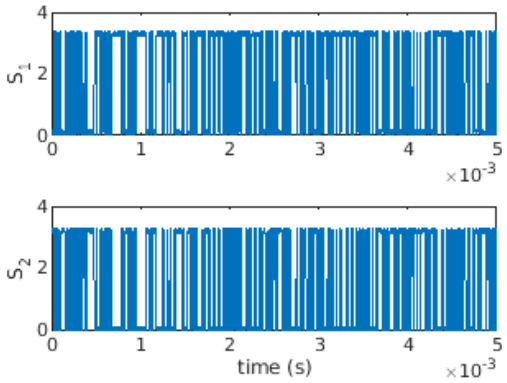


Figure 3.18: Stochastic signals S_1 and S_2 generated by the control circuit.

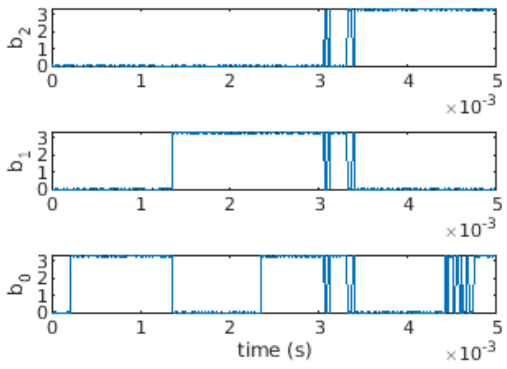


Figure 3.19: Three least significant bits of the counter (b_0 is the least significant bit) at a specific time.

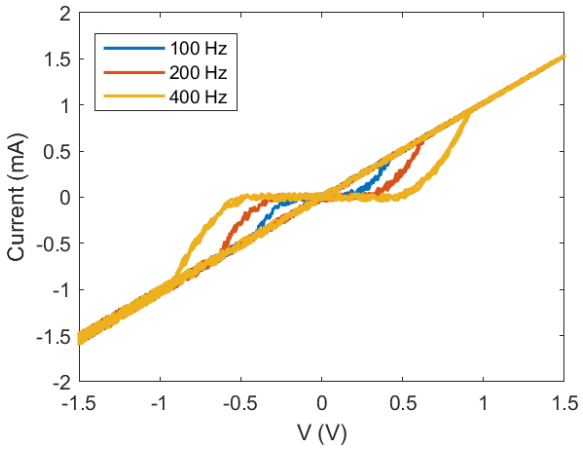


Figure 3.20: Measured I-V signals at different frequencies.

The experimental I-V loops are depicted in Figure 3.20. On the left figure, the three experimental I-V curves for the corresponding frequencies in Figure 3.15 (simulations) appear.

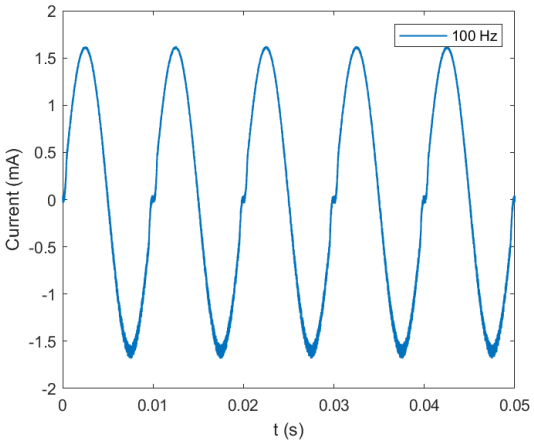
The temporal behavior of the current in these three cases is shown in Figure 3.21. The currents are clearly nonlinear because of memory: if they were nonlinear due to other effects, then they would be symmetrical, which they are not. In addition, they are showing a dependence on the frequency, as expected for a memristor.

It is apparent that, in all cases, the experimental fingerprint of a memristor, i.e., the pinched loop [95], is clearly demonstrated. This means that the device has a resistive behavior (it is pinched, which means no current when no voltage is applied), and that this resistance has a memory effect (there is a loop, which means that there are two possible values of the resistance and, hence, the current, for each voltage input value).

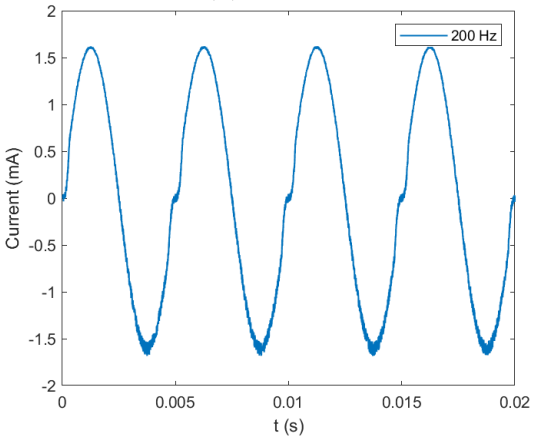
Finally, it has to be noted that the area of the loop changes with frequency, with the higher area corresponding to higher frequencies. This is caused by the saturation of the internal counter that corresponds to the flux integral (Equation (3.3) and Figure 3.14), which leads to a linear behaviour once the maximum value is reached.

3.5.5 Discussion

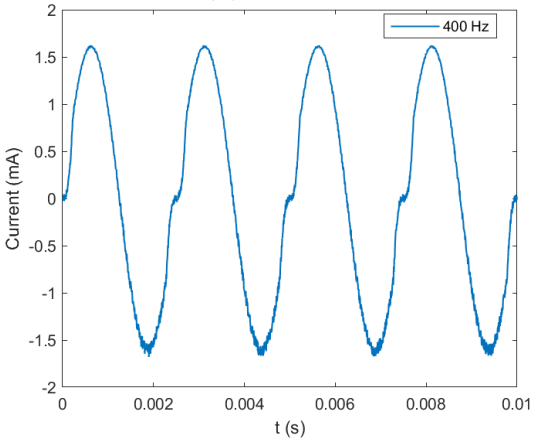
As discussed above, the design and implementation of memristor emulators is an active research field. In this thesis, we have made a contribution to this area by presenting the design, simulation, and experimental implementation of such an emulator. Our proposal is based on using switched capacitors to implement the variable resistor and on using stochastic computing to implement the control part.



(a) 100Hz



(b) 200Hz



(c) 400Hz

Figure 3.21: Temporal graphs of the measured response (current signals) of the realized memristor at 3 different frequencies corresponding to the simulated frequencies for driving sine voltage of (**a**) 100 Hz, (**b**) 200 Hz, and (**c**) 400 Hz.

The switched capacitor block has been implemented using standard off-the-shelf components with a maximum switching frequency of 25kHz. The control signals at this frequency are generated inside the control block, which has been implemented into a DE0-nano FPGA. The FPGA reads the analog inputs (the input voltage of the analog block) using its built-in AD converters.

As a first step, we have shown using MATLAB simulation that the design is sound and can implement a system showing the expected fingerprints of a memristor: a closed loop, pinched at the origin. Finally, we have experimentally implemented the design. This actual implementation has been tested using sinusoidal waveforms of different frequencies, and it has behaved as expected. The system shows the memristor fingerprints with noise induced by the switching, as expected.

Thus, the proposed emulator has been shown to perform with its expected behavior, being a promising alternative to be implemented as an IP block into IC designs, since it is a very simple design requiring a lower number of digital gates than similar designs using conventional arithmetic implementations. This implementation would allow the increase of the limiting factor of the switching frequency at 25kHz caused by the use of a discrete component, and would also proportionally increase the working frequency of the emulator.

Chapter 4

Applications

4.1 Introduction

In this chapter, we will treat some of the applications we have considered useful for being developed with memristors, within a stochastic computing environment. Specifically, the applications developed and we will present in this chapter are:

- Resolution of digital mazes using a memristor emulator.
- Real time Cellular Nonlinear Networks (CNNs).

Before a detailed presentation these two application paradigms a brief presentation of them is apposed.

4.1.1 Maze solver

Along the first part of the chapter, we develop the design and implement a system capable of searching for optimal paths using a FPGA. This FPGA implementation is based on a memristor emulator which is used as a delay element. The idea is implemented within a stochastic computing environment, by configuring the test graph as a memristor network and applying a parallel algorithm to reduce computing time and increase efficiency.

Beforehand we check the operation of the algorithm in Matlab and then we export it into two different Intel FPGAs: a DE0-Nano board and an Arria 10GX. In both cases we obtain reliable results quickly and conveniently, even for the case of a 300x300 nodes maze. This part of the chapter was partially published in [36].

4.1.2 Stochastic CNN

In the second and last part of this chapter, we propose the utilization of SC in designing and implementing a memristor-based Cellular Nonlinear

Network (CNN). This kind of networks are a concept introduced in 1988 by Leon Chua and Lin Yang as a bio-inspired architecture, capable of massive parallel computation. Later on, CNNs have been enhanced by incorporating designs that incorporate memristors, thus profiting from their processing and memory capabilities. In addition, SC can be used to optimize the quantity of required processing elements; thus providing a lightweight approximate computing framework, quite accurate and effective, though.

As a proof of the proposed concept, we present an example of application that combines the Matlab environment and a FPGA in order to create the CNN implementation. We have used the implemented CNN to perform three different real-time applications on a 512x512 gray-scale and a 768x512 color image: storage of the image, edge detection, and image sharpening.

Finally, it has to be pointed out that the same CNN has been used for the three different tasks, with the sole change of some programmable parameters. Results show an excellent capability with significant accompanying advantages, like the low number of needed elements further allowing for a low cost FPGA-based system implementation, something confirming the system's ability for real time operation. This part of the chapter was partially published in [38] and in [37].

4.2 Maze solver

Since ancient times humankind has tried to solve labyrinths or mazes. The paradigm of maze solving is found in the Greek myth of Ariadne who used a thread to help Theseus getting out of Minotaur's labyrinth. Today, maze resolution can have multiple applications, as in robotics, topology and many areas of science and technology [105–107].

Graph theory is used as an element to define the maze problem, where optimized path solving algorithms could then be applied. Some algorithms simply obtain an exit path, while others optimize it by finding the shortest one. One of the latter is the Dijkstra algorithm [108] that calculates all possible paths to reach a final node beginning from an initial one, and then compares the total cost of all of them, eventually keeping with the shortest. Notice that this algorithm is considered as good and efficient as all of its alternatives, quantum computing excluded [109], and requires a long computation time when dealing with complex graphs. To overcome this, parallel computing becomes a very good alternative in reducing computing time and further improve efficiency [110–112].

One of the trends in high performance computing is the use of arrays of memristors or memristive grid performing parallel computing. Memristors are resistive devices whose resistance depends on their dynamical history [113]. In fact, they can be thought of as variable resistances capable to remember their past; this is: memristors can be used as a memories [114], as

well as computation elements. One of the applications they have been used in is modelling the distance between nodes in a maze. In this approach, the shortest path between two points is corresponded to the current path with the minimum resistance [112], converting the problem to one determining the maximum current path.

Implementation and design of circuits with memristors requires extensive simulations when the number of devices involved is large like in memories or bio-inspired circuits [115]. Even though there are SPICE implementations of different models [116–120], in order to speed up simulations some researchers use digital, analog, or mixed-signal emulators, [32, 38, 78, 85, 121–124]. The use of these emulators can improve the simulation time, allowing the physical implementation of memristive circuits, while eliminating some undesired effects like the cycle to cycle variability appearing in ReRAMs [125, 126].

As mentioned above, in this first part of the chapter, we implement a fully digital system (under the acronym *GERARD*: *GE*neral *RA*pid *R*esolution of *D*igital mazes) that solves mazes in a digital environment by implementing the topology of those mazes as a grid of nodes in a programmable device (FPGA), which allows for parallel computing. In this proposal, the interconnections between the nodes of the grid are implemented using memristor emulators that are purely digital, as in [31]. Determining the minimum current path is achieved by mapping the distance between nodes to a memristance, which charges a fixed capacitor with a fixed voltage. This way, the time needed for charging the capacitor up to a given voltage provides with an estimation of the value of the memristance, thus the length of the path. In section 4.2.1 the general method is described, section 4.2.2 explains the algorithm implementation, section 4.2.3 presents the results, and finally the section 4.2.4 discusses the work.

4.2.1 General method

The proposed method for the maze solver is based on representing the maze as a matrix of nodes connected through memristors (Fig. 4.1) to its four nearest neighbors, forming a memristive grid as in the original work of Pershin et al. [112]. The main idea in that paper was measuring the current through the interconnecting memristors, getting this way the minimum current path. That method was based on the capability of memristors to be programmed to a given resistance value using a (relatively) high voltage, while during its normal operation it could be considered to hold its programmed resistance value.

That method had the problem of determining which was the minimum current path, which is a rather complex experimental problem. In our approach to this problem, we propose another novel method, based on measuring the time demanded for a grounded capacitor to be charged through a memristor, as in Fig. 4.1. This time obviously depends on the memristance

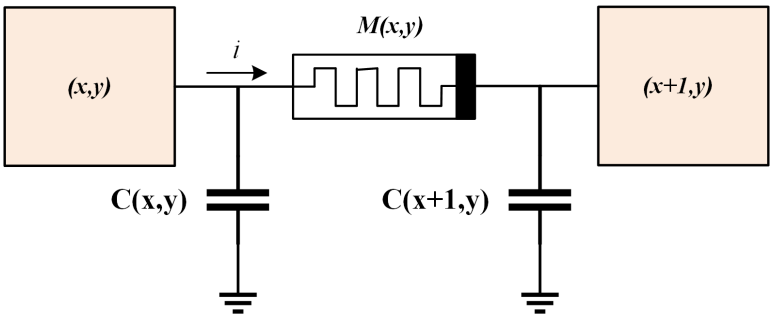


Figure 4.1: Scheme of the general interconnection pattern between two arbitrary adjacent nodes (x,y) and (x+1) of a NxM grid. Notice that for the sake of clarity only one of the four connections of each node is shown.

value (in fact, this is a dynamically changing resistance). Since the input is considered to be a constant current I_C fed through the memristor, its voltage V_C will be:

$$V_C = \frac{I_C}{C} \cdot t \tag{4.1}$$

where C is the value of the capacitor, and t is the time since the capacitor has started to charge, assuming an initial value for $V_C(t = 0) = 0$. The voltage drop through the memristor is $V_M = I_C M$, with M being the programmed resistance value of the memristor. The time t_C needed for these two voltages to equate is just:

$$t_C = MC \tag{4.2}$$

Thus, for a constant value of C, measuring t_C allows for directly estimating the value of M. This way the minimum time is used to calculate the minimum resistance current-path, instead of the maximum current. As a side comment, it is worth noticing that a similar effect could be achieved by using a complementary configuration with fixed resistors and memcapacitances.

The flow diagram of the algorithm is presented in Fig. 4.2. The time $t_{(x,y;x+1,y)}$ needed for a signal to propagate from an initial node (x, y) to a destination node $(x + 1, y)$ (Fig. 4.1), is calculated by Eq. (4.2). Once the signal propagates, the destination node is activated and performs a series of actions:

1. It stops listening to any other input, so no other signal can trigger it.
2. It identifies and stores the triggering input port.
3. It propagates the signal to all its non-activated ports.

When the final target node is reached, it sends a signal to the controller, which in turn, recovers the path. This is simply achieved by recovering the

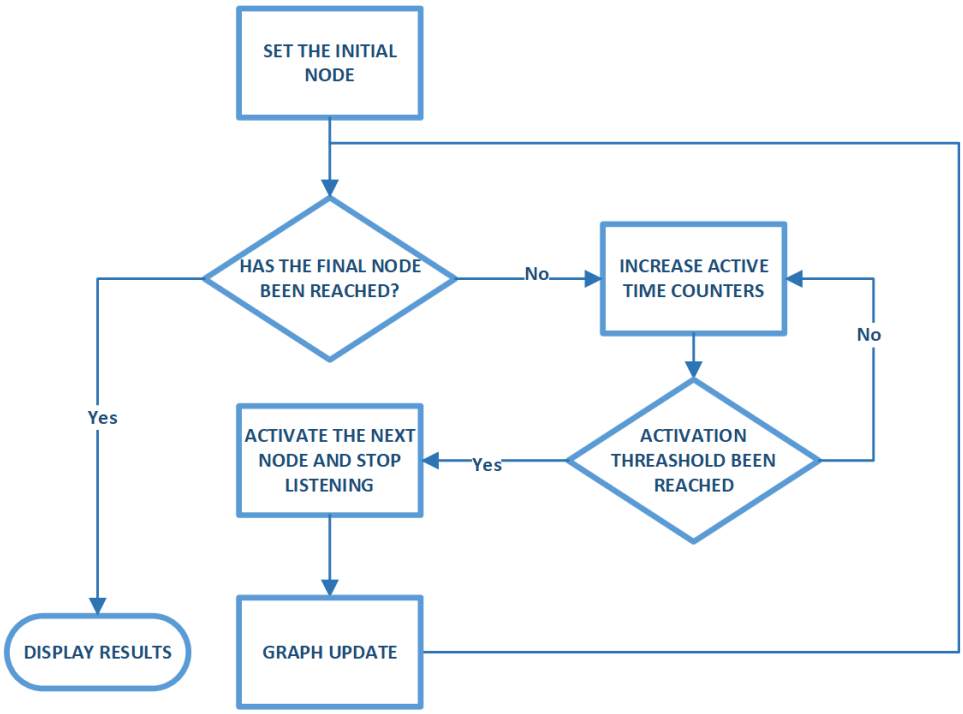


Figure 4.2: The flow diagram of the proposed shortest path algorithm.

activated input from each node, and working the way back from the final node to reconstruct the actual path.

Consequently, the time the algorithm needs to reach the solution is determined by the length of the path and the cost between the nodes in the path, as described in Eq. (4.3):

$$t_{total} = \sum t_{(x,y;x',y')} = C \sum M_{(x,y;x',y')} \tag{4.3}$$

where the summation is performed over the nodes (x, y) and (x', y') in the shortest path and $M_{(x,y;x',y')}$ is the resistance between them.

4.2.2 Algorithm Implementation

4.2.2.1 Memristor Model Implementation

There are many models proposed in the literature that reproduce the electrical behavior of memristors. Just for historical reasons, it is worth mentioning that the very first model was proposed in [127], in the same paper that revealed the discovery of actual memristors. It was based on ionic diffusion, and received many posterior improvements (see, for instance, [128–131], among others). Another approach is using charge and flux, as proposed in [93]. Following this approach, some models have also been proposed

[75, 76, 132, 133]. In any case, there are many models to be found in the literature (for a review, see [134], [135] or [136]).

Following the charge and flux paradigm, we have implemented a memristor model based on a purely digital emulator [31]. This emulator implements a simple relation between charge Q and flux ϕ:

$$Q = M(\phi)\phi \tag{4.4}$$

Memresistance $M(\phi)$ is also calculated with the simplest relation:

$$M(\phi) = M_0\phi \tag{4.5}$$

Full details of the implementation are provided in [31].

Note that only a minor modification was needed to fulfill the requirements for this application. This was adding a switch keeping constant the value of the memresistance, or allowing it to be programmed, as discussed in the section above. With this modification, once the memristor is programmed, it behaves as an element with a constant resistance.

4.2.2.2 Matlab Implementation

The operation of the system developed in Matlab implements the flow diagram appearing in Fig. 4.2, and performs the following operations:

1. Program all memristors with a memristance value M corresponding to the distance between nodes;
2. Set the starting point, taking into account that the bottom right element is the end by default (without any loss of generality);
3. Start counting with the first node and propagate the signal to its neighbours with a delay given by Eq. (4.2);
4. When a node receives an input signal, it is marked as active and treated as a new starting point;
5. Repeat from step no. 3 until the final node is reached;
6. If the end node is reached, a signal that the process is finished is sent to the control unit, and the shortest path is then retrieved.

It is noted that the Matlab implementation of the designed algorithm has been validated against the Dijkstra algorithm [108] up to an 8x8 matrix, providing exactly the same results. An example is shown in Fig. 4.3 for a simple 3x3 matrix, where the numbers between the nodes correspond to the distance between them. The green color defines the calculated shortest path (which is the same both by Dijkstra and the proposed algorithm).

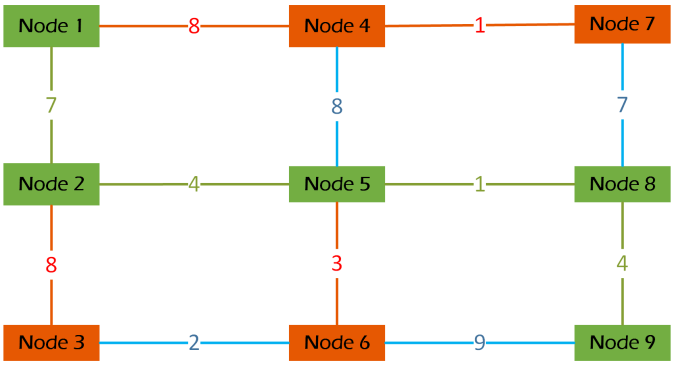


Figure 4.3: The solution in the case of a 3x3 example. Notice that the numbers between the nodes represent the resistance. The shortest path is the one passing through the green nodes. The initial node is node 1, and the final node is 9.

4.2.2.3 Programmable Device Implementation

The FPGA implementation of this novel maze solver is divided into three different parts, namely: the control system, an interconnection element including the memristor emulator, and the nodes themselves. It is illustrated in Fig. 4.4, while the set of instructions implemented to control the system is shown in Table 4.1. Note that the number of cells and memristor-blocks depends on the grid size, since there is an one-to-one correspondence between the physical modules and the maze net.

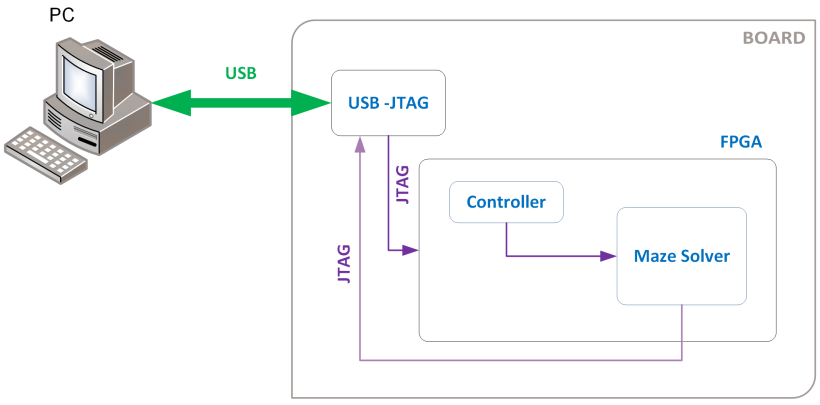


Figure 4.4: Illustration of the system, including the PC running the external software, the USB-to-JTAG interface on the electronic board, and the FPGA, where the general controller and the maze solver are located.

Table 4.1: Set of instructions for the control system. Notice that instructions 010 and 011 need additional arguments (target row and column) to function.

Code	Description	Parameters
001	Reset all the internal registers	–
010	Program the value of a memristor	Row, column
011	Set the starting point	Row, column
100	Start the process	–
101	Get the calculated path	–

4.2.2.3.1 Communications Block

This block is responsible for the communication between the programmable device and the external systems, in this case a PC. The communications part between the computer (Matlab) and the FPGA was implemented using a JTAG interface, as in [137], where a full description of the procedure is provided. This part consisted of two standard blocks: the vJTAG component and the vJTAG-interface, as shown in Fig. 4.5.

The vJTAG component is responsible for receiving the information and injecting it into the vJTAG-interface. The later saves the data received in a register and then sends its contents to the other components of the system through a dedicated bus. In addition, these components were responsible for sending the result back to the user. The interface between the user and the maze solver in the FPGA was implemented in Matlab. This uses TCP/IP with a dedicated socket [137], and was responsible for converting

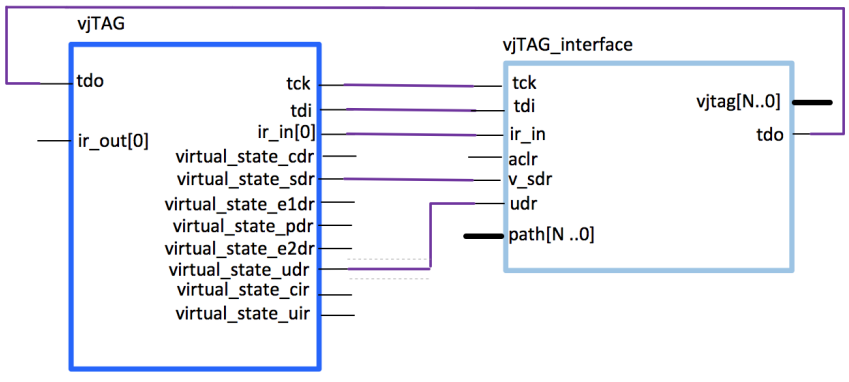


Figure 4.5: The connection between the vJTAG blocks in the communication module, as discussed in [137] and [138]

.

user-commands to binary code. It was also receiving back data from the FPGA, displaying it accordingly.

4.2.2.3.2 Interconnection Block

This block implements a delay equivalent to the cost needed to traverse it. As mentioned above, this delay module (shown in Fig. 4.6) implements a memristor-capacitor emulator, as in Fig. 4.1. The system can be programmed to a given delay by setting the memristor to an equivalent value provided by Eq. (4.1) and the actual cost of the maze.

Once the memristor is programmed, and one of the input ports of the memristor has been activated, the capacitors are charged using a constant voltage input V_s until a threshold value is reached. For a known value of the capacitor and the memristor, this would be equivalent to determining the value of the current used to reach the threshold value.

We have used the memristor emulator implemented in [31] as described above, and considered it to be connected to a capacitor at each end, which was initially connected to ground. The capacitor has been implemented as an accumulator with input i_C and output v_{Co}. Moreover, the equations have been simplified by setting all the constants of the system to 1, without any loss of generality. At each clock cycle, $v_{Co}(t)$ would be updated as:

$$v_{Co}(t+1) = v_{Co}(t) + i_C C \Delta t \tag{4.6}$$

This block will then propagate the signal to the other end by activating the out terminal, when $t = M$, as determined by Eq. (4.2). Notice that the

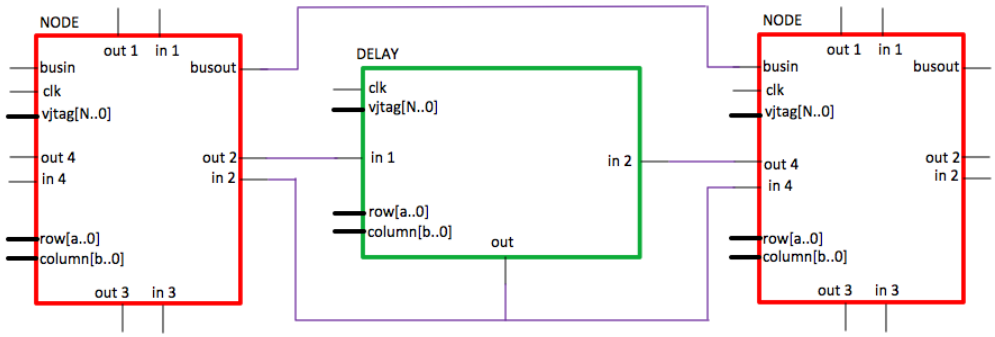


Figure 4.6: The equivalent FPGA implementation of Fig. 4.1, with the nodes and their interconnection. The block labelled *DELAY* is the equivalent of the memristor-capacitor elements, while the *NODE* block corresponds to the (x,y) node elements. All these blocks also show the additional inputs that allow external programming and data recovering, as discussed in the text.

block must only accept the first input that reaches it (either *in1* or *in2*), and any subsequent input signal has to be disregarded. In this module, the numbered inputs and the output are connected to the nodes, while all other entries follow the same logic as the Node. This module uses the *clk* fall edge to create a small delay between the components, allowing the Node to update its signals before the *Delay* starts to work. In addition, inside each node and memristor there was also a control unit connected to the vjtag[N..0] bus input, implemented as a state machine, with the corresponding part of the set of instructions shown in Table 4.1 that allows it to be programmed to the initial value. In order to use a single template component, we have also added two inputs setting the block row and column that identifies the block for programming purposes.

4.2.2.3.3 The Node Element

The Node components represent each of the network-nodes and are connected according to the pattern established previously in Matlab. These interconnections were made using the intermediate components that generate the signal delay, i.e. the memristor and capacitor emulator described above. Notice that each block has four *in* and four *out* terminals, numbered clockwise from the top, that connect to the delay block as shown in Fig. 4.6.

The numbered inputs and outputs of the Node block are used to form an (NxM) graph, corresponding to the actual maze, and are connected to the delay modules. We repeat that the maze path is defined by the interconnections between these nodes. The vJTAG entry is the input for the data sent by the communications module through a shared, read-only bus, whereas the row and column entries mark the position of the component within the system for programming purposes. The modules accepts the corresponding programming instructions appearing in Table 4.1. These instructions allow the user to use the communications block to set the starting point at a specific node, defined by its position, and, once it is set, to start the process of finding the shortest path between this node and the (N,M) node. Finally, the *busin* input and *busout* outputs are connected between each pair of nodes to return the path through the JTAG interface.

4.2.3 Results

Having in mind the global operation that has been described, this was implemented as follows: when the initial Node gets activated by the user, it activates its four outputs in order to start the counter of all four Delays connected to it; then, when the assigned weight value has been reached, the Delay activates its output, resulting in turning on the Node on the opposite side.

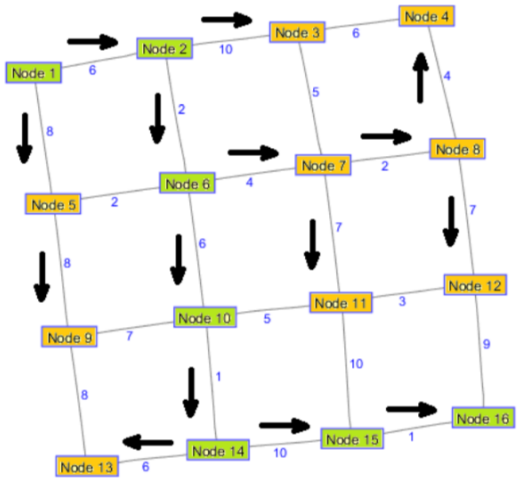


Figure 4.7: The FPGA returned result in the case of a 4x4 maze. The corresponding recovered shortest path (nodes in green) appears, showing the directions in Fig. 4.8, according to Table 4.2 .

Table 4.2: Incoming signal direction codes.

Code	Description	Code	Description
000	Initial node	111	Final node
001	Above	011	Below
010	Right	100	Left

An activated Node saves in its internal register the one out of the four entries that launched the activation, stopping at the same time to listen to the other entries, which are now transformed into outputs. Then, the node sends a pulse through these new outputs, that propagates in the same fashion until the end node is reached. Applying this approach, there are several counters running in parallel, achieving the goal of reducing calculation time, thus, improving the overall efficiency. As has been explained above, then each node stores the information of the direction from where the first pulse reached it. This information is sent through a bus connecting each and every node up to the communications module. The later concatenates each of the bits it receives, forming a N-bit vector that is sent to the user via the vJTAG interface (*path[N..0]* input).

Initially, we checked the proof of concept of the proposed algorithm using the implementation of a 4x4 maze-example, as shown in Fig. 4.7. In this case the system needed 16 Node and 24 Delay modules, to implement the desired grid into the DE0-Nano FPGA board, using a 49-bit vJTAG vector and 3-bit row and columns vectors.

"000"	"100"	"100"	"011"
"001"	"001"	"100"	"100"
"001"	"001"	"001"	"001"
"010"	"001"	"100"	"100"

Figure 4.8: Array obtained from FPGA for the example graph, indicating the first activating input for each node.

In this example, the memristive grid defining the maze was initially described in Matlab, and then programmed into GERARD through the dedicated interface. As described above, once the solver was programmed, the signal propagated to the end node and the system returned a signal, which in our case was a 49-bit vector containing the direction of the incoming signal for each node according to the Table 4.2 code. Notice that each node used three bits and these were reorganized in a 4x4 array as shown in Fig. 4.8. Once the result vector was obtained, the user interface decoded it by working its way backwards, from the end node back to the initial node, obtaining the result shown in Fig. 4.7. In this specific case (a 4x4 matrix), the design required a total of 6142 elements (28 % of the total), with 6033 combinational functions (27 %) and 3274 dedicated logic registers (15 %), using a clock frequency of 50 MHz.

Finally, another example was implemented into an Arria 10 GX 220 FPGA card at 200 MHz using the Matlab FPGA-in-the-loop (FIL) methodology. In this example the FPGA has been used to speed-up the parallel calculations, and a 300x300 maze was generated, using a total of 196840 logic elements (89 %), 68654 ALM (85 %), 301368 registers (93 %), 10442 Kb of M20K memory (88 %) and 1612 Kb (95 %) of the MLAB memory. The resistance between the nodes was generated using 150 2D-Gaussian distributions with random position, dispersion and height, as shown in Fig. 4.9, where the colors represent the cost. In this same Figure, the red line depicts the shortest path, as returned by the algorithm between the start (left, bottom) and the end points (top, right). The time needed for a full Matlab implementation (with no FPGA) to solve the circuit was around 1900 s, while the FIL version needed only 82ms to solve the maze, for a total cost of 81630 (the total resistance, in arbitrary units) for a path length of 608 cells. Retrieving of the shortest path required thus 1800 bits. A comparison with the Dijkstra algorithm was not possible using the Matlab built-in algorithm, since it ran out of memory.

4.2.4 Concluding Remarks

In this part of the chapter an inherently parallel computation algorithm has been described and demonstrated. It was initially designed in Matlab, then implemented in a FPGA using a memristor emulator and returned reliable

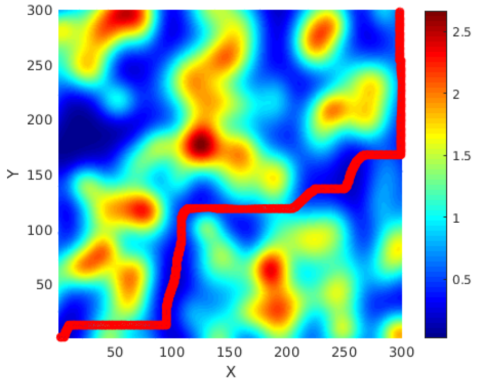


Figure 4.9: An illustration of the results returned in the case of a 300x300 maze, used for testing the implementation of the solver, appears. The colors represent the cost, which is the resistance between paths, as indicated in the right bar in arbitrary units, while the red line shows the returned shortest path.

results, equivalent to those obtained using Dijkstra's algorithm. It took profit of the study of multiple paths in parallel, showing how GERARD helps Ariadne to determine the way out of a maze. Two different examples were demonstrated, one with a 4x4 matrix, and another using a 300x300 matrix, both working in a very straightforward way.

The proposed design is simple and easy to scale up for implementing different graph configurations and has been checked with many other examples and using Dijkstra's algorithm [108]. Scalability of the system is limited only by the size of the FPGA. Overcoming this, a proper partitioning scheme could be also utilized. Finally, once actual memristor devices are finally out as a mainstream technology, they could be actually used to implement the proposed maze solver, paving the way for their use in autonomous robotics, among other possible fields.

4.3 Stochastic Computing-based Cellular Nonlinear Networks

Cellular Nonlinear Networks (CNN) were introduced by Chua and Yang [139] in 1988, and can be described as a mixture between Cellular Networks and Artificial Neural Networks that can implement a parallel processing universal computer machine. This bio-inspired architecture is able to process in parallel massive amounts of data, thus becoming suitable for image processing, with single ASIC CMOS prototypes already implemented being able to deal with rates up to $3 \cdot 10^4$ frames per second [140].

On the other hand, memristors have been proposed as a device that may

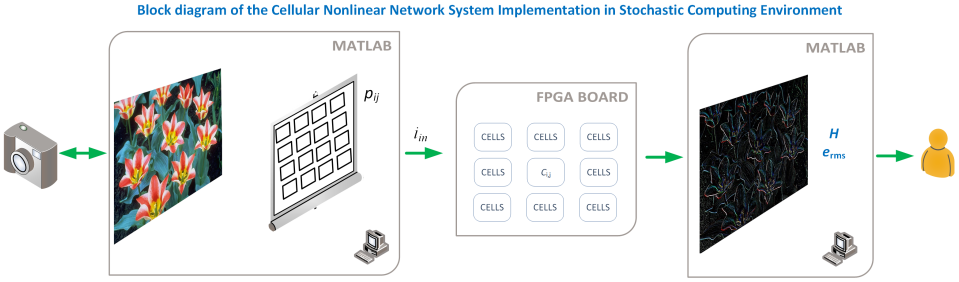


Figure 4.10: Conceptual depiction of the system, showing the tasks assigned to Matlab and those performed by the FPGA. The FPGA and Matlab are used jointly by using the FPGA-in-the-loop tool from Matlab, where the VHDL code is automatically generated, uploaded, and integrated with the main script at the computer.

help to implement this kind of circuits ([141,142]), but experimental implementations are still lacking. These devices, memristors, are passive, two-pole elements also introduced by Chua in 1971 [113], as a theoretically possible basic circuit element. In 2008, Strukov et al. [127] realized their ReRAM devices were, actually, a kind of memristor. There have been many groups dedicated to create either devices or emulators, ever since. One of the more classical mathematical memristor description, including memconductance G can be written as:

$$i(t) = G(Q) \cdot v(t) \tag{4.7}$$

where Q (also known as *charge*) is the integral over time of the current i:

$$Q(t) = \int^{t} i(t)dt \tag{4.8}$$

Notice that the requirement for the device to be a memristor is mapped to the requirement for the characteristics of the device to be dependent on some internal variables, as will be further discussed below, plus some fingerprints [92,94].

One of the main problems for using memristors into circuits is that they are not yet readily available for implementation in usual technologies. In this thesis we use Stochastic Computing to implement a fully digital realization of a CNN using memristors. To do so, we have used a memristor emulator presented in [33], as well as a Stochastic Computing implementation of a CNN using it, as in our previous work [38], where a simpler implementation was presented operating exclusively on gray images.

The full system we have implemented is depicted in Fig. 4.10. The first part is processed in the computer, where the images are read using a Matlab script and converted to gray scale if needed. The resulting image is then sent

to the FPGA board, which is used as an accelerator and connected using the FPGA-in-the-loop methodology that allows to integrate it in a seamless way. The FPGA does the processing using a massively parallel implementation of the proposed Stochastic Memristive Cellular Nonlinear Network, and the result is then read back into the computer and represented to the user, along with the entropy of the image and the rms error if needed.

This part of the thesis is structured as follows: after this introduction, the basics of memristors and Cellular Nonlinear Networks are presented in section 4.3.1, which is being used in section 4.3.2 to implement the basic CNN cell in Stochastic Computing. Section 4 presents the results obtained using three different pictures (two gray, one color) with three different sets of parameters. These three sets of parameters allow the CNN to perform three different operations on the images: storing, edge detection, and image improvement. Finally, section 4.3.3 concludes the chapter.

4.3.1 Memristive Cellular Nonlinear Networks

4.3.1.1 Memristors and memristive systems modeling

Among the possible theoretical descriptions of memristive systems, Corinto *et al*. present in [93] a very complete framework to study memristors and, in general, systems that may demontrtate memristive behavior. They utilize both the classical description, using voltage and current, and the flux–charge (φ-q) approach.

The memristive systems can be classified according to how far they are from ideality. Following the taxonomy proposed in [92], there are three distinct possibilities: the ideal, the generic, and the extended memristor. This extended categorization was a theory requirement, needed to cover the description of pinched, hysteretic behaviours found in numerous new various elements.

The most general class of memristors are the extended memristors. The memristors belonging to this class are described by extra internal state variables (in addition either to current and voltage, or to φ and q). As has been discussed before, we repeat here the next equations ((4.9)) to ((4.11)) which implement the case of flux-controlled memristors, in order to make the reading of this thesis easier :

$$i = G(\varphi, v, \mathbf{x}) \cdot v \tag{4.9}$$

$$\dot{\mathbf{x}} = \mathbf{g}_{\varphi}(\varphi, v, \mathbf{x}) \tag{4.10}$$

$$\dot{\varphi} = v \tag{4.11}$$

This way, an extended memristor has a memristance M represented by the nonlinear memconductance G (or, more accurately, its inverse) in Eq. ((3.1)), where φ is the flux, and v is the voltage between the terminals of the memristive device. The extra variables are grouped into the vector $\mathbf{x}$, and they may comprise different physical magnitudes depending on the specific memristive system; as examples, we can mention the radius of a conducting filament, the internal temperature of the system, as well as other non-electrical variables that may be used to describe the state of the memristor. These state variables $\mathbf{x}$ present a dynamic behavior described by g_φ and Eq. (3.2). As a side comment, it is worthy noticing that all the devices described as being memristors are indeed extended memristors.

When no parasitic effects are present, those extended memristors are better described as generic memristors (or, simply, memristors), since function g_φ depends only on the state variables $\mathbf{x}$ and φ. Ideal memristors, finally, are those corresponding to the original definition [113], and can be considered in this framework as generic memristors with no state variable dependence other than charge or flux.

As an example, let's consider the simplest memristor model that can be devised, similar to those discussed in [93] or, for real devices, in [74,143–145]. In this case, the model of an ideal memristor where the memresistance or the memconductance depends only on the charge or the flux can be written as:

$$G = G_0 \left(1 + \frac{\phi}{\phi_0}\right) \tag{4.12}$$

where G_0 is the unperturbed conductance, and ϕ_0 includes the importance of the memristive effect. Notice that for $\phi_0 \to \infty$, the behavior tends to be similar to that of an ideal resistor. The behavior of the device is represented in Fig. 4.11, for $G_0 = 0.1\ mS$, $\phi_0 = 10$, and three different frequencies. Notice how the device reproduces the two fingerprints of a memristor [92,94]: it presents a pinched loop whose area tends to zero at high frequency (green line, 'x' symbol).

4.3.1.2 Cellular Nonlinear Networks

The systems discussed in this thesis, Celullar Nonlinear Networks (CNNs) [139], are not to be confounded with Convolutional Neural Networks (also CNN), even if they share the same acronym. The CNNs discussed here represent a powerful massively parallel, multivariate signal processing paradigm. In their most basic description, they are made of independent processing units, called *cells*, where each cell has an input, an output that is fed back as another input, and also feels the effect of the inputs and outputs of its nearest neighbors. These effects are then processed internally into a state

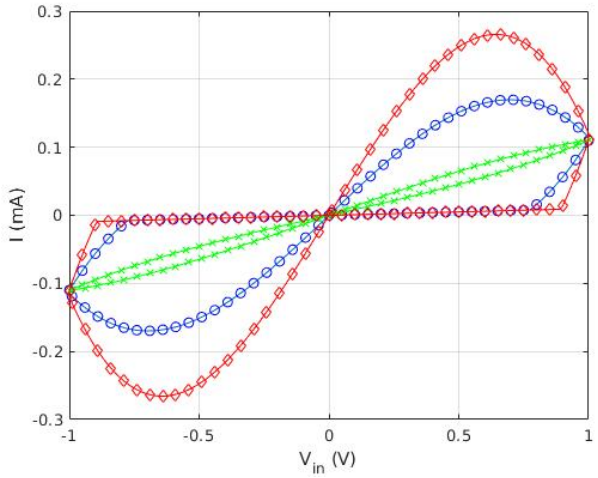


Figure 4.11: Representation of the I-V characteristics of the memristor defined by Equations (4.7) and (4.12) for $G_0 = 0.1\ mS$, $\phi_0 = 10$, and three different frequencies ($\omega(red\ \diamond) < \omega(blue\ o) < \omega(green\ x)$).

variable, and the output is linearly dependent on the result of this processing, with a positive and negative saturation.

As an example of hardware implementation of a CNN, we find [140], where each processing element typically accommodates additional data storage units, which allow the CNNs to store the programming parameters at the cell level. As a result, these Universal Machines (UMs) can be considered as one of the earliest examples of a non-von Neumann computer. Unfortunately, these memory blocks need a large integrated circuit (IC) area to be implemented, which increases significantly the size of each cell. As a consequence, the spatial resolution is quite poor compared with simple image sensors, which is a common problem that CNN-UMs and, as a inherited problem, arrays comprising sensor-processor cells based upon them, suffer from.

Mathematically, the behavior of the i, j cell can be described by a differential equation as:

$$\begin{aligned}\frac{dx_{ij}}{dt} = &- x_{ij} + a_{0,0} f(x_{ij}) + z_{ij} + b_{00} u_{ij} \\ &+ \sum_{k,i \in N_{i,j}, k \neq i, l \neq j} a_{k-i,l-j} y_{kl} \\ &+ \sum_{k,i \in N_{i,j}, k \neq i, l \neq j} b_{k-i,l-j} u_{kl}\end{aligned} \tag{4.13}$$

where x_{ij} is the state variable, u_{ij} and y_{ij} are the inputs and outputs, respectively, and a_{ij} and b_{ij} are the *feedback* and *feed forward* coefficients.

The stability of the system can be controlled by setting the value z_{ij} to an appropriate value [142], and is equivalent to a constant bias in the electrical equivalent. Function $f(x)$ is a nonlinear function, saturating to a minimum and a maximum value (v_{min} and v_{max}). In this sense, it is similar to the output function of a neuron. However, the most usual shape for it is a piecewise linear function, defined as:

$$f(v) = \frac{1}{2}\left(|v + v_{sat}| - |v - v_{sat}|\right) \tag{4.14}$$

Equation (4.13) is usually rearranged as:

$$\begin{aligned}\frac{dx_{ij}}{dt} =& g(x_{ij}) + z_{ij} + b_{00}u_{ij} \\ &+ \sum_{k,l\in N_{i,j},k\neq i,l\neq j} (a_{k-i,l-j}y_{kl} + b_{k-i,l-j}u_{kl})\end{aligned} \tag{4.15}$$

$$g(x_{ij}) = -x_{ij} + a_{0,0}f(x_{ij}) \tag{4.16}$$

Notice that, even if the sum is made over the whole set of integers, we usually restrict ourselves to just the nearest neighbors. This Eq. (4.16) can be implemented with a single element thanks to the unique non linear behavior of memristors. Thanks to this capability, to process or store data within a common physical nanoscale medium, their use in future CNN cell designs may allow to remove the burden of extra memory blocks within each processing element, allowing the development of co-located sensor-processor arrays with enormous high resolution levels, specially suited for the Internet-of-Things (IoT) industry.

Referring to the determination of the coefficients, some methods have been published [146,147] for the traditional CNN, and , more recently, using the so-called Dynamic Route Map (DRM) [141, 142, 148, 149], to the first-order approximate model of each cell of a Memristor CNN (M-CNN). Such an approximation is depicted as a circuit in Fig. 4.12, and the evolution of the state variable (x_{ij}) is described by an equation equivalent to Eq. (4.15):

$$\begin{aligned}\frac{dx_{ij}}{dt} = k(x_{ij})\,[\;\; &\theta(v_{x;i,j}f_{+}^{p}(x_{m;i,j}) \\ &+\theta(-v_{x;i,j}f_{-}^{p}(x_{m;i,j})\big]\end{aligned} \tag{4.17}$$

where i ∈ 1,..., M, j ∈ 1,..., N, $\theta(x)$ is the unit step function, $v_{x_{i,j}}$ stands for the voltage across the capacitor C_x, whereas $xm_{i,j}$, and $v_{m_{i,j}} \equiv v_{x_{i,j}}$ denote, respectively, the state and voltage of memristor m_x, whose current is described via the generalized Ohm's law from Eq. (3.1): $i_{m_{i,j}} = G(x_{m_{i,j}})v_{m_{i,j}}$, with the memductance given by $G(x_{m_{i,j}}) = x_{m_{i,j}}^{-1}$. The nonlinear function proposed in [149] to characterize the memristive CNNs are:

$$k(v_{x_{i,j}}) = -\beta v_{x_{i,j}} + \frac{\beta - \alpha}{2}(\left|v_{x_{i,j}} + V_t\right| - \left|v_{x_{i,j}} - V_t\right|) \tag{4.18}$$

with $\beta > \alpha \in \Re^+$, featuring units $\Omega V^{-1} s^{-1}$, and $V_t \in \Re^+$ denoting the minimum voltage needed by the memristor for switching. In addition, there are two different window functions, to ensure that the memristor stays in between the two possible states $[x_{on}, x_{off}]$. These two window functions can be written in a compact way [149] as:

$$f_r^p(x) = 1 - \left(\xi + \frac{x - xon}{x_{off} - xon}\right)^{2p} \tag{4.19}$$

where $\xi = -1$ when $r =$ " + " (the upper boundary), and $\xi = 0$ in the opposite case; p can be any integer ($p \in \mathbb{Z}$). In addition, the dynamical evolution of the voltage $v_{x_{i,j}}$ of each cell in system presented in Fig. 4.12 is:

$$\begin{aligned} C_x \frac{dv_{x_{i,j}}}{dt} = &-(\frac{1}{R} + G(x_{m_{i,j}})) \cdot v_{x_{i,j}} + a_{0,0} f_{out}(v_{x_{i,j}}) \\ &+ i_{bias} + b_{0,0} v_{in_{j,k}} \\ &+ \sum_{k,l \in [-1,1]} \left(b_{k,l} v_{in_{i+k,j+l}} + a_{k,l} v_{out_{i+k,k+l}}\right) \end{aligned} \tag{4.20}$$

$$f_{out}(v) = \frac{R_y g_{lin}}{2}\left(|v + v_{sat}| - |v - v_{sat}|\right) \tag{4.21}$$

Notice the equivalence of Eq. (4.20) with Eq. (4.15): the last line of both equations is equivalent, while the first line of Eq. (4.15) corresponds to the first two lines of Eq. (4.20). Thus, we can conclude that the circuit in Fig. 4.12 accurately represents a possible implementation of a CNN cell, and it is the circuit we will implement in the next section as a SC module.

4.3.2 M-CNN Stochastic Computing

4.3.2.1 Stochastic Computing Implementation of a Memristor Emulator

The described advantages of SC framework were used by a stochastic computing implementation of a memristor emulator [33], which describes the memristor using equation (4.12), and was written in the form of equations (4.22)-(4.24), so that it could be implemented in a discrete way. The mathematical operations eqs. (4.22)-(4.24) were carried out by simple digital gates. A simple manner to model memristors is using a linear relation of the charge Q with the memconductance $G(Q)$, with an upper and a lower value, G_{max} and G_{min}, correspondingly. Then according to this approach:

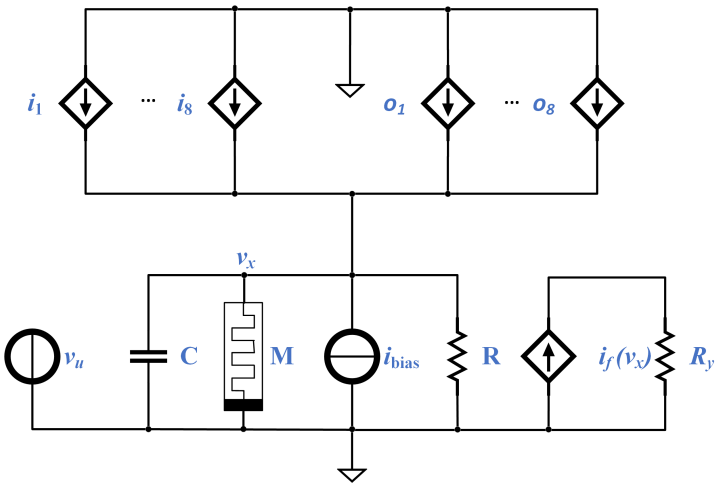


Figure 4.12: Schematics and Stochastic implementation of a CNN cell, the processing element in cell C(i,j) (i∈{1,...,M} , j∈{1,...,N}). The other elements (resistor, memristor, and capacitor), present the same values from cell to cell, i.e. Cxi,j=Cx , mxi,j=mx , and Ryi,j=Ry . Adapted from [141].

$$G_1(Q) = G_0 + G_1 \cdot Q \tag{4.22}$$
$$G_2(Q) = min(G_{max}, G_1(Q)) \tag{4.23}$$
$$G(Q) = max(G_{min}, G_2(Q)) \tag{4.24}$$

It is apparent that now Stochastic Computing can be used to implement such a model within a digital environment (an FPGA, or an ASIC). Other complex, physically-based models simulating memristive behavior in FPGAs can be found in the literature [85, 86], but they are very mathematically complex models, requiring a very large number of gates.

The emulator discussed above presents all the standard fingerprints of a memristor as required by the theory [94]. This implementation appears as a block diagram in Fig. 4.13, where v_x and GND are the SCN values of the positive and negative terminals of the memristor, respectively, while the calculated current is represented by the stochastic value i_M. The SCN value of GND has to be represented by a probability of 0.5 for an "1", since we are mapping the interval [0..1] to an interval that includes negative and positive values. In our case, we have used a public version of the Mersenne twister algorithm, available from GitHub [104].

Rewriting equations (4.22)-(4.24) to allow them to be implemented in discrete time results in:

$$Q = \int^t i(t) \approx \Delta t \cdot \sum_j i(t = j \cdot \Delta t) \tag{4.25}$$

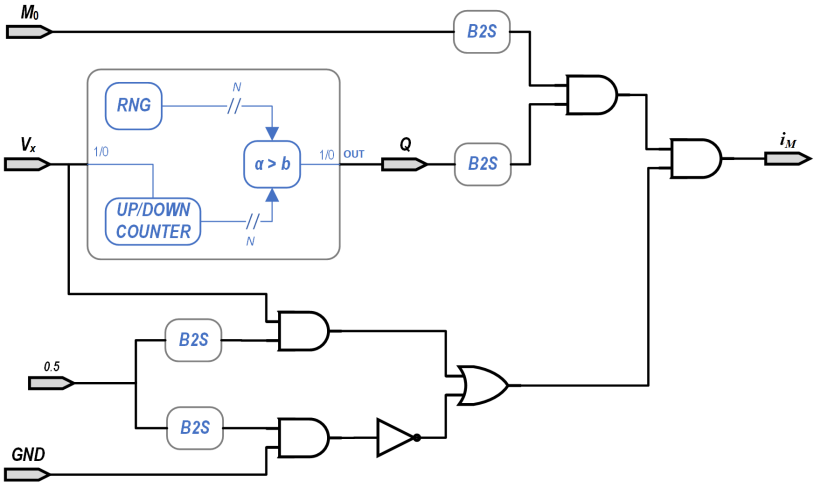


Figure 4.13: The proposed Stochastic Computing memristor implementation. Inputs v_x and GND are the SCN values of the positive and negative inputs of the memristor respectively, and i_M is the calculated SCN value of the current.

where the integration step is Δt. Using Eq. (4.25), we can rewrite (4.22) as:

$$G_1(Q) = G_0 + G_1 \cdot \Delta t \cdot \sum_j i(t = j \cdot \Delta t)S \tag{4.26}$$

The adder needs to increase or decrease its output by a unit, depending on the inputs, since i and GND are both SEN. An increasing is performed when $v_x = 1$, while when $GND = 1$ the output is reduced. A single constant is used to groupe M_1 and Δt, and the max and min functions are built into the adder by establishing a maximum and a minimum values. A SEN output is generated from the adder's output by comparing the it to a random number spanning $[0..(2^{NB} - 1)]$. An AND gate is then used to obtain the current as per Eq. (4.7), as in Fig. 4.13.

4.3.2.2 Stochastic Computing implementation of a M-CNN

As discussed above, a very compact implementation of a single cell of a Cellular Nonlinear Network single cells can be performed as in Fig. 4.12 [141]. The output voltage $v_{y;i,j}$ presents a nonlinear dependence on the internal voltage $v_{x;i,j}$. The dynamic behavior of this v_x is governed by a differential equation:

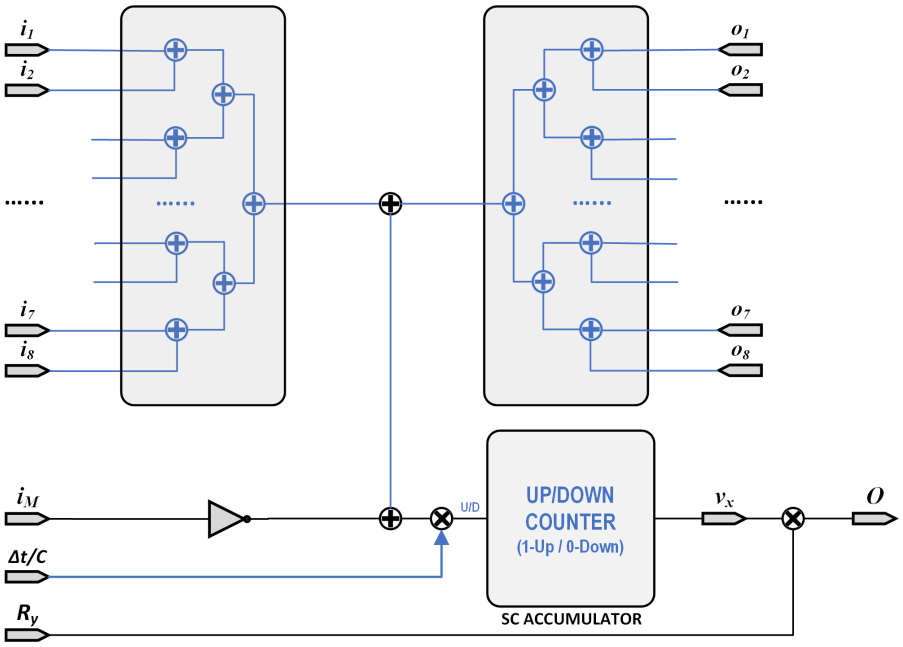


Figure 4.14: Stochastic Computing Circuit implementation of the M-CNN processing element C(i,j) as in Fig. 4.12.

$$\frac{dv_x}{dt} = \frac{1}{C}\left(\sum_{i,j}(B_{i,j}i_{i,j} + A_{i,j}o_{i,j}) - i_M\right) \tag{4.27}$$

where i_i are the input currents caused by the *inputs* of the nearest cells. The currents o_i correspond to the *outputs* of those cells, while the current i_M is that flowing through the memristor. As proposed in [141,142], we only consider the 8 closest neighbors. This way, Stochastic Computing can be used to make an implementation of this equation. As an initial step, we do a first order integration of equation (4.27):

$$\Delta v_x = \frac{\Delta t}{C}\left(\sum_{i}(B_i i_i + A_i o_i) - i_M\right) \tag{4.28}$$

The above Eq. (4.28) has been implemented in Fig. 4.14 as a stochastic equivalent circuit, where all data circulation correspond to 1-bit lines. Thus, implementation of Eq. (4.28) has been performed with 16 adders each implemented using 1 OR gate with a 1-bit *multiplexers*, 2 multipliers implemented as AND gates, 1 inverter, and 1 accumulator. Additionally, a random number generator is also needed (note that it may be shared between different cells) along with the memristor emulator previously presented.

The speed of the system can be estimated with the length of the chain

Table 4.3: Parameter values for the elements of the circuit in Fig. 4.12, where the memristor is defined by Eq. (4.7)

Parameter	Value
R	100 kΩ
C	50 μF
G_0	100 kΩ
ϕ_0	10 V·s

needed to implement the SCN representation of the numbers to be used. These numbers are determined considering the input images. In our case, we are using both gray images and color images, all of them downloaded from `http://www.hpca.ual.es/~vruiz/images`. The gray images use a 8-bit single plane to store it, while the color images use three different 8-bit planes. Thus, at least 8 bits need to be recovered faithfully. As has been discussed above [50], the use of 14 bits corresponds to a chain length of $2^{14} = 16384$ bits and can represent values with to an error confined in the last 6 bits, with more than a 95% probability. Following this reasoning, we choose these 14 bits for both the length of the chain and the accumulators.

4.3.3 Image Processing Results

The CNN described above was simulated using Matlab, with the circuit parameters equivalent to those appearing in Table 4.3. Parameter values have been chosen to be similar to those proposed in [33], to optimize the emulator behavior. Another option would have been using the process described in [141, 149] or [150], but then scaling of the values was differing significantly to allow the emulator performing efficiently. The FPGA-in-the-loop methodology, as shown in Fig. 4.10, was implemented to speed up the simulation, using an Arria V development kit. This FPGA system was connected to the computer running the Matlab code via a cabled network.

The A and B matrices were changed to three different sets, corresponding to three different cases: store, edge, sharpening, as discussed below. The notation to represent the coefficients in the matrices is shown in Table 4.4.

Table 4.4: Coefficient notation for $M = A, B$ ($m = a, b$). Notice that the coefficient for the current node is $(0, 0)$.

M	$\begin{pmatrix} m_{-1,-1} & m_{0,-1} & m_{1,-1} \\ m_{-1,0} & m_{0,0} & m_{1,0} \\ m_{-1,1} & m_{0,1} & m_{1,1} \end{pmatrix}$

The images have all different size (notated as NxM), as reported in Ta-

ble 4.5, and they were all initially color images with 8-bit color resolution at each RGB plane. We have processed these figures by performing two different experiments, both of them using three different (A,B) sets of parameters: first, we used a color-to-gray conversion, and we processed the images through the three different CNN. In a second step, we used a color image, and we processed each color plane independently to finally reconstruct a color image from these three planes.

In order to keep a good NF during the stochastic processing, and according to Eq. (1.9) and the discussion in the previous section, each pixel was converted from 8 to 14-bits by padding the least significant positions with zeros. After the processing, the stochastic images to normal images were converted back by disregarding the 6 least significant bits of the corresponding accumulator in Fig. 4.13.

The results are shown in two different ways: as pictures, and also using the RMS error and the entropy of the image. The RMS error e_{rms} is defined in Eq. (4.29), where $p_{i;i,j}$ and $p_{o;i,j}$ are the values of the pixel (i, j) for the input and the processed image, respectively. The entropy H is calculated using the Matlab implementation of Eq. (4.30), where p_i contains the normalized histogram counts for each gray level. The entropy of the unprocessed images is reported in Table 4.6. Notice that for the color images we report the values of the entropy for each channel, while for the rest we report only the entropy of the gray image.

$$e_{rms} = \frac{1}{N \cdot M} \sum_{i,j} (p_{o;i,j} - p_{p;i,j})^2 \tag{4.29}$$

$$H = -\sum_{i} p_i log_2(p_i) \tag{4.30}$$

4.3.3.1 Store image

As a first example, we show the results of storing the image into the CNN. That is, the values of internal voltage are evolved until the output becomes equal to the input. As a comment, this is the easiest "program" that can be implemented into the SM-CNN, and can be used as a first step for more

Table 4.5: Picture size in pixels. The color depth is 8 bits per channel.

Figure	Size (M x N) pixels^2
Fig. 4.15	512 x 512
Fig. 4.16	768 x 512
Fig. 4.18	768 x 512

Table 4.6: Calculated values of entropy of the images. The results for the color image show the three color planes separately.

Figure	Original	Store	Edge	Enhance
Fig. 4.15	6.70	6.71	4.03	6.90
Fig. 4.16	7.18	7.20	3.87	7.45
Fig. 4.18 (R)	7.15	7.07	4.04	7.38
Fig. 4.18 (G)	7.16	7.27	4.19	7.88
Fig. 4.18 (B)	7.16	7.28	4.26	7.76

Table 4.7: Coefficients for the input and output weights in Eq. (4.27) for the case of the image store setup.

A	$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.9 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$
B	$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.1 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$

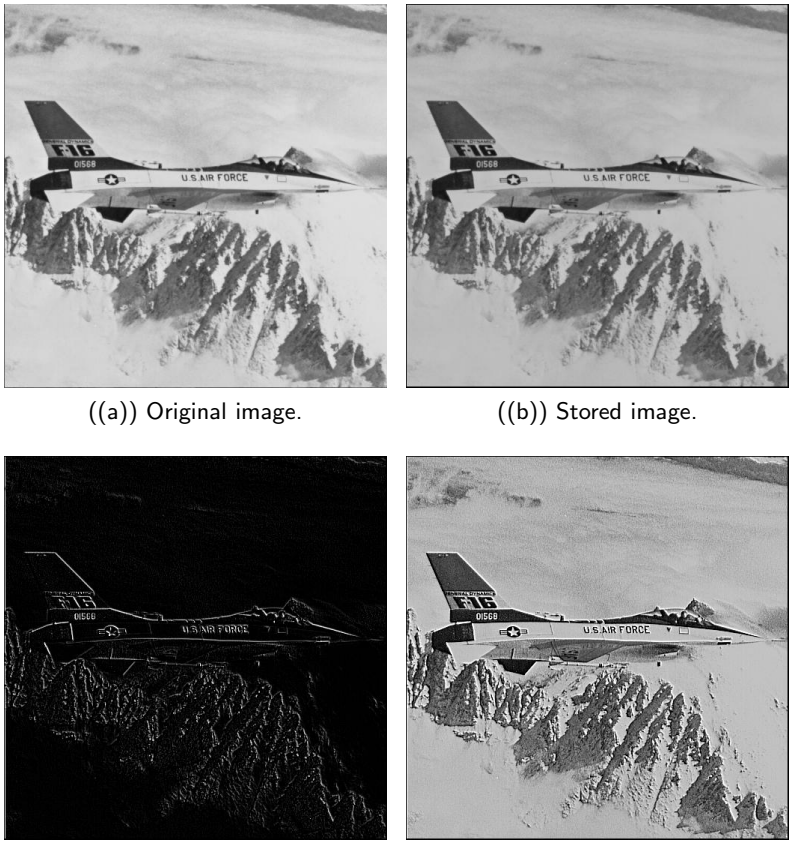
complex algorithms as can be, for instance, a background removal or a motion detection algorithms. The coefficients of the matrices are provided in Table 4.7.

We have represented both the input and output images for two different cases input gray images (Figures 4.15(a) and 4.16(a)) and a color image (Fig. 4.18(a)). The results for the store process are shown in Figures 4.15(b) and 4.16(b) for the gray images, while the stored color image is presented in Fig. 4.18(b). Visually, it can be seen there that the algorithm performs correctly.

Additionally, we have calculated the entropy of the images and the RMS error, as shown in Tables 4.6 and 4.8. The entropy of the images is nearly the same, and the rms error is kept, at most, below 1.3%.

Table 4.8: Calculated RMS of the stored images, referred to the original image. The results for the color image show the three color planes separately.

Figure	RMS
Fig. 4.15(b)	1.29%
Fig. 4.16(b)	0.57%
Fig. 4.18(b) (R)	1.13%
Fig. 4.18(b) (G)	0.58%
Fig. 4.18(b) (B)	0.61%



((a)) Original image. ((b)) Stored image.

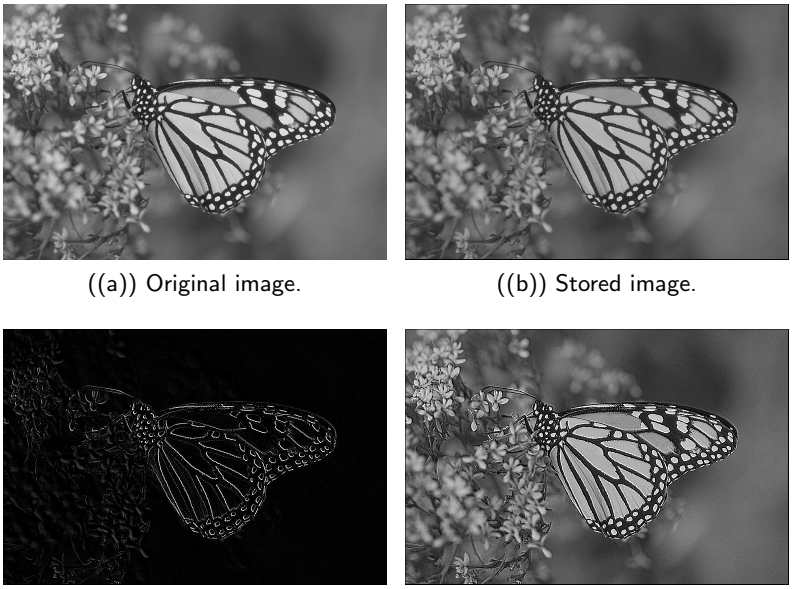
((c)) Edge detection result. ((d)) Sharpened image.

Figure 4.15: Example 1: results obtained using the three proposed stochastic computing CNN with different gene values.

4.3.3.2 EDGE detection using SM-CNN

The proposed SM-CNN was further checked using an implementation of one of the stochastic systems proposed in [142]. Specifically, we have improved the EDGE routine presented in [38]. This routine performs a border detection algorithm in the image using the coefficients in Table 4.9. In the previous work, the routine was fixed, and no quantitative analysis was performed. The edge algorithm aims to detect changes between adjacent pixels, so the output value will evolve to a 1 or 0, depending on the change of color. The evolution will depend on the threshold of the output function and can thus be changed.

We have used the same images as in the previous example, where two of the images were gray 8-bit images and another one was a 24-bit color image (3x8 bits planes). They were processed as in the previous case to 14-bits by padding, and back to 8-bits by truncation. We have represented both the



((a)) Original image. ((b)) Stored image.

((c)) Edge detection result. ((d)) Sharpened image.

Figure 4.16: Example 2: results obtained using the three proposed stochastic computing CNN with different gene values.

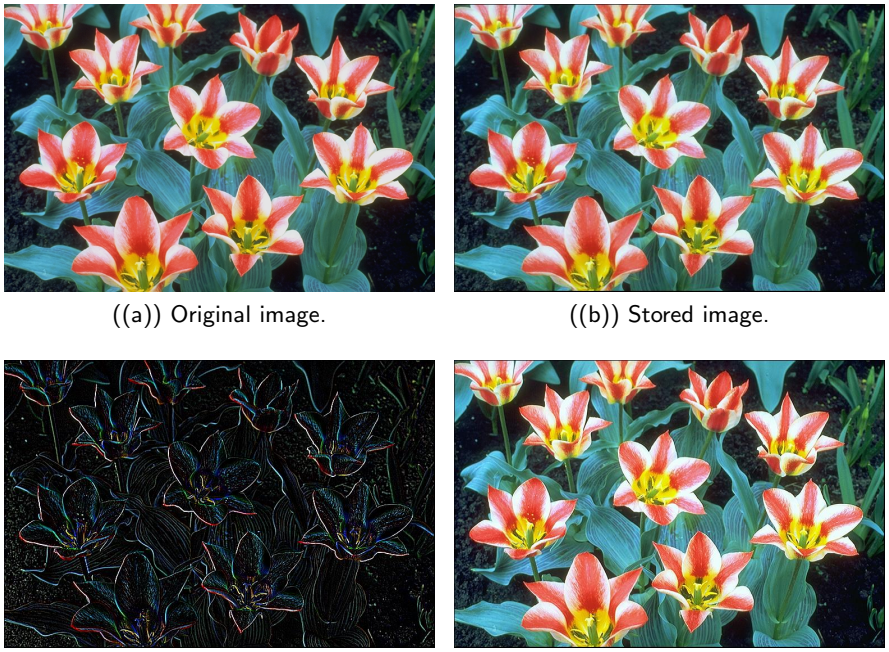
((a)) Zoom of the original image. ((b)) Zoom of the sharpened image.

Figure 4.17: Example 2: zoom in of Fig 4.16(a) and 4.16(d), showing a detail of the sharpening results obtained using the proposed stochastic computing CNN.

input and output images for two different gray images in Fig. 4.15(c) and Fig. 4.16(c), while the result for the color image is shown in Fig. 4.18(c). It can be seen there that the algorithm performs as expected, showing also the corresponding decreasing in the values of the entropy in Table 4.6.

4.3.3.3 Sharpening

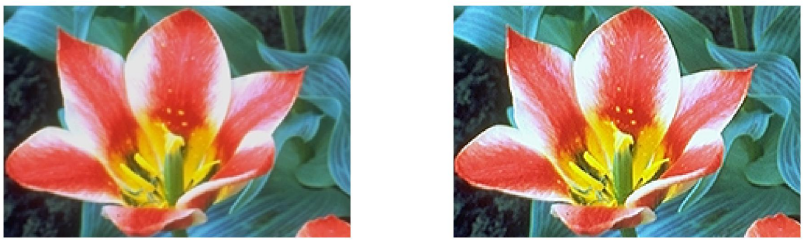
The sharpening algorithm is a variation of the EDGE detection, combined with the STORE genii. In fact, we have calculated a new coefficient set as



((a)) Original image. ((b)) Stored image.

((c)) Edge detection result. ((d)) Sharpened image.

Figure 4.18: Example 3: Color figure, showing the sharpening results obtained using the proposed stochastic computing CNN.



((a)) Zoom of the original image. ((b)) Zoom of the sharpened image.

Figure 4.19: Example 3: zoom in of Fig 4.18(a) and 4.18(d), showing a detail of the sharpening results obtained using the proposed stochastic computing CNN.

Table 4.9: Coefficients for the input and output weights in Eq. (4.27) for the case of the edge detection setup.

A	$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$
B	$\begin{pmatrix} -1/8 & -1/8 & -1/8 \\ -1/8 & 1.0 & -1/8 \\ -1/8 & -1/8 & -1/8 \end{pmatrix}$

a linear combination of the two previous sets: A_s, B_s for the store matrices, and A_e, B_e for the edge detection. The new set A_i, B_i is calculated as:

$$M_i = \lambda_1 M_s + \lambda_2 M_i \tag{4.31}$$

where M stands for both A and B. In this case, $\lambda_1 = 1/3$ and $\lambda_2 = 2/5$, and the corresponding matrix coefficients are provided in Table 4.10.

Table 4.10: Coefficients for the input and output weights in Eq. (4.27) for the case of the image enhancement setup.

A	$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.3 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$
B	$\begin{pmatrix} -0.05 & -0.05 & -0.05 \\ -0.05 & 0.43 & -0.05 \\ -0.05 & -0.05 & -0.05 \end{pmatrix}$

We have used the same images as in the previous example, where two of the images were gray 8-bit images and another one was a 24-bit color image (3x8 bits planes). They were processed as in the previous case to 14-bits by padding, and back to 8-bits by truncation. Results of applying this set of coefficients to gray images are shown in Fig. 4.15(d), 4.16(d) (with a zoom comparing original in Fig. 4.17(a) against the processed output in 4.17(b)). The results for a color image are depicted in Fig. 4.18(d), with a zoom in shown in Fig. 4.19(a) and 4.19(b) for the original and processed images, respectively. Visually, the images seem to be improved, with less fuzzy edges. This is further corroborated by the increase in the entropy

shown in Table 4.6.

4.3.4 Concluding remarks

A fully digital implementation of a Memristive Cellular Nonlinear Network profiting from the Stochastic Computing paradigm, has been performed. The basic unitary cell of the proposed CNN features a digital memristor emulator, plus several arithmetic units that are implemented as very simple gates, allowing for an enormous number of cells in parallel, which can translate into a very fast image processor.

We have implemented this full CNN structure into an ARRIA V FPGA, and we have tested it along with Matlab by implementing three different procedures: a image store, an edge detection, and, finally, an image sharpening process. Notice that these three procedures involve only the change of the matrix coefficients, that are common to all the cells. As has been discussed, the results imply that the system performs smoothly, with errors lower than 1.3% in the storage, an excellent edge detection capability, and a very good detail sharpening. A full FPGA implementation of images with lower number of pixels would allow for a very high image processing speed, adequate for real time needs, and well inside the capabilities and requirements of edge computing. In addition, as shown in [150], the proposed kind of memristive CNNs are resilient to individual "pixel" failures.

As an example, in this thesis we have used 14-bits stochastic numbers, which translates into a chain length of $2^{14} = 16384$. An entry-level FPGA can run at 80MHz, so it could operate around 4800 operations per second. Since all the operations in the circuit are sequential, this is also the speed at which each step of the numerical integration is performed. Assuming that you need around 10-20 steps to reach the stable point, we could be processing more than 200 points per second. Then, the full resolution would be a matter of how many parallel threads can be implemented into an FPGA or an ASIC, but the numbers show that it seems to be adequate for real-time processing, even using low-speed systems.

Notice that in this chapter only the proof of concept for the SC Memristive CNN has been discussed, not comparing it against any other improving algorithm using, for instance, a hard-wired algorithm implementation or Neural Networks, which may show much better image improvement. It has to be noted, however, that the method presented here is training-free, which simplifies the design when compared against NNs and also removes any possible bias introduced by the training. In addition, the facility to change the algorithm is also worth to mention, since it reduces to changing the values of the coefficients in the the cells. This makes this approach specially suited over a hard implementation of specific algorithms in, for instance, multi-purpose systems that can need to be swapping functions on the fly.

Chapter 5

Conclusion

In this thesis we have studied different applications of Stochastic Computing (SC) to nonlinear systems. Specifically, we have first studied the error propagation in this kind of architecture, seeing how it can be controlled.

As a second step, we have presented a new scheme to implement arbitrary functions using a quadratic approach, instead of the linear approach normally used. Our scheme clearly improves the use of memory and computational resources, with a very small overhead, while also increasing the obtained precision.

The third step involved studying the implementation of nonlinear differential equation systems. In order to proceed, we have first studied how to implement an integrator, and we have seen how our proposal integrates a constant and also how it can be used in a feedback loop to provide an oscillator by implementing the classical equation $\ddot{x} = -x$. Afterwards, we have implemented a Shimizu-Morioka system, which uses only three dynamical variables. Results show that the implementation using SC may need massive parallelization to be effective, but it reproduces fairly well the results obtained by integrating the system using Matlab.

Another part of the thesis is devoted to the implementation of memristor emulators. As is known, memristors are nonlinear systems, and there is also a great deal of effort involved in creating emulators. We have first proposed a simple digital emulator, showing that it reproduces the expected results when using, for instance, imply gates. Afterwards, we have also implemented a physical emulator using physical elements and switched capacitors. We have presented two different emulators: the first one [32] was a proof of concept using a only switched capacitors controlled by a linear relation between duty cycle and flux, while the second emulator [34] implemented all the operations (including switch control) using stochastic computing.

Finally, we have used the digital emulator to implement some more applications, all of them also published. First, we have implemented a maze

solver [36] into a FPGA. This maze solver implements a version of the Dijkstra algorithm, and it works well with grids up to 300 x 300 elements. As a second application, we have implemented a Cellular Nonlinear Network (CNN) [30], with application to image processing. The results obtained using this emulator show that they work very well, with less energy consumption that comparable implementations using other techniques, if we keep below 16 bits, while providing a great capability for changing on the fly the image processing algorithm.

In brief, we have shown how Stochastic Computing can occupy some niches, mainly for edge computing, or for low-precission, low-power systems. Extensions of the present work should include ASIC implementations of the proposed blocks, maybe with applications in robotic navigation or as a first layer in image processing. Further improvements could also include using memristors to implement all the operations with in-memory computing, since these elements seem to have intrinsic randomness, which could be harnessed using SC.

List of Figures

List of Tables

Bibliography

[1] W. Shi, G. Pallis, and Z. Xu, "Edge computing [scanning the issue]," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1474–1481, 2019.

[2] S. Venkataramani, K. Roy, and A. Raghunathan, "Efficient embedded learning for IoT devices," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 308–311.

[3] M. Shafique, T. Theocharides, C.-S. Bouganis, M. A. Hanif, F. Khalid, R. Hafız, and S. Rehman, "An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IOT era," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 827–832.

[4] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2015.

[5] H. Jayakumar, A. Raha, Y. Kim, S. Sutar, W. S. Lee, and V. Raghunathan, "Energy-efficient system design for IoT devices," in *21st Asia and South Pacific Design Automation Conf (ASP-DAC)*. IEEE, 2016, pp. 298–301.

[6] M. Gao, Q. Wang, M. T. Arafin, Y. Lyu, and G. Qu, "Approximate computing for low power and security in the internet of things," *Computer*, vol. 50, no. 6, pp. 27–34, 2017.

[7] L. Du, Y. Du, Y. Li, J. Su, Y.-C. Kuan, C.-C. Liu, and M.-C. F. Chang, "A reconfigurable streaming deep convolutional neural network accelerator for internet of things," *IEEE Trans on Circuits and Systems I: Reg. Papers*, vol. 65, no. 1, pp. 198–208, 2017.

[8] E. Ipek, "Memristive accelerators for dense and sparse linear algebra: From machine learning to high-performance scientific computing," *IEEE Micro*, vol. 39, no. 1, pp. 58–61, 2019.

[9] F. Yu, Z. Zhang, L. Liu, H. Shen, Y. Huang, C. Shi, S. Cai, Y. Song, S. Du, and Q. Xu, "Secure communication scheme based on a new

5d multistable four-wing memristive hyperchaotic system with disturbance inputs," *Complexity*, vol. 2020, 2020.

[10] A. Dukhan, D. Jayalath, P. van Heijster, B. Senadji, and J. Banks, "A generalized multilevel-hybrid chaotic oscillator for low-cost and power-efficient short-range chaotic communication systems," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, no. 1, p. 23, 2020.

[11] F.-Y. Rao and E. Bertino, "Privacy techniques for edge computing systems," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1632–1654, 2019.

[12] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge computing security: State of the art and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1608–1631, 2019.

[13] G. Alvarez and S. Li, "Some basic cryptographic requirements for chaos-based cryptosystems," *International journal of bifurcation and chaos*, vol. 16, no. 08, pp. 2129–2151, 2006.

[14] A. N. Miliou, I. P. Antoniades, S. G. Stavrinides, and A. N. Anagnostopoulos, "Secure communication by chaotic synchronization: Robustness under noisy conditions," *Nonlinear analysis: real world applications*, vol. 8, no. 3, pp. 1003–1012, 2007.

[15] A. Anagnostopoulos, A. Miliou, S. Stavrinides, A. Dmitriev, and E. Efremova, "Digital information transmission using discrete chaotic signal," in *Chaos Synchronization and Cryptography for Secure Communications: Applications for Encryption.* IGI Global, 2011, pp. 439–462.

[16] S. Stavrinides, A. Anagnostopoulos, A. Miliou, A. Valaristos, L. Magafas, K. Kosmatopoulos, and S. Papaioannou, "Digital chaotic synchronized communication system," *Journal of Engineering Science and Technology Review*, vol. 2, no. 1, pp. 82–86, 2009.

[17] S. Stavrinides, N. Karagiorgos, K. Papathanasiou, S. Nikolaidis, and A. Anagnostopoulos, "A digital nonautonomous chaotic oscillator suitable for information transmission," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 887–891, 2013.

[18] A. Miliou, A. Valaristos, S. Stavrinides, K. Kyritsi, and A. Anagnostopoulos, "Characterization of a non-autonomous second-order nonlinear circuit for secure data transmission," *Chaos, Solitons & Fractals*, vol. 33, no. 4, pp. 1248–1255, 2007.

[19] A. Miliou, S. Stavrinides, A. Valaristos, and A. Anagnostopoulos, "Nonlinear electronic circuit, part ii: synchronization in a chaotic modem scheme," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 71, no. 12, pp. e21–e31, 2009.

[20] J. C. Sprott, *Chaos and Time-Series Analysis*. USA: Oxford University Press, Inc., 2003.

[21] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[22] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2688–2699, 2017.

[23] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[24] A. Morro, V. Canals, A. Oliver, M. L. Alomar, and J. L. Rossello, "Ultra-fast data-mining hardware architecture based on stochastic computing," *PloS one*, vol. 10, no. 5, p. e0124176, 2015.

[25] R. Wang, J. Han, B. Cockburn, and D. Elliott, "Stochastic circuit design and performance evaluation of vector quantization," in *Application-specific Systems, Architectures and Processors (ASAP) IEEE 26th Int. Conf. on*. IEEE, 2015, pp. 111–115.

[26] B. Yuan, Y. Wang, and Z. Wang, "Area-efficient scaling-free DFT/FFT design using stochastic computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 12, pp. 1131–1135, 2016.

[27] S. T. Marin, J. Q. Reboul, and L. G. Franquelo, "Digital stochastic realization of complex analog controllers," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 5, pp. 1101–1109, 2002.

[28] S. Toral, J. Quero, J. Ortega, and L. Franquelo, "Stochastic A/D sigma-delta converter on FPGA," in *Circuits and Systems, 1999. 42nd Midwest Symposium on*, vol. 1. IEEE, 1999, pp. 35–38.

[29] H. Schuster and W. Just, *Deterministic Chaos: An Introduction*. Wiley, 2006. [Online]. Available: https://books.google.de/books?id=-14Y2WPfYgsC

[30] O. Camps, S. G. Stavrinides, and R. Picos, "Stochastic computing implementation of chaotic systems," *Mathematics*, vol. 9, no. 4, 2021. [Online]. Available: https://www.mdpi.com/2227-7390/9/4/375

[31] O. Camps, M. M. Al Chawa, C. de Benito, M. Roca, S. G. Stavrinides, R. Picos, and L. O. Chua, "A purely digital memristor emulator based on a flux-charge model," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2018, pp. 565–568.

[32] G. Svetoslavov, O. Camps, S. G. Stavrinides, and R. Picos, "A switched capacitor memristive emulator," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.

[33] O. Camps, R. Picos, C. de Benito, M. M. Al Chawa, and S. G. Stavrinides, "Emulating memristors in a digital environment using stochastic logic," in *2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2018, pp. 1–4.

[34] C. de Benito, O. Camps, M. Al Chawa, S. Stavrinides, and R. Picos, "A stochastic switched capacitor memristor emulator," in *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2021, pp. 1–4.

[35] C. de Benito, O. Camps, M. M. Al Chawa, S. G. Stavrinides, and R. Picos, "A switched capacitor memristor emulator using stochastic computing," *Technologies*, vol. 10, no. 2, p. 39, 2022.

[36] P. Dopazo, C. de Benito, O. Camps, S. G. Stavrinides, and R. Picos, "Gerard: General rapid resolution of digital mazes using a memristor emulator," *Physics*, vol. 4, no. 1, pp. 1–11, 2021.

[37] O. Camps, M. M. Al Chawa, S. G. Stavrinides, and R. Picos, "Stochastic computing emulation of memristor cellular nonlinear networks," *Micromachines*, vol. 13, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2072-666X/13/1/67

[38] O. Camps, S. G. Stavrinides, and R. Picos, "Efficient implementation of memristor cellular nonlinear networks using stochastic computing," in *2020 European Conference on Circuit Theory and Design (ECCTD)*. IEEE, 2020, pp. 1–4.

[39] H. Hui, C. Zhou, S. Xu, and F. Lin, "A novel secure data transmission scheme in industrial internet of things," *China Communications*, vol. 17, no. 1, pp. 73–88, 2020.

[40] A. Voronova, P. Tsareva, and A. Zhilenkov, "The synthesis problem of a chaotic signal computer system for secure data transmission," in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, 2020, pp. 551–555.

[41] H. Peng, Y. Tian, J. Kurths, L. Li, Y. Yang, and D. Wang, "Secure and energy-efficient data transmission system based on chaotic compressive sensing in body-to-body networks," *IEEE transactions on biomedical circuits and systems*, vol. 11, no. 3, pp. 558–573, 2017.

[42] M. R. Alam, M. H. Najafi, N. T. Nejad, M. Imani, and R. Gottumukkala, "Stochastic computing in beyond von-neumann era: Processing bit-streams in memristive memory," *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–1, 2022.

[43] B. Moons and M. Verhelst, "Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 4, pp. 475–486, 2014.

[44] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "SCOPE: A stochastic computing engine for dram-based in-situ accelerator." in *MICRO*, 2018, pp. 696–709.

[45] S. Toral, J. Quero, and L. Franquelo, "Stochastic pulse coded arithmetic," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 1. IEEE, 2000, pp. 599–602.

[46] F. A. Khanday and R. Akhtar, "Reversible stochastic computing," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. n/a, no. n/a, p. e2711, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jnm.2711

[47] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 149–156.

[48] S. Mitra, D. Banerjee, and M. K. Naskar, "A low latency stochastic square root circuit," in *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*, 2021, pp. 7–12.

[49] D. Wu and J. S. Miguel, "In-stream stochastic division and square root via correlation," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[50] O. Camps, R. Picos, C. de Benito, M. M. Al Chawa, and S. G. Stavrinides, "Effective accuracy estimation and representation error reduction for stochastic logic operations," in *2018 7th Int. Conf. on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2018.

[51] T.-H. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 116–121.

[52] Z. Qin, Y. Qiu, M. Zheng, H. Dong, Z. Lu, Z. Wang, and H. Pan, "A universal approximation method and optimized hardware architectures for arithmetic functions based on stochastic computing," *IEEE Access*, vol. 8, pp. 46 229–46 241, 2020.

[53] J. Tellinghuisen, "Statistical error propagation," *The Journal of Physical Chemistry A*, vol. 105, no. 15, pp. 3917–3921, 2001.

[54] L. A. Goodman, "On the exact variance of products," *Journal of the American Statistical Association*, vol. 55, no. 292, pp. 708–713, 1960. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1960.10483369

[55] M. Loeve, "Elementary probability theory," in *Probability theory i*. Springer, 1977, pp. 1–52.

[56] S. Liu, W. J. Gross, and J. Han, "Introduction to dynamic stochastic computing," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 19–33, 2020.

[57] T. Shimizu and N. Morioka, "On the bifurcation of a symmetric limit cycle to an asymmetric one in a simple model," *Physics Letters A*, vol. 76, no. 3-4, pp. 201–204, 1980.

[58] F. Neugebauer, I. Polian, and J. P. Hayes, "S-box-based random number generation for stochastic computing," *Microprocessors and Microsystems*, vol. 61, pp. 316–326, 2018.

[59] V. K. Rai, S. Tripathy, and J. Mathew, "Memristor based random number generator: Architectures and evaluation," *Procedia Computer Science*, vol. 125, pp. 576–583, 2018.

[60] B.-B. Yang, N. Xu, E.-R. Zhou, Z.-W. Li, C. Li, P.-Y. Yi, and L. Fang, "A method of generating random bits by using electronic bipolar memristor," *Chinese Physics B*, vol. 29, no. 4, p. 048505, 2020.

[61] M. Téllez, J. Mejía, H. López, and C. Hernández, "Random number generator with long-range dependence and multifractal behavior based on memristor," *Electronics*, vol. 9, no. 10, p. 1607, 2020.

[62] R. Picos, M. Roca, B. Iniguez, and E. Garcia-Moreno, "A new procedure to extract the threshold voltage of mosfets using noise-reduction techniques," *Solid-State Electronics*, vol. 47, no. 11, pp. 1953–1958, 2003.

[63] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical systems and turbulence, Warwick 1980*. Springer, 1981, pp. 366–381.

[64] P. Grassberger and I. Procaccia, "Characterization of strange attractors," *Physical review letters*, vol. 50, no. 5, p. 346, 1983.

[65] ——, "Dimensions and entropies of strange attractors from a fluctuating dynamics approach," *Physica D: Nonlinear Phenomena*, vol. 13, no. 1-2, pp. 34–54, 1984.

[66] ——, "Measuring the strangeness of strange attractors," in *The Theory of Chaotic Attractors*. Springer, 2004, pp. 170–189.

[67] P. Bryant, R. Brown, and H. D. Abarbanel, "Lyapunov exponents from observed time series," *Physical Review Letters*, vol. 65, no. 13, p. 1523, 1990.

[68] H. D. Abarbanel, R. Brown, J. J. Sidorowich, and L. S. Tsimring, "The analysis of observed chaotic data in physical systems," *Reviews of modern physics*, vol. 65, no. 4, p. 1331, 1993.

[69] M. Mathur *et al.*, "Demystification of vedic multiplication algorithm," *American Journal of Computational Mathematics*, vol. 7, no. 01, p. 94, 2017.

[70] M. M. Kamble and S. P. Ugale, "FPGA implementation and analysis of different multiplication algorithms," *Int J Comput Appl*, vol. 149, no. 2, p. 8887, 2016.

[71] P. S. Georgiou, S. N. Yaliraki, E. M. Drakakis, and M. Barahona, "Window functions and sigmoidal behaviour of memristive systems," *International Journal of Circuit Theory and Applications*, 2016.

[72] A. Ascoli, R. Tetzlaff, and L. Chua, "Continuous and differentiable approximation of a tao memristor model for robust numerical simulations," in *Emergent Complexity from Nonlinearity, in Physics, Engineering and the Life Sciences*. Springer, 2017, pp. 61–69.

[73] F. Jimenez-Molinos, M. Villena, J. Roldan, and A. Roldan, "A spice compact model for unipolar rram reset process analysis," *IEEE Transactions on Electron Devices*, 2015.

[74] R. Picos, J. B. Roldan, M. M. Al Chawa, P. Garcia-Fernandez, F. Jimenez-Molinos, and E. Garcia-Moreno, "Semiempirical modeling of reset transitions in unipolar resistive-switching based memristors," *RADIOENGINEERING*, vol. 24, no. 2, p. 421, 2015.

[75] J. Secco, F. Corinto, and A. Sebastian, "Flux–charge memristor model for phase change memory," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 1, pp. 111–114, 2017.

[76] M. M. Al Chawa, R. Picos, J. B. Roldan, F. Jimenez-Molinos, M. A. Villena, and C. de Benito, "Exploring resistive switching-based memristors in the charge–flux domain: A modeling approach," *International Journal of Circuit Theory and Applications*, vol. 46, no. 1, pp. 29–38, 2018.

[77] Z. Kolka, D. Biolek, V. Biolkova, and Z. Biolek, "Evaluation of memristor models for large crossbar structures," in *Radioelektronika (RADIOELEKTRONIKA), 2016 26th International Conference*. IEEE, 2016, pp. 91–94.

[78] S. Stavrinides, R. Picos, F. Corinto, M. Al Chawa, and C. de Benito, *Mem-elements for Neuromorphic Circuits with Artificial Intelligence Applications*. Elsevier, June 2021, ch. Implementing memristor emulators in hardware.

[79] A. Ascoli, F. Corinto, and R. Tetzlaff, "A class of versatile circuits, made up of standard electrical components, are memristors," *International Journal of Circuit Theory and Applications*, vol. 44, no. 1, pp. 127–146, 2016.

[80] A. Ascoli, R. Tetzlaff, L. Chua, W. Yi, and R. Williams, "Memristor emulators: A note on modeling," in *Advances in Memristors, Memristive Devices and Systems*. Springer, 2017, pp. 1–17.

[81] J. Kalomiros, S. G. Stavrinides, and F. Corinto, "A two-transistor non-ideal memristor emulator," in *Modern Circuits and Systems Technologies (MOCAST), 2016 5th International Conference on*. IEEE, 2016, pp. 1–4.

[82] C. Sánchez-López, J. Mendoza-Lopez, M. Carrasco-Aguilar, and C. Muñiz-Montero, "A floating analog memristor emulator circuit," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 5, pp. 309–313, 2014.

[83] M. T. Abuelma'atti and Z. J. Khalifa, "A continuous-level memristor emulator and its application in a multivibrator circuit," *AEU-International Journal of Electronics and Communications*, vol. 69, no. 4, pp. 771–775, 2015.

[84] M. Al Chawa, C. de Benito, M. Roca, R. Picos, and S. Stavrinides, "Design and implementation of passive memristor emulators using a charge-flux approach," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[85] I. Vourkas, A. Abusleme, V. Ntinas, G. C. Sirakoulis, and A. Rubio, "A digital memristor emulator for FPGA-based artificial neural networks," in *Verification and Security Workshop (IVSW), IEEE International*. IEEE, 2016, pp. 1–4.

[86] R. Ranjan, P. M. Ponce, A. Kankuppe, B. John, L. A. Saleh, D. Schroeder, and W. H. Krautschneider, "Programmable memristor emulator ASIC for biologically inspired memristive learning," in *Telecommunications and Signal Processing (TSP), 2016 39th International Conference on*. IEEE, 2016, pp. 261–264.

[87] Z. Kolka, J. Vavra, V. Biolkova, A. Ascoli, R. Tetzlaff, and D. Biolek, "Programmable emulator of genuinely floating memristive switching devices," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2019, pp. 217–220.

[88] H. Kim, M. P. Sah, C. Yang, S. Cho, and L. O. Chua, "Memristor emulator for memristor circuit applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 10, pp. 2422–2431, 2012.

[89] Z. Li, Y. Zeng, and M. Ma, "A novel floating memristor emulator with minimal components," *Active and Passive Electronic Components*, vol. 2017, 2017.

[90] F. J. Romero, A. Ohata, A. Toral-Lopez, A. Godoy, D. P. Morales, and N. Rodriguez, "Memcapacitor and meminductor circuit emulators: A review," *Electronics*, vol. 10, no. 11, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/11/1225

[91] L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209–223, 1976.

[92] L. O. Chua, "Everything you wish to know about memristors but are afraid to ask," *Radioengineering*, vol. 24, no. 2, p. 319, 2015.

[93] F. Corinto, P. P. Civalleri, and L. O. Chua, "A theoretical approach to memristor devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 2, pp. 123–132, 2015.

[94] D. Biolek, Z. Biolek, V. Biolková, and Z. Kolka, "Some fingerprints of ideal memristors," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 201–204.

[95] L. Chua, "If it's pinched it'sa memristor," *Semiconductor Science and Technology*, vol. 29, no. 10, p. 104001, 2014.

[96] D. Ielmini and V. Milo, "Physics-based modeling approaches of resistive switching devices for memory and in-memory computing applications," *Journal of Computational Electronics*, vol. 16, no. 4, pp. 1121–1143, 2017.

[97] K.-C. Chang, T.-C. Chang, T.-M. Tsai, R. Zhang, Y.-C. Hung, Y.-E. Syu, Y.-F. Chang, M.-C. Chen, T.-J. Chu, H.-L. Chen *et al.*, "Physical and chemical mechanisms in oxide-based resistance random access memory," *Nanoscale research letters*, vol. 10, no. 1, p. 120, 2015.

[98] R. S. Williams, M. D. Pickett, and J. P. Strachan, "Physics-based memristor models," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 217–220.

[99] C. de Benito, M. M. Al Chawa, R. Picos, and E. Garcia-Moreno, "A procedure to calculate a delay model for memristive switches," in *Workshop on Memristor Technology, Design, Automation and Computing*, 2017.

[100] I. Vourkas and G. C. Sirakoulis, "Emerging memristor-based logic circuit design approaches: A review," *IEEE Circuits and Systems Magazine*, vol. 16, no. 3, pp. 15–30, 2016.

[101] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (imply) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2014.

[102] M. Uno and A. Kukita, "Pwm switched capacitor converter with switched-capacitor-inductor cell for adjustable high step-down voltage conversion," *IEEE Trans. on Power Electronics*, vol. 34, no. 1, pp. 425–437, 2019.

[103] J. W. Kimball, P. T. Krein, and K. R. Cahill, "Modeling of capacitor impedance in switching converters," *IEEE Power Electronics Letters*, vol. 3, no. 4, pp. 136–140, 2005.

[104] A. Forencich, "Verilog implementation of mersenne twister prng," https://github.com/alexforencich/verilog-mersenne, 2018.

[105] S. Mishra and P. Bande, "Maze solving algorithms for micro mouse," in *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*, 2008, pp. 86–93.

[106] M. Fattah, A. Airola, R. Ausavarungnirun, N. Mirzaei, P. Liljeberg, J. Plosila, S. Mohammadi, T. Pahikkala, O. Mutlu, and H. Tenhunen, "A low-overhead, fully-distributed, guaranteed-delivery routing algorithm for faulty network-on-chips," in *Proceedings of the 9th International Symposium on Networks-on-Chip*, 2015, pp. 1–8.

[107] O. Kathe, V. Turkar, A. Jagtap, and G. Gidaye, "Maze solving robot using image processing," in *2015 IEEE Bombay Section Symposium (IBSS)*. IEEE, 2015, pp. 1–5.

[108] Suriyanath, "Dijkstra algorithm," 2014. [Online]. Available: https://es.mathworks.com/matlabcentral/fileexchange/46883-dijkstra-algorithm

[109] F. Caruso, A. Crespi, A. G. Ciriolo, F. Sciarrino, and R. Osellame, "Fast escape of a quantum walker from an integrated photonic maze," *Nature communications*, vol. 7, no. 1, pp. 1–7, 2016.

[110] I. Vourkas, D. Stathis, and G. C. Sirakoulis, "Massively parallel analog computing: Ariadne's thread was made of memristors," *IEEE Trans. on Emerging Topics in Computing*, vol. 6, no. 1, pp. 145–155, 2015.

[111] G. Papandroulidakis, I. Vourkas, G. C. Sirakoulis, S. G. Stavrinides, and S. Nikolaidis, "Multi-state memristive nanocrossbar for high-radix computer arithmetic systems," in *2015 IEEE 15th International Conference on Nanotechnology (IEEE-NANO)*. IEEE, 2015, pp. 625–628.

[112] Y. V. Pershin and M. Di Ventra, "Solving mazes with memristors: A massively parallel approach," *Physical Review E*, vol. 84, no. 4, p. 046703, 2011.

[113] L. O. Chua, "Memristor-the missing circuit element," *Circuit Theory, IEEE Transactions on*, vol. 18, no. 5, pp. 507–519, 1971.

[114] ——, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, no. 4, pp. 765–783, 2011.

[115] D. Biolek, Z. Kolka, V. Biolková, Z. Biolek, M. Potrebić, and D. Tošić, "Modeling and simulation of large memristive networks," *International Journal of Circuit Theory and Applications*, vol. 46, no. 1, pp. 50–65, 2018.

[116] D. Batas and H. Fiedler, "A memristor spice implementation and a new approach for magnetic flux-controlled memristor modeling," *IEEE Transactions on Nanotechnology*, vol. 10, no. 2, pp. 250–255, 2010.

[117] Z. Biolek, D. Biolek, and V. Biolkova, "Spice model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, no. 2, pp. 210–214, 2009.

[118] V. Mladenov, "A unified and open ltspice memristor model library," *Electronics*, vol. 10, no. 13, p. 1594, 2021.

[119] E. Garcia-Moreno, R. Picos, and M. M. Al-Chawa, "Spice model for unipolar rram based on a flux-controlled memristor," in *Power, Electronics and Computing (ROPEC), 2015 IEEE International Autumn Meeting on.* IEEE, 2015, pp. 1–4.

[120] F. García-Redondo, R. P. Gowers, A. Crespo-Yepes, M. López-Vallejo, and L. Jiang, "Spice compact modeling of bipolar/unipolar memristor switching governed by electrical thresholds," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 8, pp. 1255–1264, 2016.

[121] V. Saxena, "A compact cmos memristor emulator circuit and its applications," in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS).* IEEE, 2018, pp. 190–193.

[122] D.-S. Yu, T.-T. Sun, C.-Y. Zheng, H. Iu, and T. Fernando, "A simpler memristor emulator based on varactor diode," *Chinese Physics Letters*, vol. 35, no. 5, p. 058401, 2018.

[123] Y. V. Pershin and M. Di Ventra, "Emulation of floating memcapacitors and meminductors using current conveyors," *Electronics Letters*, vol. 47, no. 4, pp. 243–244, 2011.

[124] N. Wang, G. Zhang, and H. Bao, "Bursting oscillations and coexisting attractors in a simple memristor-capacitor-based chaotic circuit," *Nonlinear Dynamics*, vol. 97, no. 2, pp. 1477–1494, 2019.

[125] R. Picos, J. Roldan, M. Al Chawa, F. Jimenez-Molinos, and E. Garcia-Moreno, "A physically based circuit model to account for variability in memristors with resistive switching operation," in *Design of Circuits and Integrated Systems (DCIS), 2016 Conference on.* IEEE, 2016, pp. 1–6.

[126] R. Naous, M. Al-Shedivat, and K. N. Salama, "Stochasticity modeling in memristors," *IEEE Transactions on Nanotechnology*, vol. 15, no. 1, pp. 15–28, 2015.

[127] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[128] D. Biolek, Z. Biolek, and V. Biolkova, "Pspice modeling of meminductor," *Analog Integrated Circuits and Signal Processing*, vol. 66, no. 1, pp. 129–137, 2011.

[129] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, "A versatile memristor model with nonlinear dopant kinetics," *IEEE transactions on electron devices*, vol. 58, no. 9, pp. 3099–3105, 2011.

[130] V. Mladenov and S. Kirilov, "A memristor model with a modified window function and activation thresholds," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[131] S. Kvatinsky, K. Talisveyberg, D. Fliter, A. Kolodny, U. C. Weiser, and E. G. Friedman, "Models of memristors for spice simulations," in *2012 IEEE 27th Convention of electrical and electronics engineers in Israel*. IEEE, 2012, pp. 1–5.

[132] M. M. Al Chawa, R. Picos, and R. Tetzlaff, "A compact memristor model for neuromorphic reram devices in flux-charge space," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 9, pp. 3631–3641, 2021.

[133] M. M. Al Chawa, R. Tetzlaff, and R. Picos, "A flux-controlled memristor model for neuromorphic reram devices," in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2020, pp. 1–4.

[134] D. Panda, P. P. Sahu, and T. Y. Tseng, "A collective study on modeling and simulation of resistive random access memory," *Nanoscale research letters*, vol. 13, no. 1, pp. 1–48, 2018.

[135] J. B. Roldán, G. González-Cordero, R. Picos, E. Miranda, F. Palumbo, F. Jiménez-Molinos, E. Moreno, D. Maldonado, S. B. Baldomá, M. Moner Al Chawa *et al.*, "On the thermal models for resistive random access memory circuit simulation," *Nanomaterials*, vol. 11, no. 5, p. 1261, 2021.

[136] A. G. Alharbi and M. H. Chowdhury, "Memristor models and emulators: A literature review," *Memristor Emulator Circuits*, pp. 9–18, 2021.

[137] Intel, *Virtual JTAG Intel FPGA IP Core User Guide*, Intel. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/bhc1411109490717.html

[138] H. Zou, J. Huang, and M. Gao, "The application of virtual jtag technology in fpga design and debugging," in *2011 International Confer-*

ence on Electrical and Control Engineering. IEEE, 2011, pp. 2637–2640.

[139] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Transactions on circuits and systems*, vol. 35, no. 10, pp. 1257–1272, 1988.

[140] A. Rodriguez-Vazquez, J. Fernandez-Berni, J. A. Lenero-Bardallo, I. Vornicu, and R. Carmona-Galan, "Cmos vision sensors: embedding computer vision at imaging front-ends," *IEEE Circuits and Systems Magazine*, vol. 18, no. 2, pp. 90–107, 2018.

[141] R. Tetzlaff, A. Ascoli, I. Messaris, and L. O. Chua, "Theoretical foundations of memristor cellular nonlinear networks: Memcomputing with bistable-like memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 502–515, 2020.

[142] A. Ascoli, R. Tetzlaff, S. Kang, and L. O. Chua, "Theoretical foundations of memristor cellular nonlinear networks: A DRM_2-based method to design memcomputers with dynamic memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2020.

[143] M. M. Al Chawa, R. Picos, E. Covi, S. Brivio, E. Garcia-Moreno, and S. Spiga, "Flux-charge characterizing of reset transition in bipolar resistive-switching memristive devices," in *11th Spanish Conference on Electron Devices, 2017*. IEEE, 2017.

[144] R. Picos, J. Roldan, M. Al Chawa, F. Jimenez-Molinos, M. Villena, and E. Garcia-Moreno, "Exploring reram-based memristors in the charge-flux domain, a modeling approach," in *Memristive Systems (MEMRISYS) 2015 International Conference on*. IEEE, 2015, pp. 1–2.

[145] M. M. Al Chawa and R. Picos, "A simple quasi-static compact model of bipolar reram memristive devices," *IEEE Trans. on Circuits and Systems II: Express Briefs*, 2019.

[146] M. Itoh and L. O. Chua, "Designing cnn genes," *International Journal of Bifurcation and Chaos*, vol. 13, no. 10, pp. 2739–2824, 2003.

[147] M. Itoh, "Some interesting features of memristor cnn," *arXiv preprint arXiv:1902.05167*, 2019.

[148] D. Maldonado, M. B. Gonzalez, F. Campabadal, F. Jimenez-Molinos, M. M. Al Chawa, S. G. Stavrinides, J. B. Roldan, R. Tetzlaff, R. Picos, and L. O. Chua, "Experimental evaluation of the dynamic route

map in the reset transition of memristive rerams," *Chaos, Solitons & Fractals*, vol. 139, p. 110288, 2020.

[149] A. Ascoli, R. Tetzlaff, I. Messaris, S. Kang, and L. Chua, "Image processing by cellular memcomputing structures," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[150] S. Duan, X. Hu, Z. Dong, L. Wang, and P. Mazumder, "Memristor-based cellular nonlinear/neural network: Design, analysis, and applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 6, pp. 1202–1213, 2015.