POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Analyzing SCERPA and ToPoliNano Simulators for the Design of Molecular Field-Coupled Nanocomputing Circuits

Supervisors
Prof. Mariagrazia Graziano
Dr. Fabrizio Riente
Dr. Yuri Ardesi

Candidate
Dario Castagneri

April 2023

Abstract

Moore's Law, proposed by Intel co-founder Gordon Moore in 1965, forecast that the number of transistors on a microchip would double approximately every two years, paving the way to significant advancements in computing technology, efficiency, and performance. Initially, the projection proved to be highly accurate, describing the rapid progression of the semiconductor industry for over five decades. However, as the physical limitations of current transistor technology have been approached, researchers and engineers have questioned the long-term viability of Moore's Law, prompting the exploration of alternative nanoscale computing approaches, such as the Beyond CMOS. The paradigm involves the development of unconventional materials and device structures that are beyond traditional silicon technology to favor a continuous evolution of next-generation systems. Molecular Field-Coupled Nanocomputing (mFCN) is one of the most promising Beyond CMOS implementations. It relies on molecules arranged in nanolevel structures, with their charge distributions encoding logic information. Computation and signal propagation are enabled by exploiting Coulomb electrostatic interactions across molecules in cooperation with external electric fields. The mFCN approach offers several advantages over traditional CMOS digital systems, including significantly reduced power consumption, higher operating frequency, and highly scalable device sizes. To study the paradigm, the VLSI Lab group of the Politecnico di Torino has developed a collection of nanoelectronic applications: MagCAD, a graphical 3D editor used to design emerging computing devices; ToPoliNano, a CAD tool for studying and simulating these systems, recently embedded with SCERPA (Self-Consistent ElectRostatic Potential Algorithm), an academic MATLAB procedure for molecular circuit analysis; and FCNviewer, a 3D visualization program used to represent and check for correct information propagation and elaboration. The thesis aims at reviewing the integrability of the research-based algorithm in the ToPoliNano software and assessing the accuracy of the CAD simulation results. To achieve the goal, a comprehensive verification and validation strategy, supported by a series of comparing and testing scripts, is implemented and applied to a range of increasingly complex molecular circuits. Furthermore, new features and enhancements are introduced to both SCERPA and ToPoliNano suite, maximizing their compatibility, usability, and functionality. One specific improvement of the SCERPA algorithm is the introduction of a dynamic damping feature that accelerates the convergence of the method, reducing the time required for molecular analysis. The study also explores the possibility of designing novel and more complex molecular devices by using the newly verified and improved ToPoliNano package. In particular, it generalizes the 1-bit full adder layout to an N-bit version and successfully converts and validates the ISCAS c17 benchmark circuit into its molecular equivalent. In conclusion, the thesis has contributed to the field of molecular Field-Coupled Nanocomputing (mFCN) by providing researchers with an effective and reliable framework to investigate the behavior of mFCN systems.

Contents

List of Figures VII

List of Tables XI

List of Algorithms XII

List of Acronyms XIII

1 Introduction and background 1

1.1 More Moore . . . 2

1.2 More than Moore . . . 4

1.3 Beyond CMOS . . . 5

1.4 Field-Coupled Nanocomputing . . . 6

1.5 Cellular Automata . . . 7

1.6 Quantum-dot Cellular Automata . . . 7

1.6.1 Designing logic circuits . . . 9

1.6.2 Enhanced QCA cell . . . 11

1.6.3 Four-phase clocking system . . . 12

1.6.4 Materials and techniques for QCA fabrication . . . 13

2 Molecular Quantum-dot Cellular Automata 19

2.1 Promising candidate molecules for mQCA . . . 20

2.1.1 Dyallyl-butane . . . 22

2.1.2 Decatriene . . . 22

2.1.3 Bis-ferrocene carbazole . . . 23

2.2 Methodologies for analyzing mQCA systems . . . 24

2.2.1 Ab initio methods . . . 24

2.2.2 QCADesigner . . . 25

2.2.3 MosQuiTo . . . 26

2.2.4 Characterization of the single molecule . . . 26

2.2.5 Post-processing of results . . . 27

2.2.6 Analysis at the system-level . . . 32

2.3 Molecular FCN devices . . . 34

2.3.1 Single molecular cell . . . 34

2.3.2 Single phase wire . . . 34
2.3.3 Three phases wire . . . 35
2.3.4 Two-line three phases wire . . . 35
2.3.5 Two-line L-connection . . . 35
2.3.6 Two-line T-connection . . . 36
2.3.7 The inverter . . . 36
2.3.8 The majority voter . . . 38
2.3.9 The exclusive OR . . . 38
2.4 Manufacturing molecular FCN devices . . . 39

3 The Self-Consistent ElectRostatic Potential Algorithm (SCERPA) 42
3.1 The layout . . . 42
3.2 The algorithm . . . 44
3.3 The optimizations . . . 46
3.3.1 Interaction Radius . . . 46
3.3.2 Active Region . . . 46
3.4 The convergence . . . 48

4 The ToPoliNano EDA framework 50
4.1 ToPoliNano EDA framework . . . 51
4.1.1 ToPoliNano simulator . . . 52
4.1.2 MagCAD graphical editor . . . 52
4.1.3 FCNviewer . . . 53
4.2 Detailed overview of molecular ToPoliNano . . . 53

5 Comparative analysis of the molecular FCN simulators 61
5.1 Methodology . . . 62
5.2 Evaluating SCERPA importers . . . 63
5.2.1 Result discussion . . . 67
5.3 Evaluating SCERPA and ToPoliNano simulators . . . 67
5.3.1 Result discussion . . . 68
5.4 Molecule encoding checker . . . 70

6 Enhancing the SCERPA algorithm 72
6.1 Dynamic damping . . . 72
6.1.1 Assessing the impact of dynamic damping performance . . . 74
6.2 Charge convergence . . . 74
6.3 ToPoliNano clock waveforms . . . 76

7 Enhancing the ToPoliNano EDA framework 80
7.1 Parallelizing ToPoliNano simulator . . . 80
7.1.1 Assessing the impact of parallelization . . . 81
7.2 Introducing double-driver . . . 82
7.3 Enabling hierarchical layouts . . . 84

7.4 Toward information crossing 87
7.4.1 The new molecule in the ToPoliNano EDA framework 88
7.4.2 Simulating the crosswire with ToPoliNano 90
7.5 Consistent visualization in the FCNviewer 91

8 Advanced molecular FCN circuit design with the ToPoliNano EDA framework 95
8.1 QCA adders 96
8.2 Molecular 1-bit full adder 97
8.2.1 Layout 98
8.2.2 Simulation and testing 98
8.3 Molecular 2-bit full adder 99
8.3.1 Layout 100
8.3.2 Simulation and testing 100
8.4 Molecular N-bit full adder 100
8.5 Molecular ISCAS C17 101
8.5.1 Layout 103
8.5.2 Simulation and testing 104

References 116

Appendix A Comparative analysis of the molecular FCN simulators 117
A.1 Evaluating SCERPA importers 117
A.2 Evaluating SCERPA and ToPoliNano simulators 119
A.3 Molecule encoding checker 121

Appendix B Enhancing the SCERPA algorithm 123
B.1 Dynamic damping 123
B.2 ToPoliNano clock waveforms 124

Appendix C Enhancing the ToPoliNano EDA framework 126
C.1 Parallelizing the ToPoliNano simulator 126
C.2 Introducing double-driver 127
C.3 Enabling hierarchical layouts 129
C.4 Consistent visualization in the FCNviewer 130

List of Figures

1.1 Graphical representation of Moore's Law, showing the exponential increase in the number of transistors on integrated circuits over time. 2
1.2 Paradigms in the semiconductor industry. *Moore Moore*, *More than Moore*, and *Beyond CMOS* paradigms, along with their main characteristics. . . . 3
1.3 Numbering convention for the dots in a QCA cell used to compute the polarization of the cell. 8
1.4 Polarization states of a QCA cell representing logic '0' and logic '1' derived by the charge distribution within the quantum dots. 9
1.5 A binary wire in QCA designs comprised of six cells transmitting logic '1'. 9
1.6 Two different layouts for the QCA inverter. (a) The widely used *dual-branch* inverter, and (b) the alternative *single-branch* inverter. 10
1.7 Majority voter circuit implemented using QCA principles with three input cells, one evaluation cell, and one output cell. 11
1.8 Three states of a QCA cell: logic '0', logic '1' and null state. 12
1.9 Trapezoidal clock waveforms scheme for reliable and smooth information transmission in complex QCA layouts. 13
1.10 Four-phase clocking scheme. Adiabatic switching ensures smooth transition and propagates information among clock zones. 14
1.11 Two logical encoding states of a metal-dot QCA cell. 15
1.12 Three states of a magnetic QCA cell. Up state encoding logic '0', down state encoding logic '1', and null state. 17
1.13 Examples of MQCA devices: (a) Magnetic QCA wire. (b) Magnetic QCA majority voter. 18

2.1 Encoding states of a 2-dots oxidized molecule: logic '0', transient state, and logic '1'. 19
2.2 Three states of a molecular QCA cell with three dots. 20
2.3 Structural and schematic representations of the dyallyl-butane molecule. . 22
2.4 Structural and schematic representations of the decatriene molecule. . . . 23
2.5 Structural and schematic representations of the bis-ferrocene carbazole molecule. 24
2.6 Switching field effects on a bis-ferrocene molecule. A positive field restrains the charge in Dot2, whereas a negative field confines it in Dot1. . . 27

2.7 Clock field effects on a bis-ferrocene molecule. A positive field facilitates the molecule's transition to an encoding state, whereas a negative field causes the charge to be pushed into the null state. 28
2.8 Charge distribution of the bis-ferrocene molecule alongside its aggregated charges. 29
2.9 Interactions in a complete mQCA cell. (a) 3D view and (b) top view. . . 31
2.10 $V_{in} - V_{out}$ Transcharacteristics of a complete bis-ferrocene cell subjected to three clock fields. 31
2.11 V_{in}–AC transcharacteristics of a bis-ferrocene molecule subjected to three clock fields. 32
2.12 MosQuiTo schematic model for system-level interactions. 33
2.13 Schematic representation of the molecular wire. 34
2.14 Single mFCN cell layout made of a driver, two adjacent molecules and an output. 34
2.15 Single phase mFCN wire layout made of six cells. 35
2.16 Three phases mFCN wire layout made of twelve cells. 35
2.17 Two-line three phases mFCN wire layout. 36
2.18 Two-line mFCN L-connection layout. 36
2.19 Two-line mFCN T-connection layout. 37
2.20 Two-line mFCN inverter gate layout. 37
2.21 Two-line mFCN majority voter layout. 38
2.22 XOR gate implemented in the digital domain with a majority voter and inverter-based logic. 39
2.23 Two-line mFCN XOR gate layout. Each square corresponds to an mQCA cell, consisting of two adjacent bis-ferrocene molecules. 40
2.24 Top view representation of a practical clocked wire implementation. . . . 41

3.1 SCERPA directory structure. 43
3.2 Schematic of SCERPA highlighting the main sections. 43

4.1 The ToPoliNano EDA framework. 51
4.2 ToPoliNano's EDA framework flow to desing, simuale and validate mFCN circuits. 53
4.3 Single-line mFCN majority voter layout implemented with MagCAD editor. 54
4.4 ToPoliNano's GUI to specify simulation parameters. 56
4.5 FCNviewer graphical representation of the majority voter simulated at four different time instances: (a) when all cells, except drivers, are in reset, (b) the propagation of the driver value to the central cell, (c) the evaluation of the central cell based on the most recurrent input, and (d) the final propagation of the result toward the output. 59
4.6 FCNviewer graphical representation of the clock waveforms and the charge variations of the output cell dots during the simulation. 60

6.1 Dynamic damping analysis for the inverter gate. 75

6.2 Dynamic damping analysis for the majority voter gate. 76
6.3 The effects of dynamic damping on simulation time for the inverter gate compared to static damping. 77
6.4 The effects of dynamic damping on simulation time for the majority voter gate compared to static damping. 78
6.5 ToPoliNano clock waveforms generated with `nSteps = 20` and `N = 4`. . . 78

7.1 Multi-threaded and single-thread simulation times for a set mFCN circuits. 82
7.2 The effects of parallelization on simulation time for a set of mFCN circuits compared to single-thread. 83
7.3 Single (a) and double-driver (b) representation of a majority voter. 84
7.4 One-level hierarchical design applied to the XOR gate. 87
7.5 Neutralized bis-ferrocene molecular QCA cell. 88
7.6 mFCN crosswire layout made of neutral oxidized bis-ferrocene molecules. 89
7.7 V_{in}–aggregated charge transcharacteristics of the neutralized oxidized bis-ferrocene molecule subjected to two clock fields. 89
7.8 The new ToPoliNano molecular GUI with the driver molecule settings. . . 91
7.9 Two mFCN crosswire layouts. (a) mFCN crosswire layout for same inputs. The input branches are synchronized by the same clock phase, and the output branches share the same clock to forward propagate the data crossing. (b) mFCN crosswire layout for opposite inputs. The input and output branches are synchronized with the same clock phase, allowing the last cell of one input branch to influence the first cell of the other branch without interference from the evaluation cell. 92
7.10 ToPoliNano simulations for the crosswire layouts. (a) Drivers encode same logic '0' state. (b) Drivers encode two opposite logic states. 92
7.11 mFCN wire layout generated by MagCAD with shifts and rotations applied to molecules. 93
7.12 mFCN wire layout represented by the SCERPA 3D viewer with shifts and rotations applied to molecules. 94
7.13 Top view of the modified mFCN wire layout generated by the FCN viewer. 94
7.14 Lateral view of the modified mFCN wire layout generated by the FCN viewer. 94

8.1 Schematic flow adopted to design, simulate and test the novel molecular architectures. 96
8.2 One of the earliest 1-bit full adder layouts proposed for QCA technology and implemented with a majority voter and inverter-based logic. 97
8.3 Optimized 1-bit full adder implemented with a majority voter and inverter-based logic. 98
8.4 1-bit molecular full adder implemented with a robust two-line fashion with a majority voter and inverter-based logic. Each square corresponds to an mQCA cell, consisting of two adjacent bis-ferrocene molecules. 99

8.5 Optimized 2-bit full adder implemented with a majority voter and inverter-based logic. 101
8.6 Optimized 2-bit full adder highlighting majority voter and inverter blocks and the double stage structure. Each square corresponds to an mQCA cell, consisting of two adjacent bis-ferrocene molecules. 102
8.7 Schematic representation of an N-bit full adder layout. 102
8.8 N-bit full adder hierarchical layout. 103
8.9 ISCAS C17 digital layout. 104
8.10 ISCAS C17 digital circuit implemented with duplicated inputs to overcome the need to use crosswire. 105
8.11 Two-line mFCN 2-input NAND layout employed a fundamental block to realize the mFCN ISCAS C17 circuit. 105
8.12 ISCAS C17 molecular circuit highlighting the fundamental NAND logic gates. Each square corresponds to an mQCA cell, consisting of two adjacent bis-ferrocene molecules. 107
8.13 ISCAS C17 molecular circuit hierarchically implemented with NAND gates. 108

List of Tables

1.1 Inverter truth table. 10
1.2 3-input majority voter truth table. 11

2.1 Exclusive-OR truth table. 39

7.1 Crosswire truth table. 88

8.1 1-bit full adder truth table. 97
8.2 NAND truth table. 104
8.3 ISCAS C17 truth table. 106

List of Algorithms

1 SCERPA pseudocode 47
2 ToPoliNano's molecular core pseudocode. 58
3 Pseudocode for printing molecules IDs and Dot4 coordinates. 65
4 Pseudocode for matching molecule IDs in MATLAB and MagCAD. 65
5 Pseudocode for comparing MATLAB and MagCAD ACs. 66
6 Pseudocode for comparing ToPoliNano and SCERPA ACs. 69
7 Pseudocode for validating cell encoding for a given molecule at step τ. . . 71

List of Acronyms

AC Aggregated Charge

AIMD Ab initio Molecular Dynamics

AR Active Region

CA Cellular Automata

CAD Computer-Aided Design

CMOS Complementary Metal-Oxide Semiconductor

CNTFET Carbon Nanotube Field-Effect Transistor

DFT Density Functional Theory

ED Ehrenfest Dynamics

EDA Electronic Design Automation

FCN Field-Coupled Nanocomputing

FinFET Fin Field-Effect Transistor

GAA-FET Gate-All-Around Field-Effect Transistor

GUI Graphical User Interface

HDL Hardware Description Language

HF Hartree–Fock

HOMO Highest Occupied Molecular Orbital

IC Integrated Circuit

iNML In-Plane Nano-Magnetic Logic

IR Interaction Radius

IRDS International Roadmap for Devices and Systems

IRTS International Technology Roadmap for Semiconductors

ISCAS International Symposium on Circuits and Systems

MC Monte Carlo

mFCN Molecular Field-Coupled Nanocomputing

MosQuiTo Molecular Simulator Quantum-dot cellular automata Torino

MQCA Magnetic Quantum-dot Cellular Automata

mQCA Mmolecular Quantum-dot Cellular Automata

MUT Molecule Under Test

NML Nano-Magnetic Logic

NWFET Nanowire Field-Effect Transistor

PCB Printed Circuit Board

pNML Perpendicular Nano-Magnetic Logic

QCA Quantum-dot Cellular Automata

RT-TDDFT Real-Time Time-Dependent Density Functional Theory

SAM Self-Assembled Monolayer

SCERPA Self-Consistent ElectRostatic Potential Algorithm

SCF Self-consistent Field

TFET Tunnel Field-Effect Transistor

ToPoliNano TOrino POLItecnico NANOtechnology

TSA Two-State Approximation

VACT Vin-Aggregated Charge Transcharacteristic

VHDL VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

VVT Vin-Vout Transcharacteristic

Chapter Summaries

The following paragraphs provide an overview of the document's organization and highlight key topics discussed in each chapter.

Chapter 1: Introduction and background

Chapter 1 provides an overview of the challenges facing traditional silicon technology and the emergence of the Momolecular Field-Coupled Nanocomputing (mFCN) paradigm as a promising solution to overcome these limitations. The chapter highlights the advantages of mFCN implementations, including speed, power, and scalability benefits.

Chapter 2: Molecular Quantum-dot Cellular Automata

Chapter 2 delves deeper into the Molecular Quantum-dot Cellular Automata (mQCA), one of the key implementations in the FCN paradigm. It begins by introducing the main characteristics of the approach, including the different types of molecules proposed in the literature, and the strategies used to analyze them. It then discusses the MosQuiTo methodology, a promising approach for modelling molecules and examining interactions. Finally, the chapter explores the challenges involved in manufacturing practical devices.

Chapter 3: The Self-Consistent ElectRostatic Potential Algorithm (SCERPA)

Chapter 3 focuses on SCERPA, a state-of-the-art mFCN research algorithm that employs a self-consistent iterative loop to solve electrostatic interactions among molecules, offering an efficient alternative to computationally expensive ab initio methods. The chapter provides a comprehensive description of the procedure, including its main steps and key characteristics.

Chapter 4: The ToPoliNano EDA framework

Chapter 4 introduces the ToPoliNano EDA Framework developed to bridge the gap between academic research and real-world EDA applications in emerging technologies. The chapter provides detailed information about the molecular simulation core of the framework, which leverages a similar approach to SCERPA to enable the study of mFCN circuits.

Chapter 5: Comparative analysis of the molecular FCN simulators

Chapter 5 aims to validate the layout importers of SCERPA and compare the results obtained between the academic and CAD simulators. The chapter explains the methodology used to automate the comparison process and provides a thorough discussion of the results, highlighting key discoveries.

Chapter 6: Enhancing the SCERPA algorithm

Chapter 6 introduces improvements made to SCERPA, including the addition of dynamic damping to promote faster analysis and a charge convergence method to assess stability. The algorithm is also equipped with clock waveforms to regulate information propagation throughout circuits, similar to the ToPoliNano simulator.

Chapter 7: Enhancing the ToPoliNano EDA framework

Chapter 7 describes novel features and enhancements introduced to the ToPoliNano framework to bridge the gap between academic research and real-world EDA applications. These include parallelizing the molecular engine, supporting hierarchical design, and introducing new molecule for more flexible layouts.

Chapter 8: Advanced molecular FCN circuit design with the ToPoliNano EDA framework

Chapter 8 showcases the EDA suite's ability to design, simulate, and test complex molecular circuits. The chapter includes demonstrations of transposing a 1-bit full adder to a 2-bit version and adapting a digital benchmarking circuit into its molecular equivalent.

Conclusions and future perspectives

The final chapter summarizes the key points discussed in this work and emphasizes the potential of mFCN technology in revolutionizing computation. Moreover, it highlights a possible area for continued research and development in the field.

Appendices

The appendices provide the code developed and presented in the chapters of this work.

Chapter 1

Introduction and background

The semiconductor industry is one of the very few industries that began with a roadmap even before its actual start. In 1965, Intel co-founder Gordon Moore predicted that the number of transistors in integrated circuits would double roughly every two years [43]. The forecast, known as Moore's law, has proven to be remarkably accurate, resulting in significant improvements in computing technology, efficiency, and performance over the past five decades, making it one of the most impressive and enduring technological achievements in history [24]. Figure 1.1 shows a graphical representation of Moore's Law.

Over time the transistor count regularly increased while the size of each transistor consistently decreased. As the transistor sizes shrunk, the actual density - defined as the number of transistors per unit area - was raised, enhancing the functionality of a given chip size. This aspect allowed the fabrication of more integrated circuits for each wafer, thus reducing manufacturing costs. In addition, bigger wafer sizes, made possible by the evolution and optimization of the lithography processes, kept production costs low. Another advantage of a scaled transistor was the reduction of its switching time which reduced power consumption along with a trend that used lower and lower supply voltages.

It is clear how the scaling transistor phenomenon was initially a success since it helped manufacture more compact, faster and more efficient chips. However, as Moore pointed out, this phenomenon would eventually face physical limitations. Not surprisingly, in the 70s already, researchers and engineers started to be sceptical about the actual duration of Moore's law and tried to forecast its conclusion or its slowdown [57]. The main factors arguing Moore's exceptional prediction accuracy are related to an imminent reach of atomistic and quantum mechanical boundaries and to the constraints imposed at technological, economic and fabrication levels. For these reasons, scientists and researchers have started examining ways to keep Moore's Law exponential trend on track by investigating alternative computing paradigms to substitute traditional silicon technology.

At the current time, Figure 1.2 shows the three main paradigms the semiconductor industry can rely on: *Moore Moore*, *More than Moore* and *Beyond CMOS* [44, 45, 16].

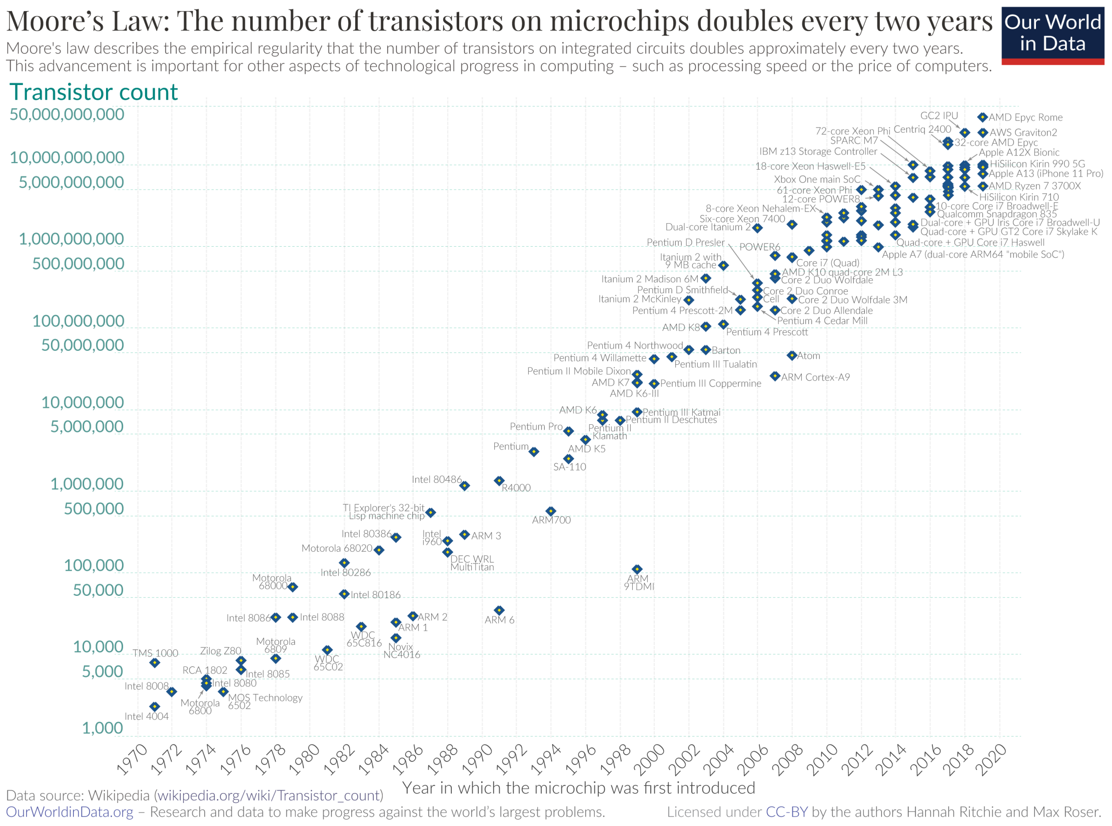


Figure 1.1. Graphical representation of Moore's Law, showing the exponential increase in the number of transistors on integrated circuits over time.

1.1 More Moore

The term *More Moore* attempts to achieve higher performance and lower production costs through continuous scaling of electronic devices, in line with the principle that enabled Moore's prediction to become a reality. This approach can pursue several methods, including traditional geometric scaling and non-geometric strategies focusing on other system qualities, such as improved electron mobility or reduced power consumption.

Geometric scaling involves shrinking the size of transistors, which increases the number of transistors per unit area, resulting in more efficient and higher-performing devices.

On the contrary, non-geometric techniques leverage new materials for manufacturing chips, such as germanium, or employ 3D integration to stack multiple layers of transistors, packing more devices into a smaller area. In addition to scaling, the paradigm emphasizes digital information processing and data storage, thanks to advancements in microprocessors, logic devices, and memories [44].

This approach enabled a smooth transition beyond the limitations of planar CMOS transistor technology, which faced issues with gate leakage and power consumption as

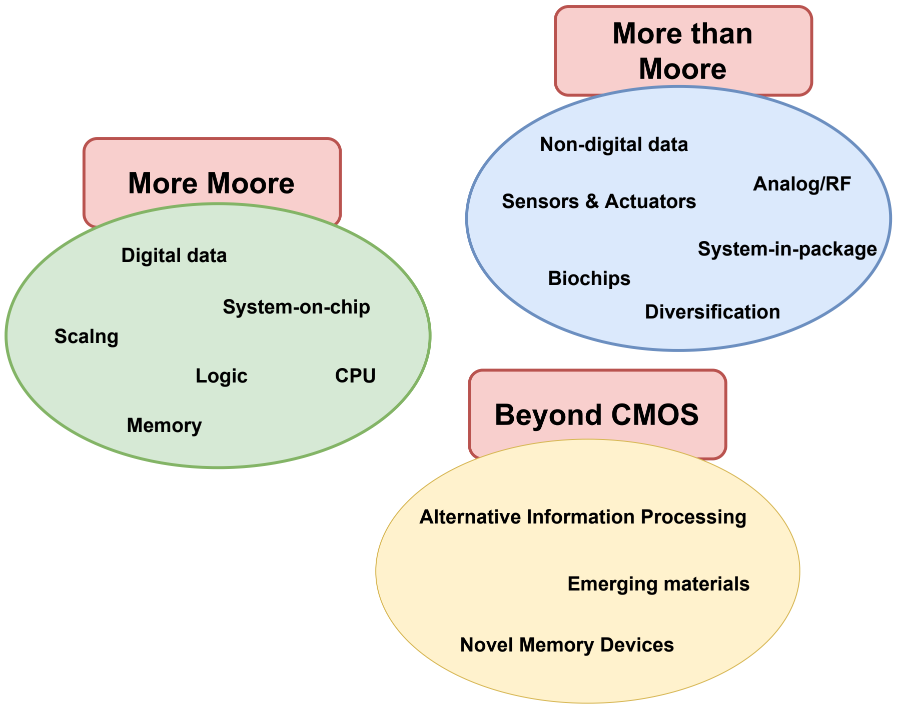


Figure 1.2. Paradigms in the semiconductor industry. *Moore Moore*, *More than Moore*, and *Beyond CMOS* paradigms, along with their main characteristics.

sizes continued to shrink. To address these challenges, researchers engineered the Fin Field-Effect Transistor (FinFET), which employs a three-dimensional device surrounded by a gate electrode that protrudes from the substrate. This structure provides better gate control, reducing leakage and improving energy efficiency. FinFETs also offer superior electrostatic control, improving transistor performance and reducing the impact of variability in the manufacturing process [20].

While the semiconductor industry has widely adopted FinFET technology, ongoing research and advancements in the field continue to lead to the introduction of new and improved versions of FinFETs, demonstrating the importance of the *More Moore* paradigm in the development of future technology transistors [44]. Some examples are:

- **Gate-All-Around Field-Effect Transistor (GAA-FET)**: The gate electrode surrounds the channel, providing better control over the current flow. Moreover, the design results in faster switching speeds, lower power consumption, and higher reliability.

- **Tunnel Field-Effect Transistor (TFET)**: The operation is based on quantum tunneling, offering reduced power consumption and better performance, especially at low voltages. Nevertheless, limitations still exist around fabrication, optimization, and high-quality tunneling materials.

- **Nanowire Field-Effect Transistor (NWFET)**: Nanowires made of silicon (Si), germanium (Ge) or indium arsenide (InAs) are employed for channel manufacture, offering advantages in terms of electrostatic control, scalability, and reduced short-channel effects. Nonetheless, obstacles remain in fabrication difficulties, reliability and correct sizing.
- **Carbon Nanotube Field-Effect Transistor (CNTFET)**: They replace the active channel of traditional transistors with nanoscale graphene structures called carbon nanotubes. They offer advantages such as improved performance and low-power consumption thanks to high carrier mobility. However, their fabrication and reliability present significant challenges.

The *More Moore* paradigm, in general, poses significant challenges in the short-term and long-term. These include physical, material, power and thermal, technological, and economical concerns [31, 44]:

- **Physical**: Leakage losses become more pronounced as devices are miniaturized, affecting their performance, stability and reliability.
- **Material**: Finding suitable materials for electronic device fabrication is difficult, especially for providing adequate conduction and insulation properties. This issue becomes increasingly important as device scaling continues.
- **Power and thermal**: As the number of transistors per unit area increases, so do power consumption and thermal activity. Proper management of these challenges is crucial to prevent overheating and device failure.
- **Technological**: Lithographical-based processes used for device manufacturing have limitations that become critical as device dimensions decrease. Ensuring the accuracy of these processes is crucial for correct device operation.
- **Economical**: Fabrication costs, including masks, wafer production, and testing, increase as sizes shrink. There is a risk that these costs may become too expensive, making *More Moore* economically unfeasible.

1.2 More than Moore

The expression *More than Moore* was proposed by the International Technology Roadmap for Semiconductors (IRTS) in 2005 [45]. It represents a significant shift in the semiconductor industry away from merely scaling transistors and toward integrating more functionalities and components into integrated circuits [28].

This trend has emerged in response to the observation that many of the market's application requirements, such as high performance, low power consumption, advanced RF communication, and the integration of sensors and actuators, cannot be met solely through Moore's Law [45]. The inability to meet these demands is mainly due to the unique and diverse needs of specific applications rather than limitations in technology development. Therefore, the approach seeks to integrate analog and mixed-signal technologies with standard silicon-based computation to create heterogeneous systems that can meet the complex demands of applications.

More Moore and *More than Moore* are not seen as two competing paradigms, but rather as complementary strategies for creating higher-value systems.

More than Moore systems are based on three main features [28, 59]:

- **Multidisciplinarity**: This principle involves knowledge integration from multiple fields into a single device to meet the specifications of the application domain. Different areas of expertise, such as physics, materials science, and computer engineering, offer *More than Moore* devices better performance and functionality compared to traditional, single-discipline approaches.
- **Heterogeneity**: The approach emphasizes the integration of multiple digital and non-digital components, such as sensors, actuators, and communication interfaces, which cooperate to achieve the desired device goals. By incorporating diverse functionalities, a *More than Moore* system can provide more comprehensive solutions for a wide range of applications.
- **Reliability**: A reliable system requires a suitable testing environment to validate its behavior in different conditions.

The *More than Moore* paradigm presents some issues that require attention. One of the major obstacles is the integration of digital and non-digital components, which requires software development to bridge the gap between the two worlds [45]. Despite these difficulties, there has been a growing interest in the *More than Moore* approach in recent years, particularly in domains that require integrating digital elements for computation and non-digital components for application-specific information extraction [28].

1.3 Beyond CMOS

Beyond CMOS refers to a class of emerging technologies and materials that seek to overcome the limitations of current silicon technology [16, 21]. These innovative alternatives introduce novel approaches to information computing and storage for addressing the challenges of power consumption, scalability, and performance that are becoming increasingly problematic for standard electronics. *Beyond CMOS* technologies have the potential to replace or supplement current silicon generation, leading to faster and more energy-efficient devices. Examples of technologies include:

- **Spintronics**: The technology exploits the spin of electrons as an alternative way of storing and processing information. The spin is manipulated using different techniques like magnetic fields or other spin interactions. This mechanism can be applied to manufacture new types of memory and logic devices with better energy efficiency, higher density, and additional functionalities.

- **Quantum Computing**: The paradigm leverages the properties of quantum mechanics to process and store data. Quantum bits represent not only the traditional binary states but also any simultaneous combination of them, thanks to the phenomenon of quantum superposition. The aspect enables quantum computing systems to potentially solve problems that classical computers cannot efficiently handle, such as breaking encryption. Nonetheless, challenges such as fragile qubits, scaling up, and maintaining consistency over time to be faced to reach their full potential.

- **Field-Coupled Nanocomputing**: The technology encodes digital information in the physical properties of nanoscale components. By leveraging local field interactions, information is transferred element by element. One example of this approach is Quantum-dot Cellular Automata (QCA), which uses the charge states of quantum dots to propagate information and perform logical operations through Coulomb interactions. It offers numerous advantages over traditional computing technologies, including higher device density and lower power consumption. Moreover, there are diverse materials for practical manufacturing, such as metals, semiconductors, magnets, and molecules.

1.4 Field-Coupled Nanocomputing

Field-Coupled Nanocomputing (FCN) is a novel class of computational technology that goes beyond the limitations of traditional silicon-based computing. While silicon-based systems have been the primary means of digital processing and storage for several decades, they are currently facing significant challenges from power consumption, scalability, and speed perspectives. In contrast, FCN offers a promising solution to overcome these issues, providing an alternative approach to computing.

Unlike traditional silicon elaboration, FCN does not rely on transistors or charge transportation to process information. Instead, it exploits local field interactions between nano-size modules organized in well-defined structures and designed to interact with each other. This characteristic is derived from the theory of cellular automata (CA), a mathematical concept for processing and storing information in a decentralized way [32].

FCN offers an unprecedented combination of higher computational performance and exceptionally low power consumption compared to current silicon devices. Furthermore, its modular design allows for a straightforward scaling up or down by adding or removing the nano-modules as needed, providing exceptional scalability and flexibility.

The development of FCN technology has been a collaborative effort of researchers, resulting in multiple possible technology implementations, each focusing on unique nano-level structures, properties, and features. One of the most promising approaches is the Quantum-dot Cellular Automata (QCA), which lays the foundations to develop both magnetic (MQCA) and molecular (mQCA) variants. These QCAs leverage the unique properties of electrons and the magnetic spin of molecules to store and process information, offering a multitude of promising paradigms beyond the conevntional computing scenario.

1.5 Cellular Automata

Cellular Automata (CA) is a theoretical mathematical concept that allows the study of complex systems and processes using simple, local rules of interaction applied in a decentralized manner. It is a discrete-time, discrete-space model composed of a number of identical, locally interacting elements arranged in a regular grid.

Each cell has a finite number of possible states and evolves according to a fixed set of rules that depend only on the states of its neighbors at the previous time step. The rules are often expressed in terms of a transition function that takes the current state of a cell and its neighbors as input and produces a new state for the cell as output. The system evolves in parallel and synchronously as a sequence of snapshots, each representing the state at a particular instant in time.

One of the most remarkable features of CA is its ability to describe and simulate complex behavior patterns that emerge from the simple interactions between individual components, making it possible to describe a wide range of phenomena with appropriate rules and initialization. Various fields, including physics, biology, electronics, and computer science, leverage this property.

In particular, researchers have explored CA as a basis for building computational systems, with one of the most promising realizations being the Quantum-dot Cellular Automata (QCA). This approach encodes digital information in the physical properties of nanoscale components and uses local field interactions to transfer information element by element, enabling higher device density and lower power consumption.

1.6 Quantum-dot Cellular Automata

Quantum Cellular Automata (QCA) is a highly promising technology for Beyond CMOS implementations, as it enables information processing and storage at the nanoscale level. QCA derives from the classical Cellular Automata (CA) paradigm, integrating quantum mechanical principles to process and store information. The concept was first introduced in the 1990s by Lent et al. [34].

In QCA, the traditional cells of CA consist of quantum dots, which are semiconductor structures at the nanoscale capable of confining electrons in three dimensions, resulting in the quantum mechanical properties of quantization of energy levels and electron tunneling.

The paradigm exploits these properties to enable fine-tuned information processing and storage down to the electron level. The elementary QCA cell is square-shaped and represented by two potential zones. The high potential zones represent quantum dots, while the low potential zones act as barriers that realize quantum confinement.

Each QCA cell contains a pair of free-to-move electrons, which behaves according to Coulomb's law. These electrons tend to occupy the farthest possible spots from each other, minimizing the electrostatic repulsion force. As a result, each cell acquires one of two energetically equivalent arrangements in which both electrons locate in the antipodal quantum dots. This binary state of the cell encodes information, with one configuration corresponding to a logic '1' and the other to a logic '0'.

Unlike conventional digital computing, which uses voltage levels to represent information, QCA relies solely on the electrostatic coupling interactions among cells, without charge movements outside them. This approach allows for high speed, low power, and scalability, making it highly attractive for alternative computing. Despite numerous advantages over traditional CMOS technology, the practical implementation of QCA presents significant challenges, especially when scaling up to larger sizes and complexities.

The polarization of a QCA cell is a measure of the charge distribution within the cell and is computed as:

$$P = \frac{(\rho_2 + \rho_3) - (\rho_1 + \rho_4)}{(\rho_1 + \rho_2 + \rho_3 + \rho_4)} \tag{1.1}$$

where ρ_i represents the electronic charge at quantum dot i, and the numbering of dots is generally decided according to a standard convention, as shown in Figure 1.3. The polarization of the stable states is $+1$ or -1 and is used to encode binary information of '1' and '0', respectively. Figure 1.4 illustrates the two encoding states of a QCA cell.

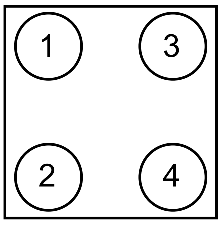


Figure 1.3. Numbering convention for the dots in a QCA cell used to compute the polarization of the cell.

Interactions between neighbouring cells are achieved through Coulomb interactions, which arise from the repulsion between the electrons in adjacent dots, resulting in a local electric field exploitable for transferring information.

The simplest QCA structure consists of two QCA cells placed at a proper distance from each other. The first cell, named driver, is polarized by forcing its electrons into one of two allowed configurations. The second cell adopts a state that minimizes Coulomb repulsion, which is the same as the driver's state, allowing encoded binary information to propagate from the first cell. Electrons within the second cell move solely toward a logic state of '0' or '1' based on the driver's polarization, without any current flows, resulting in low-power dissipation and high-speed operation.

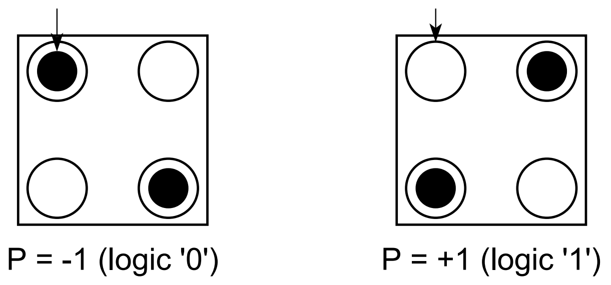


Figure 1.4. Polarization states of a QCA cell representing logic '0' and logic '1' derived by the charge distribution within the quantum dots.

Following the same principle, binary wire is created by placing multiple QCA cells next to each other in the same axis direction [25]. The Coulomb repulsion between adjacent cells rearranges the positions of electrons in all consecutive cells, allowing input information to propagate to the last cell. This collective movement of charges is usually referred to as the *domino effect* and facilitates the propagation of input information. Figure 1.5 depicts a binary wire comprised of six cells transmitting logic '1'.

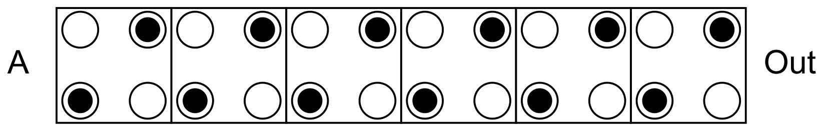


Figure 1.5. A binary wire in QCA designs comprised of six cells transmitting logic '1'.

1.6.1 Designing logic circuits

QCA cells can be arranged in specific structures to create a variety of logic computation devices. The inverter and the majority voter are the two fundamental devices in QCA technology, and any Boolean circuits can be designed by combining them [25].

The inverter

The inverter is a fundamental building block of digital circuits, responsible for transforming an input signal into its logical complement, based on the truth table provided in Table 1.1. In QCA designs, the most commonly used structure is the *double-branch* inverter, illustrated in Figure 1.6(a). The layout employs a dual-stage complement process, which ensures efficient and reliable information inversion.

In contrast, Figure 1.6(b) showcases an alternative layout known as the *single-branch* inverter, which utilizes a single-stage diagonal interaction process across two cells. This design results in more compactness at the expense of worse reliability than the dual-branch version. Nonetheless, it is often preferred when the area is critical and the circuit is less inclined to noise. Although the *single-branch* inverter can operate within the standard QCA paradigm, it can only work with neutral or zwitterionic molecules in the molecular counterpart, as demonstrated in [4].

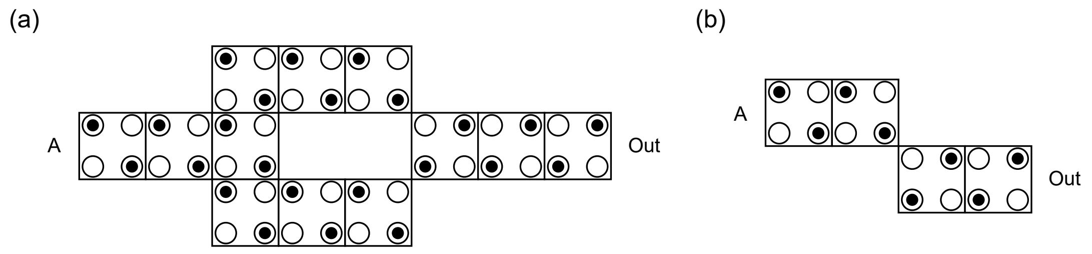


Figure 1.6. Two different layouts for the QCA inverter. (a) The widely used *dual-branch* inverter, and (b) the alternative *single-branch* inverter.

A	Out
0	1
1	0

Table 1.1. Inverter truth table.

Majority Voter Gate

In addition to the inverter, the majority voter is another crucial component in the design of QCA circuits. It acts as a logic computation device, producing an output corresponding to the most frequent input signal. Table 1.2 and Figure 1.7 show the truth table and layout of a 3-input majority voter, respectively. The structure contains three inputs, one output, and an evaluation cell. The evaluation cell is always polarized to the majority state, as this minimizes the Coulombic repulsion between the electrons of the three inputs.

The relationship between the inputs and the output of the majority gate can be reported as:

$$MV(A, B, C) = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C) \tag{1.2}$$

where $MV()$ denotes a majority voter operation and $\wedge$ and $\vee$ denote AND and OR operations, respectively.

By setting one of the three inputs of the majority gate to a constant polarization, it is possible to implement AND and OR gates. Specifically, when a single input polarizes with $P = -1$, the output is an AND of the other two inputs:

$$MV(0, B, C) = B \wedge C \tag{1.3}$$

Similarly, when a single input polarizes with $P = 1$, the output is an OR of the other two inputs:

A	**B**	**C**	**Out**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 1.2. 3-input majority voter truth table.

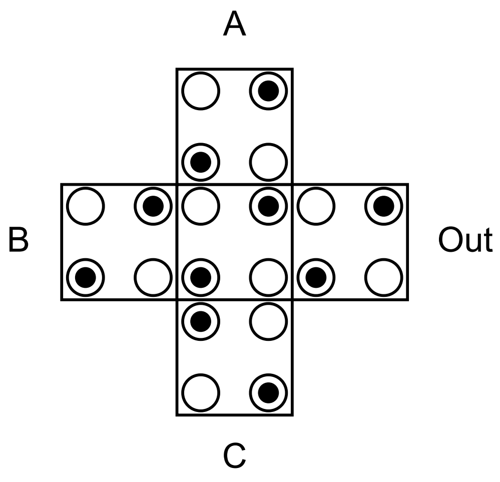


Figure 1.7. Majority voter circuit implemented using QCA principles with three input cells, one evaluation cell, and one output cell.

$$MV(1, B, C) = B \vee C \tag{1.4}$$

1.6.2 Enhanced QCA cell

The QCA paradigm relies on the assumption that each cell can switch between its two possible states seamlessly due to a low potential barrier between them while maintaining stable logic values thanks to a high potential barrier [63]. Inputs propagate in a specific direction, typically from left to right. Thus any loss of information is minimized. An edge-driven approach is employed to favor data propagation. The strategy involves changing the input condition of the circuit via the domino effect.

However, this technique has limitations, particularly with circuit size, as larger layouts tend to stabilize in unwanted metastable configurations. This behavior can corrupt the information flow and lead to data loss, making the paradigm challenging to apply in practice [47, 35].

To address these challenges, an enhanced QCA cell with two additional central dots that define a non-encoding null or reset configuration is introduced. The new cell includes two additional central dots that define a non-encoding *null* or *reset* configuration, which can only be accessed by electrons when an external signal is active [17]. These dots play a crucial role in maintaining stable logic values during switching events by increasing the distance between charge carriers and raising the potential barrier between the two logic configurations. This behavior results in more stable states and minimizes the possibility of errors. Additionally, the dots reduce the barrier for switch and transmission events, which ensures low-power consumption during switching [63, 47]. Figure 1.8 illustrates the new QCA cell and its available configurations.

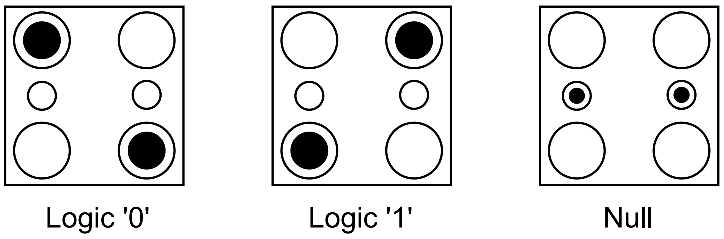


Figure 1.8. Three states of a QCA cell: logic '0', logic '1' and null state.

An external electric field handles the null state. While ideally, a perfect square wave should be employed, this is practically impossible due to the absence of instantaneous transitions. Therefore, practical scenarios leverage approximated trapezoidal waves. An active clock signal gradually pushes charges toward one of the stable states. Once the clock signal is released, the cell assumes the same configuration as the input [35, 17].

Complex circuits require adiabatic and stable signal transfer to ensure reliable operation. The layout is organized into zones controlled by a separate clock signal. These clock signals are partially superposed with adjacent zones, typically through a phase shift of $\pi/2$, to smoothly synchronize the transitions between clock zones in a pipeline fashion [35]. Each zone has a maximum number of cells to prevent data loss and assure integrity during signal transmission. Trapezoidal signals are utilized to ensure proper system function, even in the case of jitter, which can occur due to minimal variations in signal clock timing [47].

1.6.3 Four-phase clocking system

The external clock signal plays a critical role in maintaining the correct polarization of cells and ensuring reliable encoding and propagation of information [35, 10]. It consists of four distinct phases, each serving a specific purpose. Figures 1.9 and 1.10 report a graphical representation of the four-phase clocking scheme. The phases include:

- **Switch phase**: The clock is gradually intensified to slowly increment the inter-dot potential barriers. Therefore, cells encode and propagate information, moving from the null configuration to the same logical state of neighboring cells.
- **Hold phase**: The clock and the inter-dot potential barriers reach their maximum value, leading to clear polarization of the cells. As a result, cells remain locked in their encoding state, acting as drivers for the next phase.
- **Release phase**: The clock is gradually released such that the potential barriers between the logical and the middle dots are decremented. Therefore, the mobile charges tunnel from the logic toward the intermediate dots, and the cell polarization gradually becomes less clear, heading to the loss of the logical state. Once the clock field is entirely released, the cells assume the null configuration.
- **Relax phase**: The potential barriers between adjacent cells reach their minimum value, effectively isolating each cell from its neighbors in other clock zones. As a result, a buffer zone facilitates a smooth transition to the next clock phase. The duration of this phase is long enough to allow any residual charges in the cells to dissipate, thus preventing any degradation of the information during the next clock period.

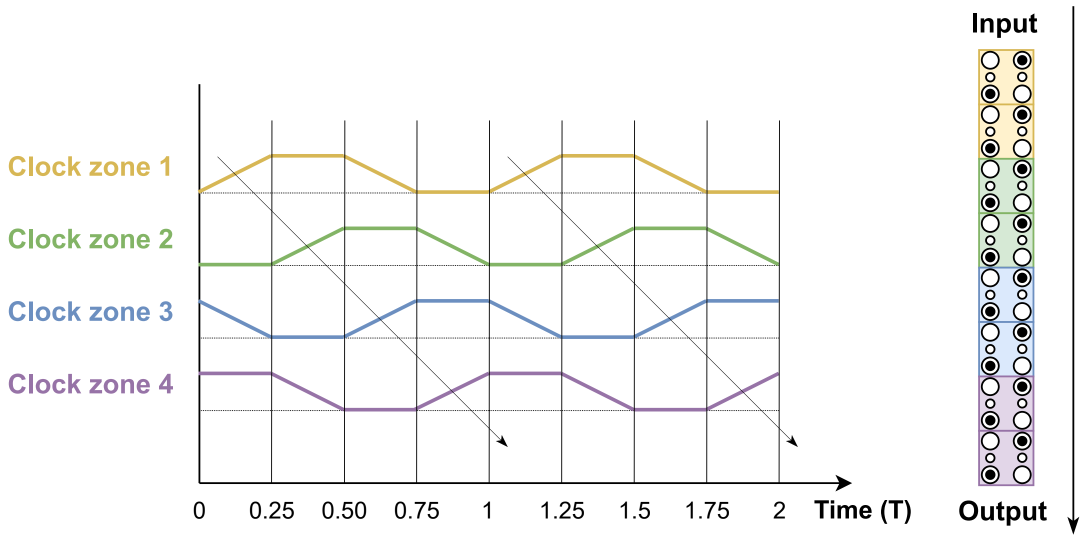


Figure 1.9. Trapezoidal clock waveforms scheme for reliable and smooth information transmission in complex QCA layouts.

1.6.4 Materials and techniques for QCA fabrication

Over the years, numerous physical implementation approaches have been proposed to advance the Quantum Cellular Automata paradigm. These strategies typically involve

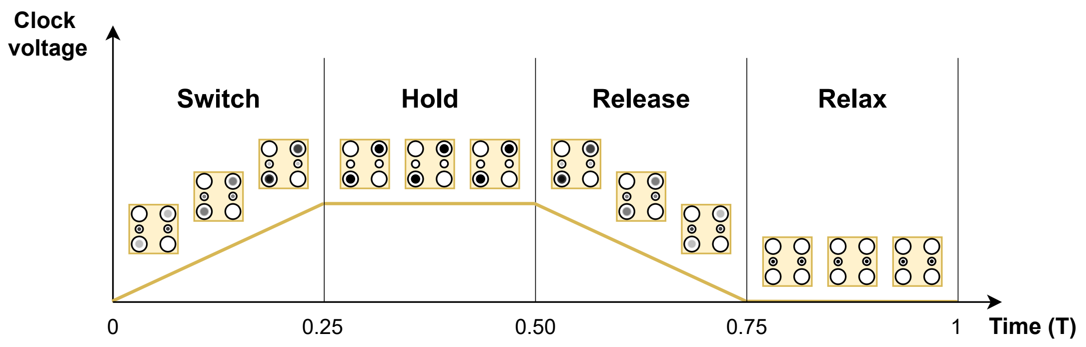


Figure 1.10. Four-phase clocking scheme. Adiabatic switching ensures smooth transition and propagates information among clock zones.

exploring different materials and their properties to encode binary information, developing suitable quantum dots, and designing alternative techniques to meet the unique demands of the paradigm. QCA cells can be manufactured using various materials, including metals, semiconductors, magnets, and molecules. The selection of material and implementation approach depends on specific requirements, such as operating temperature, speed, power consumption, and scalability.

Semiconductor QCA

Semiconductor QCA operates on the same fundamental principles as traditional QCA. It involves manipulating the state of electrons in quantum dots to transfer information and perform logical operations [42]. The QCA system consists of an array of semiconductor islands arranged in a regular lattice pattern, with each island having two stable polarization states, depending on the net charge and the direction of the electric field [56].

Neighbors influence the behavior of adjacent cells. When two semiconductor islands are brought close to each other, the movement of electrons through the tunneling barrier induces a polarization change in the neighboring cells [42].

Semiconductor technology can leverage decades of expertise in materials and fabrication techniques developed through advanced silicon fabrication to enable the realization of semiconductor QCA [42]. One of the advantages of using conventional lithography approaches is the possibility to fabricate accurate structures for essential reliable QCA cells [56]. The standard manufacturing techniques are modified to meet the specific requirements of QCA technology [56, 41, 42]. The main steps include:

- **Substrate preparation**: A smooth and uniform substrate is prepared, commonly using materials such as gallium arsenide (GaAs) or silicon (Si).
- **Island formation**: Nano-sized islands are formed on the substrate using semiconductor processes such as electron beam lithography, ion beam milling, or self-assembly.

- **Electrode deposition**: Techniques like sputtering or evaporation deposit platinum (Pt) or gold (Au) electrodes on the substrate. These electrodes provide a means to apply an electric field to the islands, allowing for manipulation of their quantum state.
- **Tunneling barriers**: They let electrons to tunnel between adjacent islands, allowing their charge state manipulation. They are realized by depositing insulating materials such as aluminum oxide (Al_2O_3) or silicon dioxide (SiO_2).
- **Wiring and interconnections**: Sputtering or evaporation techniques are used to deposit interconnections, link the islands, and form logic circuits.
- **Device testing and characterization**: The devices are tested and characterized to verify their functionality.

Metallic QCA

The Metal-dot or Metallic QCA was the first practical implementation used to demonstrate the feasibility of the QCA paradigm. According to the technology, the islands are typically made of metal, such as aluminum (Al), and the tunneling barriers are made of insulating materials, such as silicon dioxide (SiO_2) [42, 46]. The fabrication and operation principles are similar to those of the semiconductor counterpart, where islands' polarization encodes binary information.

Figure 1.11 depicts a metallic QCA cell, which can exist in two different polarization states, allowing for the implementation of logical operations. Capacitors are introduced to prevent unwanted charge movements between consecutive islands [42]. Metal QCA provides several advantages, such as a potential for high-speed operation, low power consumption, and scalability [47]. Metallic islands can also simplify the fabrication process compared to other QCAs, as standard lithographic techniques can fabricate them [47].

Despite its pioneering importance, metal QCA has some limitations, such as low polarization stability compared to its semiconductor counterpart, which can limit the size of the circuits [42, 2].

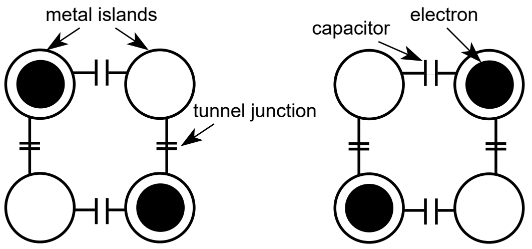


Figure 1.11. Two logical encoding states of a metal-dot QCA cell.

Challenges of semiconductor and metal QCA In recent years, metal and semiconductor QCA research has been hindered by fabrication and operating challenges at room temperature. Since QCA cells depend on the quantum behavior of electrons in a material, thermal noise at room temperature can disrupt their operation. Consequently, these cells typically work at extremely low temperatures, around $70mK$ [47], requiring expensive and complex cooling systems. Furthermore, the maximum operating frequency of cells is usually in the MHz range, which is significantly lower than magnetic and molecular QCA implementations.

Another significant challenge is fabricating high-quality quantum dots with consistent size and shape. Despite efforts to control geometry through custom growth and etching techniques, achieving uniformity in size and spacing can be challenging, and any unwanted variations can lead to misbehavior of the final systems. Additionally, the properties of the material used to manufacture them heavily influence the performance of the cells.

Magnetic QCA

Magnetic Quantum-dot Cellular Automata (MQCA) is a QCA implementation that leverages the properties of nanosize magnets to store and process information [23, 22]. Ferromagnetism is the main principle underlying MQCA, whereby certain materials such as iron (Fe), nickel (Ni), and cobalt (Co) exhibit a persistent magnetic field, even if there is no external field [13]. The phenomenon arises due to the *exchange interaction* between the magnetic moments of individual atoms in ferromagnetic materials [13].

The *exchange interaction* causes electrons with the same spin orientation to occupy adjacent atomic sites, resulting in the alignment of the magnetic moments of neighboring atoms [23]. Therefore, the magnetic moments of the entire material become aligned, leading to macroscopic magnetization. Moreover, an external magnetic field further aligns the magnetic moments of the material, which increases the overall magnetic field that persists even after the external field is released [37].

MQCA leverages the persistent magnetization of ferromagnetic materials to store and manipulate binary information [15, 37]. An external magnetic field is employed to polarize cells in either the up or down direction to encode ‘0’ or ‘1’, respectively [29]. If no external field is present, MQCA cells can be oriented horizontally to create a null state. In this state, the magnetic moments of the cell are perpendicular to the direction of magnetization, resulting in a metastable intermediate state that does not contribute to the global magnetic field [29, 26]. Figure 1.12 illustrates the three configurations of a magnetic QCA cell.

Furthermore, the inherent non-volatility of magnetic materials enables the retention of information without constant refreshing, thereby reducing energy consumption [23].

Magnetic QCA cells typically range in size from 50 to 100 nm and are arranged in a regular grid pattern [15]. These cells interact with their neighbors through magnetic field coupling [29], which depends on the alignment of their magnetic moments. If the magnetic moments are aligned, the interaction is more vigorous, whereas if not,

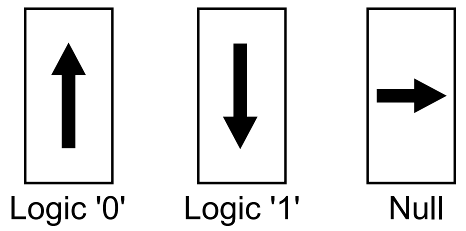


Figure 1.12. Three states of a magnetic QCA cell. Up state encoding logic ‘0’, down state encoding logic ‘1’, and null state.

it is weaker [37, 29]. MQCA exploits this field interaction to implement information propagation and logic operations in [29, 48].

In the case of an MQCA horizontal wire, transmission occurs in two steps. First, a global external magnetic field aligns all nanomagnets along their magnetically hard axes. Then, the external field is removed, and the input value is applied to the driver cell. As a result, the dipole field alignment between neighboring cells induces an antiparallel magnetization state, thereby propagating the signal [48].

Regarding logical operations, the majority voter is the fundamental logic gate in MQCA, functioning similarly to the standard QCA paradigm [15]. Figure 1.13 shows a wire and a majority voter implemented according to MQCA.

In magnetic QCA, as with standard QCA, the reliability of signal propagation can decrease as the number of cells in a circuit increases. A commonly used solution involves dividing the circuit into more compact zones, each driven by suitable clock circuitry. This partitioning limits signal propagation to a smaller number of cells, facilitating more reliable cascade propagation.

Compared to metal and semiconductor QCA implementations, magnetic QCA offers several advantages. It features more feasible fabrication processes, operates at room temperature, and consumes less power [22, 26]. Additionally, the persistent ferromagnetic property is leveraged to create systems that combine computation and memory without the need for continuous power [26]. However, stability issues may arise if the nanomagnets have dimensions below 20 nm [29]. Moreover, the maximum working frequency is not as high as in the molecular counterpart [15].

Molecular QCA

The molecular QCA approach is the most promising, along with the magnetic implementation. It employs molecules as charge storage and creates quantum-dot structures using the redox active centers within the molecule. These regions enable the exchange of charges while preserving the nature of the molecule. Due to their versatility and efficiency, molecular QCA cells have significant potential for creating powerful computing devices. The following chapter provides further details on the mQCA paradigm.

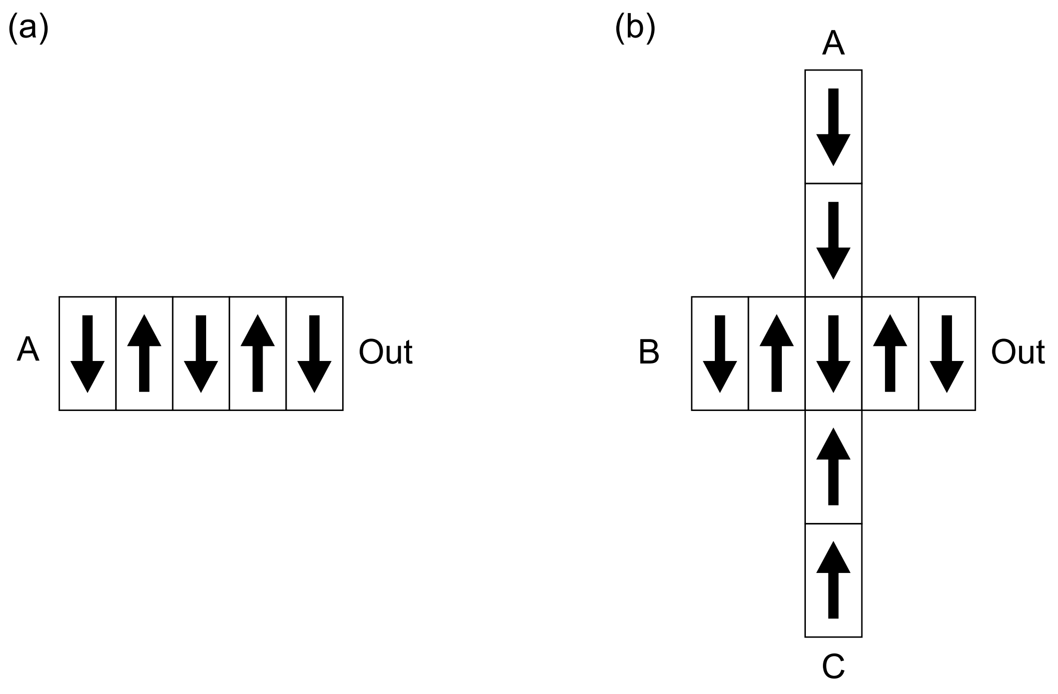


Figure 1.13. Examples of MQCA devices: (a) Magnetic QCA wire. (b) Magnetic QCA majority voter.

Chapter 2

Molecular Quantum-dot Cellular Automata

According to [36], the molecular implementation of QCA is one of the best options available in the Beyond CMOS scenario. This approach, also known as molecular field Coupled nanocomputing (mFCN), takes advantage of local field interactions among nanoscale molecules to enable propagation and computation without charge transportation.

In mFCN, redox-active centers are used as quantum dots to release or accept an electron following oxidation or reduction, respectively. These reduction-oxidation centers are sites within a molecule capable of exchanging electrons with the surroundings and modifying the charge distribution of the molecule, resulting in a net positive or negative charge. The charge positions enable the encoding of binary information while preserving the molecule integrity [36].

Molecular QCA relies on at least two redox centers connected by a linking element to promote charge tunneling and enable the realization of a half mQCA cell. Figure 2.1 illustrates the two binary encoding states of a half mQCA implemented with a two-dot molecule.

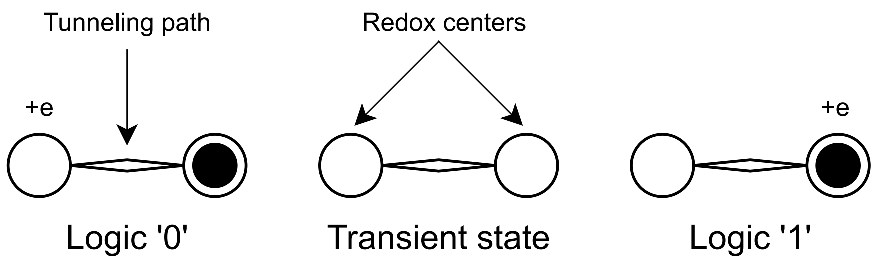


Figure 2.1. Encoding states of a 2-dots oxidized molecule: logic ‘0’, transient state, and logic ‘1’.

Nevertheless, to prevent metastability issues and ensure reliable information transmission, a third non-encoding dot is added to enable the null state, as discussed in Section 1.7. Therefore, a complete mQCA cell typically consists of two juxtaposed molecules with three dots each, resulting in six redox active sites. When two molecules are placed at a proper distance from each other, their Coulombic electrostatic coupling leads to two distinct configurations that minimize the system energy. Figure 2.2 shows the two binary encoding states and the null configuration of a complete mQCA cell.

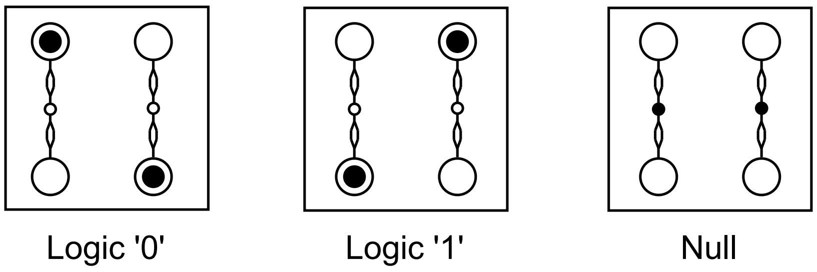


Figure 2.2. Three states of a molecular QCA cell with three dots.

Multiple mQCA cells can be arranged together to create complex molecular systems, where each molecule's charge distribution is influenced by its neighbors through electrostatic interactions. The four-phase clocking scheme, which comprises switch, hold, release, and relax phases, in combination with the null state, enables adiabatic switching.

Molecular QCA is widely recognized as a highly promising approach, offering numerous advantages from technological, computational, and operational standpoints [34]. One significant advantage is the absence of charge transport, resulting in extremely low-power dissipation. This aspect enables exceptional working frequencies in the terahertz range, high device density, and the ability to operate at ambient temperature [34, 33]. Furthermore, chemical synthesis processes can be employed to produce high-quality and uniform molecules, overcoming the inherent process variations associated with lithographic fabrication techniques [39].

2.1 Promising candidate molecules for mQCA

The development of molecular FCN has led to the proposal of a wide range of candidate molecules. However, the practical implementation of these molecules can pose significant challenges, requiring advanced processing techniques. In addition, to meet the requirements of the FCN paradigm, a suitable molecule must possess several features [61, 40] to ensure proper functioning. These include:

- **Polarization activity**: Polarizability describes a molecule's ability to generate a dipole moment when exposed to an electric field. The property is exploited for information encoding due to its potential for binary information storage.
- **Structural and electronic stability**: The molecules should exhibit structural

and electronic stability during ambient operations. Molecules with high energy barriers prevent spontaneous changes, reducing the chances of device failure in the long term.

- **Electron transport**: The molecules should possess high electron mobility and low resistance to ensure efficient information propagation within the device.

- **Binding affinity**: The molecules should allow for strong binding affinity for the substrate. Adding specific binding groups to the molecule enables a strong enough connection.

- **Synthesis feasibility**: The molecules should be synthesized in a reproducible and scalable manner, facilitating the manufacture of practical devices.

- **Testability**: The molecules should be testable to accurately characterize their properties, allowing for a deeper understanding of the molecule behavior.

- **Low-power consumption and high-speed**: The molecules should have a low energy barrier between their stable states, providing fast switching with minimal delays, meeting the unique low-power and high-speed characteristics of the mQCA paradigm.

In addition to the above strict requirements, there are also less stringent properties the paradigm can exploit for information encoding. One such property is redox activity, which allows molecules to undergo reversible changes in their oxidation state, modulating their electrical properties and enhancing their encoding capabilities. For example, the bis-ferrocene leverages its redox activity to switch between two stable states, enabling bistable information encoding. However, there are also monostable structures that can leverage only on their polarizability binary storing. Diallylbutane is one such molecule that has been demonstrated as a viable candidate for mFCN technology [7, 6].

Charge neutrality is another less strict property for modeling mFCN, but essential when manufacturing practical devices. It helps avoid unwanted vertical interactions among molecules, which can become significant in a 2D implementation, where molecules are deposited in both horizontal and vertical directions. A counterion is introduced during the synthesis process to ensure charge neutrality. [40].

Although many molecules possess the physical, electronic, and structural properties required by the paradigm, only a limited number have been successfully synthesized and tested for practical applications. Notably, diallyl-butane and decatriene are considered the benchmark for mQCA implementation due to their demonstrated feasibility [39, 33].

Furthermore, the University of Bologna synthesized the bis-ferrocene molecule specifically for mQCA applications, as documented in [68, 12]. This molecule has exhibited significant promise at the circuit level, indicating its potential as a viable compound for practical devices, as evidenced by research conducted by the Politecnico di Torino [8, 67].

2.1.1 Dyallyl-butane

One of the first molecules presented for mQCA is the diallyl-butane, which is depicted in structural and schematic forms in Figure 2.3 [8]. This molecule consists of two allyl groups that function as quantum dots and a butane connection that links them. It is often used in its cationic form to let the unpaired electron move in the structure and localize in one of the two dots. To realize a complete QCA cell, two diallyl-butane molecules are placed at a proper distance *d*.

Despite its good performaces, a real FCN implementation cannot use it due to the lack of a third dot to encode the null state. As a result, it is impossible to control it through an external clock field. Moreover, there is no binding element, making its practical realization even less feasible.

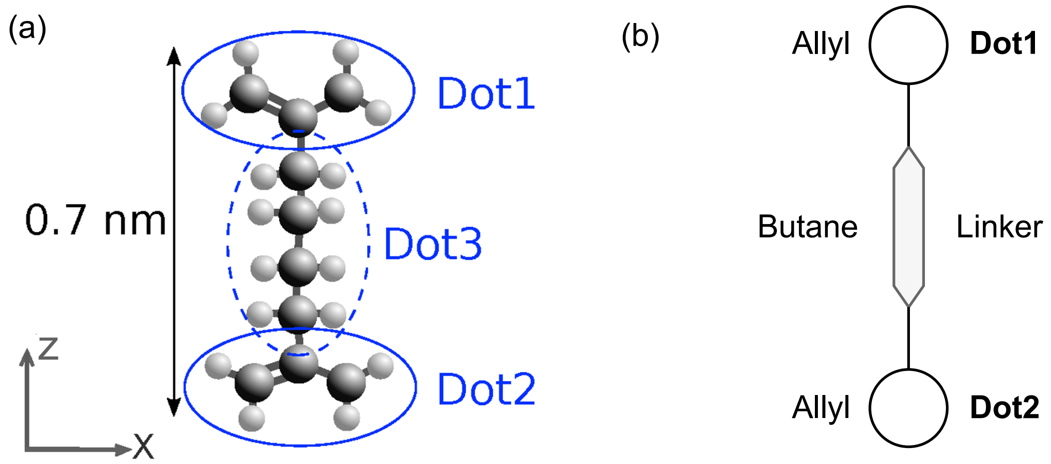


Figure 2.3. Structural and schematic representations of the dyallyl-butane molecule.

2.1.2 Decatriene

Decatriene $C_{10}H_{16}$ is a linear hydrocarbon molecule classified as an alkene due to the presence of at least one carbon-carbon double bond [39]. Its structure consists of a linear chain of ten carbon atoms with alternating double carbon-carbon bonds, creating a conjugated system of pi electrons. This configuration results in an electron cloud that contributes to the molecule's unique electronic properties and stability. These qualities make decatriene a promising candidate for various applications in chemistry and materials science [39]. Figure 2.4 shows the molecular structure and schematic representation of decatriene.

The molecule has two ethylene C_2H_4 groups at each end of the carbon chain and a third ethylene group in the center. These groups can act as quantum dots, realizing a three-polarization state. Decatriene has been proposed as a potential candidate to overcome the limitations of diallyl-butane, as it contains a third ethylene group that enables the null state, allowing for clock controlling [39]. Nonetheless, just like diallyl-butane, decatriene lacks the necessary elements to bind to a surface.

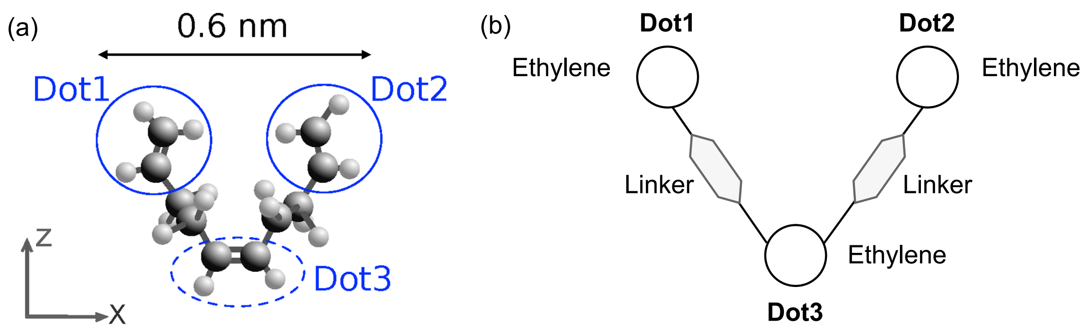


Figure 2.4. Structural and schematic representations of the decatriene molecule.

2.1.3 Bis-ferrocene carbazole

Bis-ferrocene carbazole or bis-ferrocene is a hybrid molecule that combines two ferrocene units, $Fe(C_5H_5)_2$, linked by a bridging carbazole group, $C_{12}H_9N$. The molecule has been extensively studied and synthesized by researchers, including L. Zoli in [68] and V. Arima et al. in [12], providing a comprehensive understanding of its properties and potential applications in molecular computing.

L. Zoli's work [68] focuses more on the theoretical aspects, conducting theoretical density functional theory (DFT) calculations to study the behavior and electronic properties of the molecule in circuits. On the other hand, V. Arima et al. in [12] present thiolated-carbazole linked bis-ferrocenes, which are similar to bis-ferrocene carbazole but with thiol groups added for surface binding. The work is more focused on the synthesis and characterization of these molecules and their self-assembled monolayers (SAM) on gold surfaces.

Figure 2.5 displays the structure and schematic representation of the bis-ferrocene molecule [8]. Each ferrocene unit contains two cyclopentadienyl rings bound to an iron atom, providing a pair of delocalized electrons to use as quantum dots. The carbazole group, a heterocyclic compound containing a nitrogen atom, acts as a third dot, enabling the implementation of the null state [68]. The carbazole group also offers a means of functionalizing the molecule for surface binding, which is critical for practical implementations [68].

The molecule is typically oxidized due to better performance by removing an electron through electrochemistry [8]. A counterion is combined with the molecule to maintain charge neutrality [12].

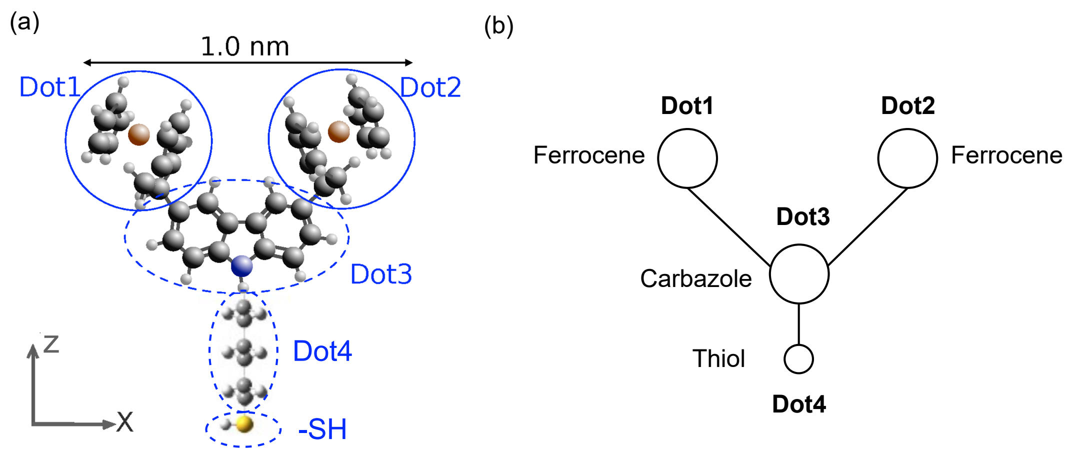


Figure 2.5. Structural and schematic representations of the bis-ferrocene carbazole molecule.

2.2 Methodologies for analyzing mQCA systems

Designing a functional molecular QCA device is a complex process that requires a thorough analysis of every aspect of the mQCA paradigm. A comprehensive methodology is demanded to ensure a rigorous and universally applicable flow to all candidate molecules.

One possible approach is to use quantum mechanical chemistry analysis through ab initio simulations to rigorously characterize each molecule and its interactions with other molecules in the system. Nevertheless, this method is computationally intensive, making it suitable for simple circuits and single molecules only.

Therefore, it is crucial to explore alternative approaches for studying the behavior of molecules in mQCA systems that can handle complex circuits and multiple interactions while minimizing computational resources.

2.2.1 Ab initio methods

Ab initio calculations are a class of computational methods that rely on the principles of quantum mechanics to model the structure and properties of various materials, including electronic energy and charge distribution, without using any experimental data or empirical parameters. Some examples are:

- **Hartree-Fock (HF)**: It is a quantum mechanical method that relies on solving the Schrödinger equation for the molecular orbitals of electronic systems.

- **Density Functional Theory (DFT)**: It computes the electronic structure of devices considering the distribution of electrons in space, unlike traditional quantum mechanical methods, which solve wavefunctions of single charges.

- **Monte Carlo (MC)**: It provides a statistical description of the system's electronic transport properties, such as conductivity, mobility, and carrier density, by randomly sampling the positions and speed of electrons.

- **Ab initio Molecular Dynamics (AIMD)**: It is based on Molecular Dynamics (MD) which describes how electronic properties of nuclei change over time by solving the classical equations of motion. A starting configuration is chosen and initial speeds are assigned to each element. The simulation proceeds by iteratively advancing the positions and velocities of the particles using numerical integration techniques to calculate the forces acting on each particle at each time step. Similarly, AIMD achieves the same goal but on the base of quantum mechanical calculation, which makes it more accurate, but also very computationally demanding.

- **Real-Time Time-Dependent Density Functional Theory (RT-TDDFT)**: If external forces acting on a system and initial state are known, the theory can predict how the system's electronic wavefunction will evolve. Chemistry and physics employ it to characterize time-dependent properties such as electron movement.

- **Ehrenfest Dynamics (ED)**: Originally introduced by Paul Ehrenfest to investigate the behavior of molecules in a gas, ED is a simulation technique that models a group of particles as individual point elements interacting with each other, tracking their positions and momenta over time. This information is used to make predictions about how the group as a whole will behave.

Ab initio calculations have been extensively utilized in researching QCA interactions among individual and multiple molecules [33, 49, 35]. One of their main advantages is their high accuracy, which allows for the precise characterization of molecular systems. However, their computational expense and time-consuming nature limit their practicality for modeling information propagation in complex systems.

2.2.2 QCADesigner

To address ab initio computations drawbacks, researchers are actively searching for more efficient methods. In the context of quantum cellular automata, two-state approximation (TSA) is a commonly used quantum mechanics model that describes the electronic states of cells using ground and exited states [38]. The approach forms the basis of QCADesigner, a highly effective simulation tool for the general QCA paradigm [65].

However, it has limited applicability in describing the behavior of molecular FCN circuits due to the use of an overly simplified model and inability to include process variations [11]. For these reasons, the authors in [9, 50] propose a new methodology to analyze molecular devices.

2.2.3 MosQuiTo

Molecular Simulator Quantum-dot cellular automata Torino (MosQuiTo) [9, 50] is a promising method for analyzing molecular devices. Although initially presented for the bis-ferrocene, any candidate mFCN molecule can employ it. The procedure is deeply connected with the nano-level physical parameters of the molecule under test and consists of three main steps:

- **Single molecule characterization**: Ab initio simulations are employed to retrieve the molecule electronic properties.
- **Post-processing**: The obtained results are post-processed to extract relevant figures of merit.
- **Analysis at the system-level**: It is performed to consider every intermolecular interaction in the circuit.

The MosQuiTo methodology translates every molecule interaction into a set of electrostatic equations that describe the actual circuit behavior at the molecular level. This approach overcomes the need to rely on very demanding ab initio simulations while preserving the connection with the underlying physics. The three-step analysis yields results employed as a model by the Self-Consistent ElectRostatic Potential Algorithm (SCERPA), a standalone procedure that simulates complex molecular FCN logic devices. Chapter 3 provides more details about SCERPA.

2.2.4 Characterization of the single molecule

The starting point of the procedure is to obtain the electronic structure and physical properties of the molecular system through quantum mechanical analysis. In case of the bis-ferrocene, density functional theory (DFT) is leveraged using Gaussian 09, a widely used software package for computational chemistry [11]. Next, the single molecule is characterized in three conditions: under the influence of another molecule, a switching field, and a clock signal.

Effect of another molecule

The characterization process of a molecular system involves analyzing the simplest form of a mFCN device to pave the way for the analysis of the full range of interactions present in more complex molecular systems. Specifically, a single molecule is analyzed in response to a driver molecule positioned at a distance d. The driver can be modeled as a neutral element with two-point charges filling the two encoding dots or as an oxidized element with a single positive charge.

Effect of switching field

The switching field is an electric field applied parallel to the working dots to induce charge localization in one of the two logical dots, allowing for the desired molecule polarization to be defined. Conventionally, if positive, the charge is confined in Dot2, and if negative, the charge localizes in Dot1. This mechanism enables the writing of logical state ‘0’ or ‘1’ to the single molecule. Figure 2.6 depicts a graphical representation of the switching field effects.

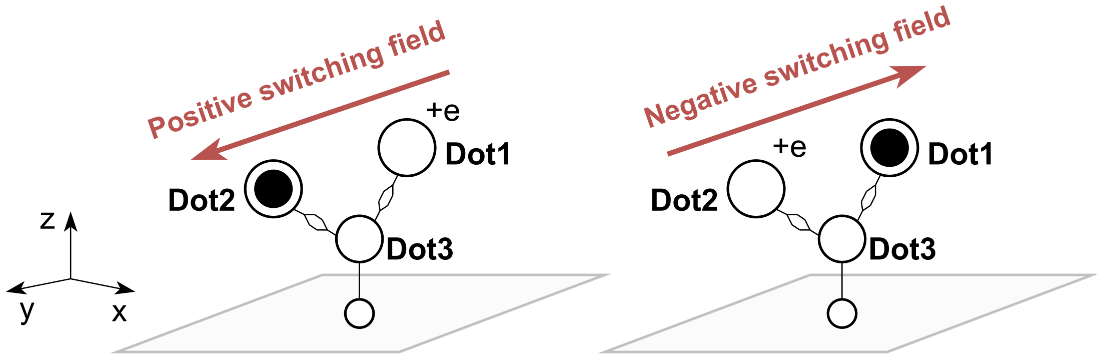


Figure 2.6. Switching field effects on a bis-ferrocene molecule. A positive field restrains the charge in Dot2, whereas a negative field confines it in Dot1.

Effect of clock field

The clock field is an electric field required to satisfy the adiabatic switching condition. It can enhance or impede the interaction between neighboring molecules, thereby providing greater control over information processing. In the context of the bis-ferrocene and other three-dot molecules, the field is applied along the vertical axis of the molecule to achieve the desired charge distribution and enable proper logic switching. Conventionally, if positive, the charge can localize in one of the encoding dots, whereas, when negative, the molecule remains in the null state. Figure 2.7 depicts a representation of the switching field effects.

2.2.5 Post-processing of results

The results of the ab initio simulation are further processed to generate more usable figures of merit, which allow for a more accurate interpretation and understanding of the physical and chemical properties of the system [11].

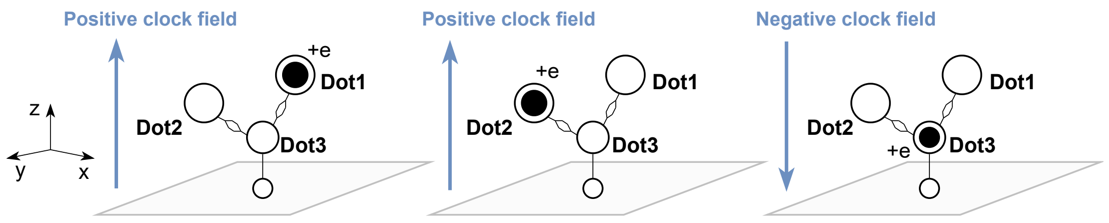


Figure 2.7. Clock field effects on a bis-ferrocene molecule. A positive field facilitates the molecule's transition to an encoding state, whereas a negative field causes the charge to be pushed into the null state.

Atomic charge

Atomic charges are computed via ab initio simulations using the Merz-Singh-Kollman (MK) approximation scheme [58]. They are an alternative to the traditional HOMO-based analysis, providing insight into the molecule's behavior under different operating conditions, such as ground state and biasing.

Aggregated Charge (AC)

The total charge of each dot in a molecule is represented by the aggregated charge. This charge is calculated by summing all the atomic charges of the redox center to which the dot belongs. The AC model considers the molecule as a collection of aggregated point charges. For instance, in the case of bis-ferrocene, the molecule is represented by three ACs: Q1, Q2, and Q3, which correspond to Dot1, Dot2, and Dot3. The charge distributions of the bis-ferrocene molecule and its corresponding aggregated charges are displayed in Figure 2.8.

Electric field

The electric field generated by the charge distribution of a molecule can be computed using electrostatic formulas. Specifically, the electric field generated by a single point charge $+q_1$ located at position $\mathbf{r}_1$ can be evaluated at any point in space $\mathbf{r}$ by introducing a test charge $+q_t$. The distance between the two charges is:

$$\mathbf{r}_1 = r_1 \cdot \hat{\mathbf{r}} \tag{2.1}$$

$$r_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} \tag{2.2}$$

where $\hat{\mathbf{r}}$ is the unit vector pointing from the charge $+q_1$ to the test charge $+q_t$ and r_1 is the distance magnitude. Gauss law is used to obtain the Coulomb force applied on the test charge $+q_t$:

$$F_1(\mathbf{r}) = \frac{1}{4\pi\varepsilon_0} \frac{q_1 \cdot q_t}{r_1^2} \cdot \hat{\mathbf{r}} \tag{2.3}$$

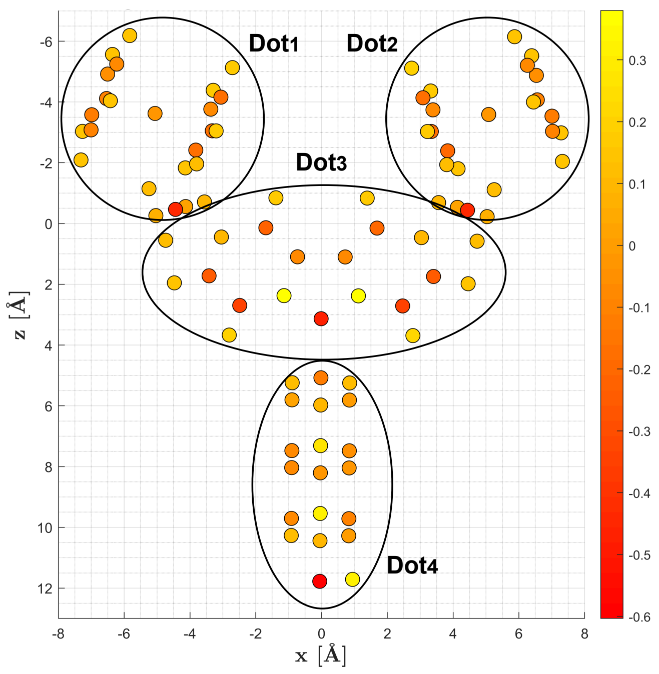


Figure 2.8. Charge distribution of the bis-ferrocene molecule alongside its aggregated charges.

where ε_0 is the vacuum permittivity.

The associated electric field $\mathbf{E}_1$ is computed as:

$$E_1(\mathbf{r}) = \frac{F_1(\mathbf{r})}{q_t} = \frac{1}{4\pi\varepsilon_0}\frac{q_1}{r_1^2} \cdot \hat{\mathbf{r}} \tag{2.4}$$

The model can be extended to a system of N point charges by summing over all the individual contributions. The total Coulomb force on the test charge $+q_t$ becomes:

$$F(\mathbf{r}) = \sum_{i=1}^{N} \frac{1}{4\pi\varepsilon_0}\frac{q_1 \cdot q_t}{r_1^2} \cdot \hat{\mathbf{r}} \tag{2.5}$$

with an associated electric field:

$$E(\mathbf{r}) = \frac{F(\mathbf{r})}{q_t} = \sum_{i=1}^{N} \frac{1}{4\pi\varepsilon_0}\frac{q_1}{r_1^2} \cdot \hat{\mathbf{r}} \tag{2.6}$$

The electric field has three components along the x, y, and z directions:

$$E_x(x, y, z) = \sum_{i=1}^{N} E_{x,i}(x, y, z) \tag{2.7}$$

$$E_y(x,y,z) = \sum_{i=1}^{N} E_{y,i}(x,y,z) \tag{2.8}$$

$$E_z(x,y,z) = \sum_{i=1}^{N} E_{z,i}(x,y,z) \tag{2.9}$$

The voltage associated to an electric field can be computed as:

$$V(\mathbf{r}) = -\int_{\gamma} \mathbf{E} \cdot d\mathbf{l} = -\int_{\gamma_x} E_x \frac{\partial V}{\partial x} d\mathbf{x} - \int_{\gamma_y} E_y \frac{\partial V}{\partial y} d\mathbf{y} - \int_{\gamma_z} E_z \frac{\partial V}{\partial z} d\mathbf{z} \tag{2.10}$$

where $\mathbf{E}$ is the electric field, $d\mathbf{l}$ is the differential length element, and γ is the path connecting the two charges. The integrals represent the line integrals of the electric field along the three coordinate axes.

Correspondingly, the electric field is calculated from the voltage as:

$$E(\mathbf{r}) = -\nabla V(\mathbf{r}) = -\frac{\partial V}{\partial x}\hat{\mathbf{x}} - \frac{\partial V}{\partial y}\hat{\mathbf{y}} - \frac{\partial V}{\partial z}\hat{\mathbf{z}} \tag{2.11}$$

where ∇ is the gradient operator and $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ are the unit vectors in the x, y, and z directions, respectively and the negative sign means that the electric field is directed from higher potential to lower potential.

Equivalent voltage at the receiver

The electrostatic model mentioned earlier is employed to analyze the system of charges illustrated in Figure 2.9. In this system, a collection of N_{AC} charges denoted by $\{Q_1^D, \ldots, Q_{N_{AC}}^D\}$ and positioned at $\{\mathbf{r}_1^D, \ldots, \mathbf{r}_{N_{AC}}^D\}$, represents a driver D. In turn, the driver generates a voltage V_D at any point $\mathbf{r}$ in space, which can be obtained as:

$$V_D(\mathbf{r}) = \frac{1}{4\pi\varepsilon_0} \sum_{\beta=1}^{N_{AC}} \frac{Q_i^D}{d\left(\mathbf{r}_\beta^D, \mathbf{r}\right)} \tag{2.12}$$

where $d\left(\mathbf{r}_\beta^D, \mathbf{r}\right)$ represents the Euclidean distance between each of the aggregated charges of the driver and the point $\mathbf{r}$.

When a molecule MUT is placed at a distance d from the driver, with its active dots Dot1 and Dot2 positioned at $\mathbf{r}_1^j$ and $\mathbf{r}_2^j$, respectively, the electric field generated by the driver on the MUT is determined as:

$$V_{in} = \int_{\gamma} \mathbf{E}_D \cdot d\mathbf{l} \; = V_D(\mathbf{r}_1^j) - V_D(\mathbf{r}_2^j) \tag{2.13}$$

where $\mathbf{E}_D$ represents the electric field generated by the driver charge distribution, and γ is the path connecting the dots of the MUT.

The voltage generated by the MUT in response to the driver voltage is named equivalent voltage at the receiver (V_{out}). To measure it, a receiver molecule is employed, resulting in a complete mQCA cell.

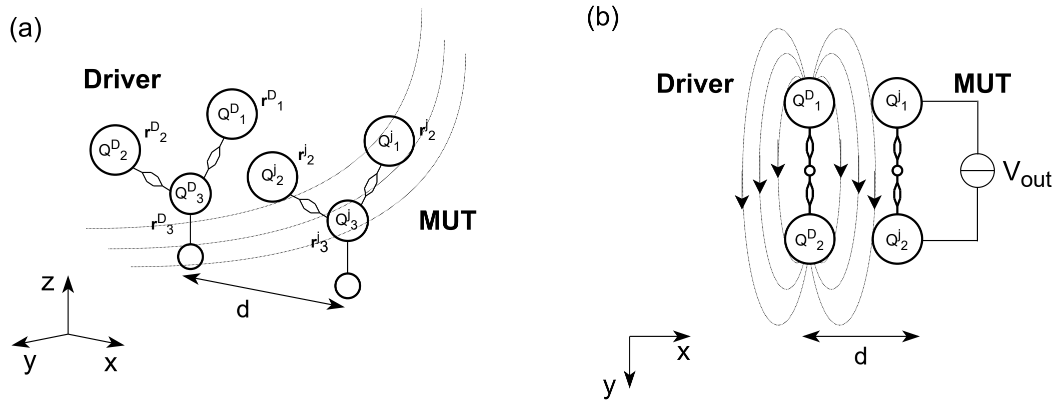


Figure 2.9. Interactions in a complete mQCA cell. (a) 3D view and (b) top view.

$V_{in} - V_{out}$ Transcharacteristic (VVT)

The relationship between the input driver voltage and the output generated voltage is named $V_{in} - V_{out}$ Transcharacteristic. This figure of merit can be studied in the presence of external clock fields and by varying intermolecular distances. Figure 2.10 depicts the variations in the $V_{in} - V_{out}$ transcharacteristic of a bis-ferrocene complete mQCA cell as a function of intermolecular distance under three external clock fields.

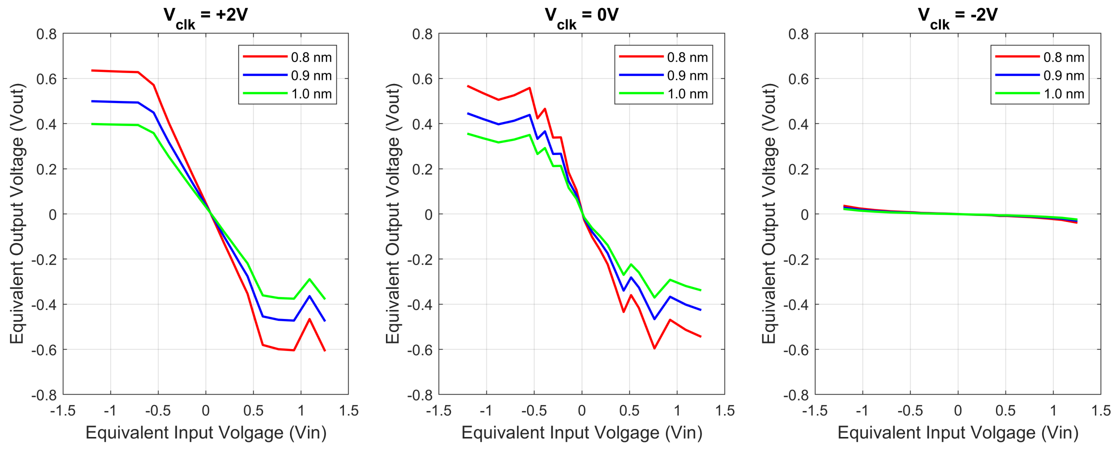


Figure 2.10. $V_{in} - V_{out}$ Transcharacteristics of a complete bis-ferrocene cell subjected to three clock fields.

V_{in}–Aggregated Charge Transcharacteristic (VACT)

The quantity is derived from ab initio simulations that quantitatively describes the relationship between the input voltage V_{in} and the aggregated charges in the presence of clock fields. The input voltage is obtained from Equation (2.13) integrated over the path that links the logical dots of the molecule. Figure 2.11 shows the variations of the V_{in}–AC transcharacteristic of a bis-ferrocene molecule as a function of intermolecular distance under three external clock fields.

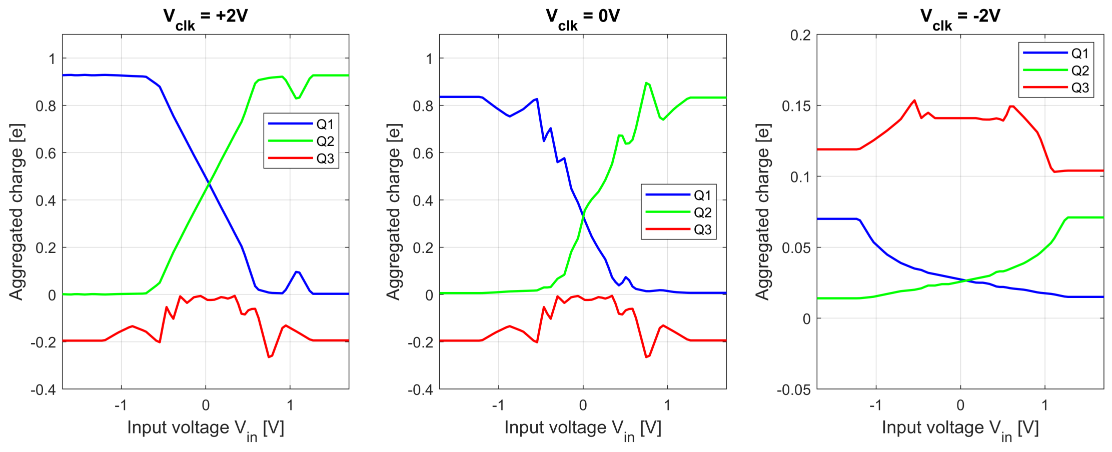


Figure 2.11. V_{in}–AC transcharacteristics of a bis-ferrocene molecule subjected to three clock fields.

2.2.6 Analysis at the system-level

The figures of merit derived earlier can be employed to shift the focus from a molecular level to a broader system perspective. A recent study [51] has successfully demonstrated the feasibility of leveraging a write-in read-out system to model a molecular wire and analyze information propagation through an iterative procedure. The method can be generalized and applied to any circuit based on mFCN technology, efficiently analyzing complex systems.

The procedure can be described using the system shown in Figure 2.12 as a reference. The generic *Molecule j* is modelled as a set of N_{AC} charges $\{Q_1^j, \ldots, Q_{N_{AC}}^j\}$ positioned at $\{\mathbf{r}_1^j, \ldots, \mathbf{r}_{N_{AC}}^j\}$. As per Equation (2.12), *Molecule j* generates a voltage in a generic position $\mathbf{r}$ in space, given by:

$$V_j(\mathbf{r}) = \frac{1}{4\pi\varepsilon_0} \sum_{\beta=1}^{N_{AC}} \frac{Q_i^j}{d\left(\mathbf{r}_\beta^D, \mathbf{r}\right)}. \tag{2.14}$$

Simultaneously, all molecules in the system are influenced by *Molecule j*, both in the backward driver and forward output direction. If another generic *Molecule i*, modelled as $\{Q_1^i, \ldots, Q_{N_{AC}}^i\}$ in positions $\{\mathbf{r}_1^i, \ldots, \mathbf{r}_{N_{AC}}^i\}$, the interaction between *Molecule j* and *Molecule i* results in the generation of a voltage by *Molecule j*, computed as:

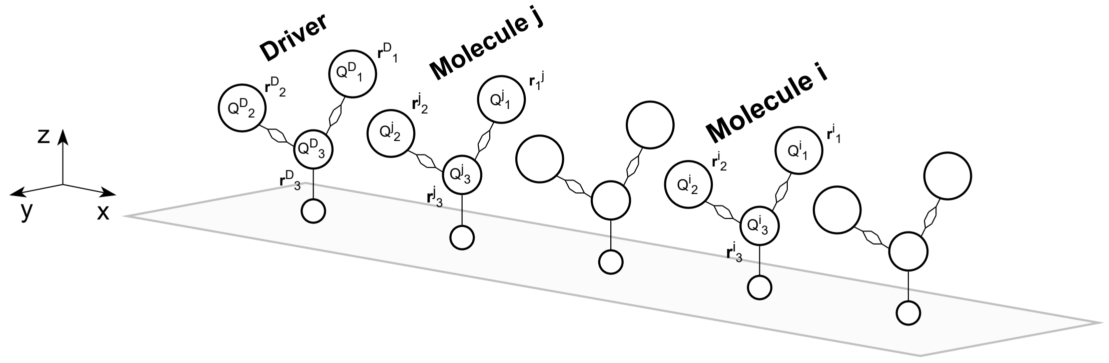


Figure 2.12. MosQuiTo schematic model for system-level interactions.

$$V_{j,i} = V_j\left(\mathbf{r}_1^i\right) - V_j\left(\mathbf{r}_2^i\right) = \frac{1}{4\pi\varepsilon_0}\sum_{\beta=1}^{N_{AC}}\left[Q_\beta^j\left(\frac{1}{d\left(\mathbf{r}_1^i,\mathbf{r}_\beta^j\right)} - \frac{1}{d\left(\mathbf{r}_2^i,\mathbf{r}_\beta^j\right)}\right)\right] \tag{2.15}$$

After *Molecule i* receives the influence of *Molecule j*, it acts as a driver that generates a backward response to every other element in the system.

In a broader sense, the input voltage $V_{in,i}$ of each *Molecule i* is determined by the combined effects of the voltage generated by the driver $V_{D,i}$ and that of other molecules $V_{j,i}$. In addition, by using the V_{in}–AC transcharacteristic, the effect of the external field is considered as:

$$V_{in,i} = V_{D,i} + \sum_{\substack{j=1\\ j\neq i}}^{N_{AC}} V_{j,i}\left(V_{in,j}, E_{clk,j}\right) \tag{2.16}$$

Furthermore, the V_{in}–AC transcharacteristic can be employed to retrieve the aggregated charges of each molecule in the system, which can assess the propagation and computation of the given circuit. Figure 2.13 illustrates the propagation of signals in a molecular wire, demonstrating how the free charges rearrange themselves in each molecule dot.

Building upon the unidirectional molecular propagation enabled by the MosQuiTo method, the authors in [11, 10] propose a more comprehensive algorithm named SCERPA for studying molecular propagation. The procedure is based on an iterative self-consistent loop that considers all molecules interactions by modeling them using their aggregated charges [52]. The algorithm enables the simulation of complex molecular FCN logic devices without expensive ab initio computations, still providing insights into the underlying physics.

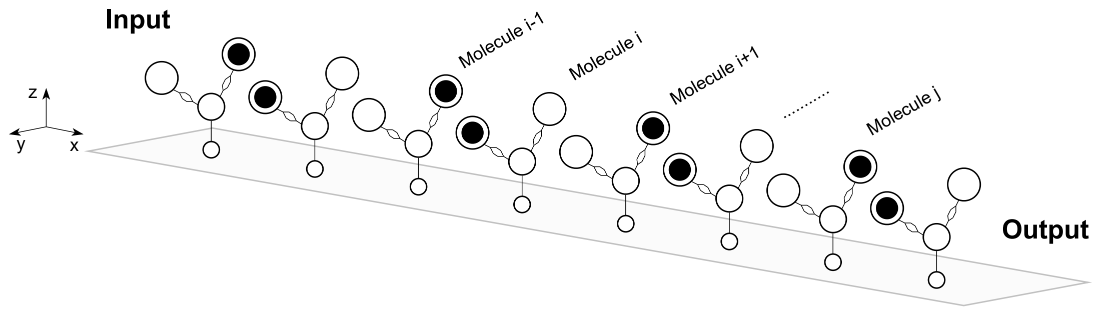


Figure 2.13. Schematic representation of the molecular wire.

2.3 Molecular FCN devices

Molecular FCN systems can be categorized into two types based on functionality: those that solely propagate information and those that realize logical operations. Circuits are divided into an appropriate number of clock zones, depending on their complexity, to ensure proper functionality and promote adiabatic switching. The following sections explore a range of increasingly complex molecular circuits.

2.3.1 Single molecular cell

The simplest molecular FCN circuit consists of a single molecular QCA cell conceived with two adjacent molecules, as depicted in Figure 2.14. This system is analogous to the one used by the Mosquito methodology to characterize the behavior of a cell under the influence of a driver placed at a distance of *d*. The cell is subjected to a single-phase clock that facilitates the rearrangement of charges in the molecule after the driver is imposed in one of the possible encoding states.

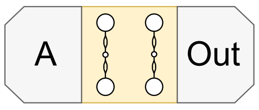


Figure 2.14. Single mFCN cell layout made of a driver, two adjacent molecules and an output.

2.3.2 Single phase wire

Multiple mFCN cells can be consecutively juxtaposed to create the simplest form of unidirectional propagation, resulting in a wire. As a reference, Figure 2.15 depicts a circuit consisting of six cells. Due to the relatively low number of cells, transmission through a wire of this size can be effectively controlled and guaranteed using a single-phase clock. Information flows from the driver toward the end of the circuit.

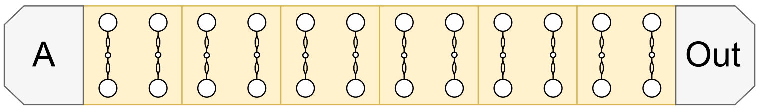


Figure 2.15. Single phase mFCN wire layout made of six cells.

2.3.3 Three phases wire

If many cells are placed consecutively, adiabatic switching may not be satisfied, leading to incorrect information propagation. As a result, long wires are partitioned into multiple clock phases. This approach is illustrated in Figure 2.16, which depicts a wire of twelve molecules divided into three clock phases. A pipeline can be created by precisely synchronizing with the external clock and modifying the driver value every time the previous information reaches the third clock zone, which occurs with a period T. In this situation, the driver is far enough to avoid any interaction with the third-phase molecules and can be changed to another value, improving the circuit bandwidth.

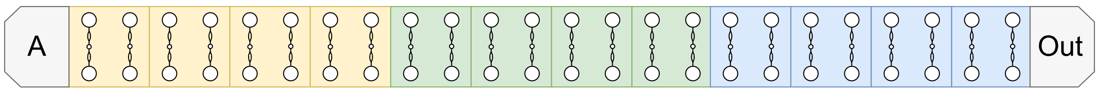


Figure 2.16. Three phases mFCN wire layout made of twelve cells.

2.3.4 Two-line three phases wire

Experimental simulations have revealed that the last molecule of each clocked wire phase exhibits weaker charge separation due to a common mFCN border effect [11, 6]. An alternative wire layout based on a two-line bus structure typically solves the issue [10]. Although it may result in crosstalk between the two adjacent lines and an uneven charge distribution along the bus, information propagation remains accurate. Figure 2.17 illustrates the alternative three phases wire layout.

2.3.5 Two-line L-connection

The L-connection is a single-input single-output propagation device. It enables a seamless 90-degree turn, allowing for easy transitions of information propagation between the horizontal and vertical directions of a 2D plane. The two-line bus structure version depicted in Figure 2.18 further enhances the device's robustness against information loss or degradation, such as border effects. The circuit consists of two branches with consecutive clock phases that create a pipeline-like structure, enabling complete information propagation in one direction before copying it to the other and reaching the output.

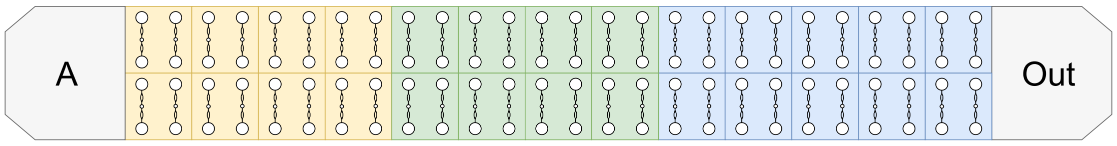


Figure 2.17. Two-line three phases mFCN wire layout.

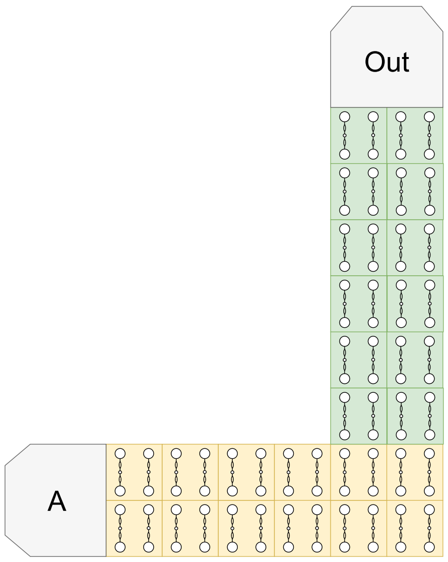


Figure 2.18. Two-line mFCN L-connection layout.

2.3.6 Two-line T-connection

The T-connection device follows the same principle as the L-connection device in separating the information propagation into two perpendicular directions. It features one input and two outputs, divided into three branches with two consecutive clock regions. The first encompasses the driver and facilitates transmission up to the copy point, while the second propagates the information toward the two outputs. Figure 2.19 shows the two-line T-connection structure.

2.3.7 The inverter

The inverter logic gate is an essential QCA device used for logic computation that produces the complement of the input value. It can be implemented in a single-line configuration, as shown in Section 1.6.1 for the standard QCA paradigm, or in a two-line bus fashion.

Notwithstanding, research has shown that the single-line structure, when implemented with the bis-ferrocene molecule, is particularly fragile and susceptible to degradation of the output cell polarization [11]. Therefore, the two-line version reported in Figure 2.20 is preferred for its enhanced encoding stability and robustness.

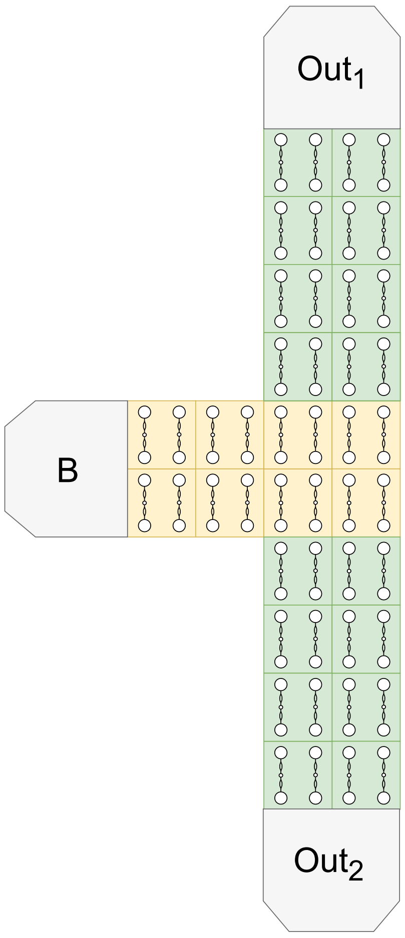


Figure 2.19. Two-line mFCN T-connection layout.

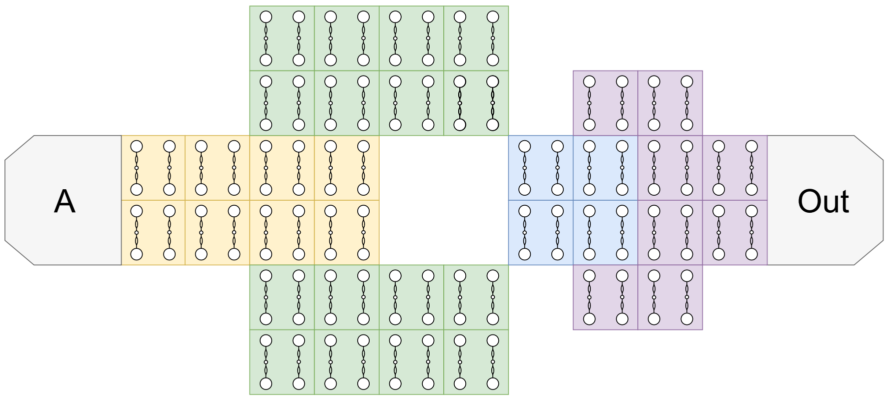


Figure 2.20. Two-line mFCN inverter gate layout.

2.3.8 The majority voter

The majority voter is an example of a logic computation device where the output corresponds to the most recurrent input signal according to the truth table presented in Table 1.2. Its molecular structure implemented in a two-line bus fashion is depicted in Figure 2.21.

The device is partitioned into three clock regions, identifying the input branches, the central region, and the output branch. The branches propagate the input signals to the central region, which determines the majority input signal and propagates it through the third region to the output.

The advantage of having a dedicated evaluation region is required to synchronize all inputs that may arrive at different times. In [67], various alternative structures of the majority voter have been proposed, some of which feature an enlarged central region that fully or partially incorporates the input branches.

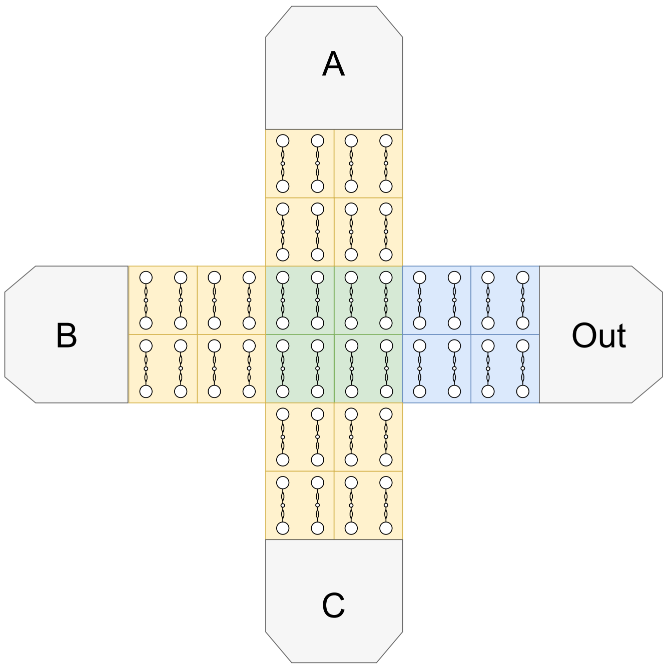


Figure 2.21. Two-line mFCN majority voter layout.

2.3.9 The exclusive OR

Majority voters and inverters can be combined to generate any Boolean function [25]. As an example, the XOR logic gate can be computed as a combination of three majority voters and two inverters:

$$A \veebar B = MV(MV(\bar{A}, B, 0), 1, MV(A, \bar{B}, 0)) = (A \wedge \bar{B}) \vee (\bar{A} \wedge B) \tag{2.17}$$

where $MV()$ refers a majority voter operation, $\bar{A}$ and $\bar{B}$ are the two inverted inputs, and $\veebar$, $\wedge$, and $\vee$ denote the XOR, AND, and OR operations, respectively. The device produces '1' when the input values differ and '0' when they are the same according to the truth table reported in Table 2.1. The digital implementation of Equation (2.7) and the two-line molecular XOR gate are shown in Figure 2.23 and Figure 2.22, respectively, with the molecular version highlighting the fundamental QCA blocks.

A	**B**	**Out**
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1. Exclusive-OR truth table.

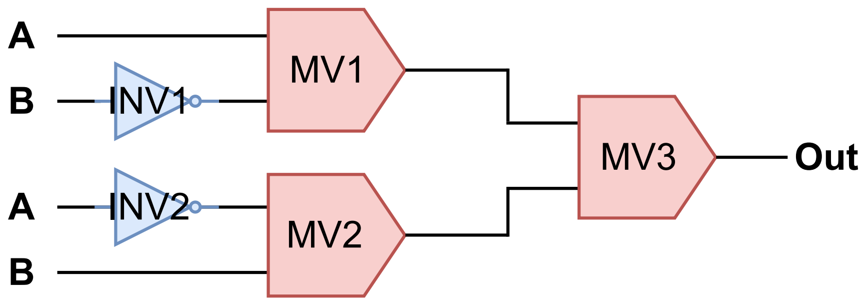


Figure 2.22. XOR gate implemented in the digital domain with a majority voter and inverter-based logic.

Each molecular block preserves its clock phase partition, ensuring information integrity throughout the three pipeline stages of the circuit. The two inverters complement the inputs synchronously, making them available to the two majority voters and implementing the AND logical gate. Finally, their outputs are combined with the third majority voter, performing the OR operation and providing the final result.

2.4 Manufacturing molecular FCN devices

The implementation of molecular FCN is a significant challenge. Currently, research in this field is limited, as evidenced by the reduced number of studies [51, 17].

One of the major obstacles is the nanometer size of the molecules involved, which typically ranges from 1 to 2 nm, making it difficult to manufacture working FCN devices using conventional lithography-based techniques. A possible solution is to exploit

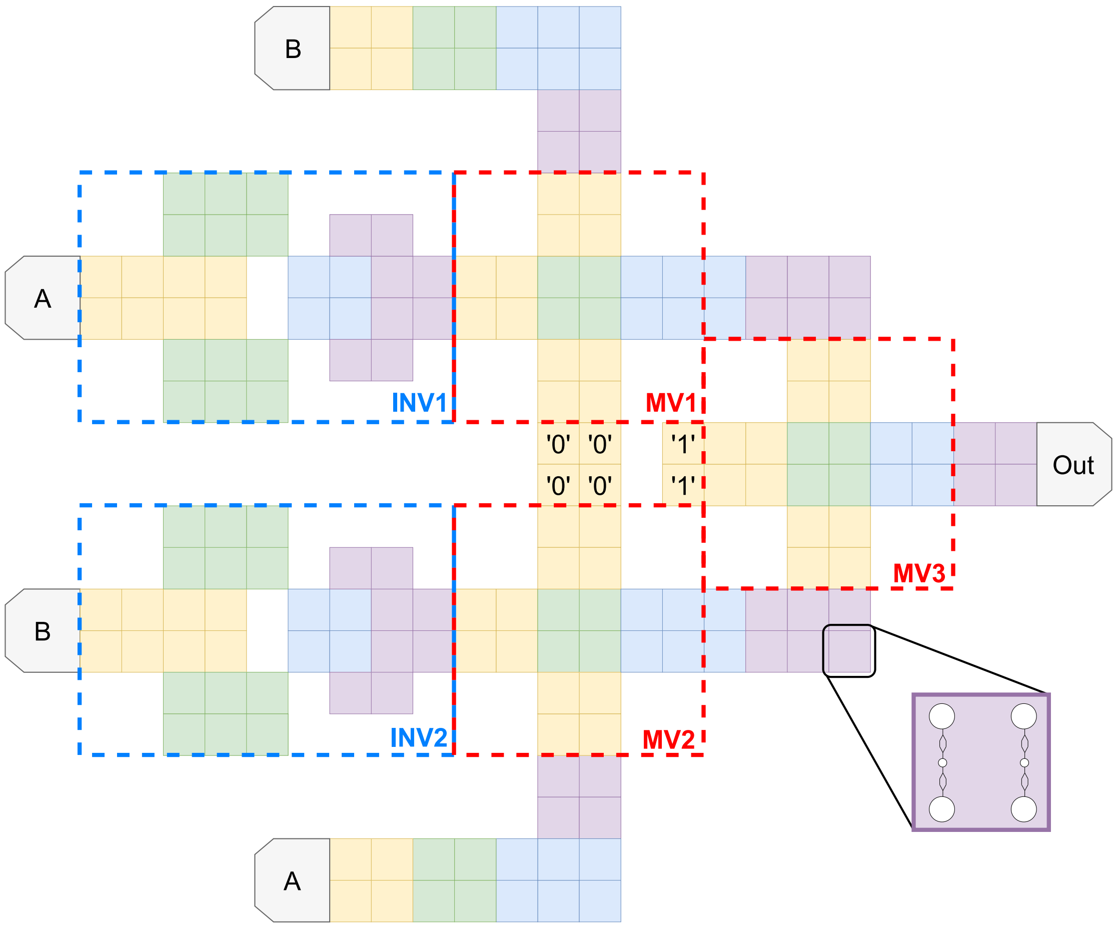


Figure 2.23. Two-line mFCN XOR gate layout. Each square corresponds to an mQCA cell, consisting of two adjacent bis-ferrocene molecules.

molecules that support Self-Assembled Monolayer (SAM). These molecules can attach to a surface in a highly organized fashion.

Generating the electric fields required for controlling the computation is another significant challenge. One possible solution involves placing molecules in a trench over a wire, with two electrodes on the edges. An electric field is generated by applying a voltage between the electrodes, acting like a capacitor [30]. Figure 2.24 shows a top view of a clocked wire. However, depending on the size of the electrodes, a certain number of molecules are inevitably affected, which can result in undesired border effects and potentially unacceptable behavior [11, 6].

Another critical challenge is finding suitable surfaces for placing molecules and electrodes. Gold surfaces are used as substrate material due to their excellent electrical conductivity, stability, and compatibility with molecular self-assembly techniques. Despite that, the precision of the surface remains limited by the lithography process, and any roughness can result in vertical misalignment, horizontal shifts, and rotations, which can affect the propagation of information.

Among the molecules discussed in Section 2.1, bis-ferrocene supports SAM. Its thiol group is employed to bind it to a gold substrate, as demonstrated in [51]. Molecules are typically transferred onto the desired surface using techniques such as microcontact printing or dip-pen nanolithography. The first involves a patterned stamp, while the latter exploits a fine-tipped pen to deposit the molecules onto the surface.

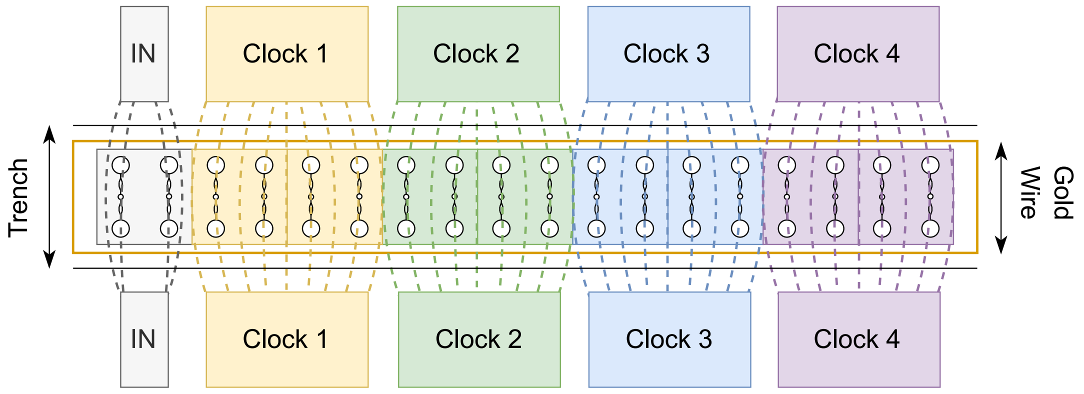


Figure 2.24. Top view representation of a practical clocked wire implementation.

Chapter 3

The Self-Consistent ElectRostatic Potential Algorithm (SCERPA)

Self-Consistent ElectRostatic Potential Algorithm (SCERPA) [11, 10] has been introduced by the VLSI Lab group at the Department of Electronics and Telecommunications of Politecnico di Torino as an effective tool aimed at the analysis of circuits based on the molecular FCN paradigm. Its main strength is the ability to model every required aspect of each molecular cell without relying on computationally expensive ab initio methods, significantly reducing simulation time compared to contender tools.

SCERPA is implemented as a collection of MATLAB scripts that leverage the molecular electronic model proposed by the MosQuiTo methodology in Section 2.2.3 [50]. The algorithm employs a self-consistent iterative loop to solve all the electrostatic interactions among molecules of a device. SCERPA generates an approximate solution that accurately describes the charge distributions of every molecule involved in the computation and propagation process.

In addition to its core functionality, the procedure provides supplementary analysis, such as the signal energy required for propagation and information processing [7]. Furthermore, the tool includes a viewer engine that visualizes the quantities generated throughout the simulation, providing a more intuitive understanding of the system's behavior.

SCERPA is organized into multiple parts, each with a specific focus. Figure 3.1 and 3.2 depict the directory structure and the algorithm scheme, respectively.

3.1 The layout

The layout section provides tools for defining the physical details of mFCN systems. The circuit structure can be specified directly within MATLAB using a matrix or importing a `qll` file created in the MagCAD graphical editor. For instance, a four-cell wire can be defined in MATLAB using the following lines:

```
circuit.structure = {'Dr_1' '1' '1' '1' '1' '2' '2' '2' '2' 'out_y'};
```

```
SCERPA
├─ Algorithm:  Carrying the molecular simulations.
│  └─ ...
├─ Database:  Ab initio molecule data.
│  └─ ...
├─ DemoFiles:  Showing the tool capabilities.
│  └─ ...
├─ Documentation:  How to use the tool.
│  └─ ...
├─ Layout:  Importing the circuit layout.
│  └─ ...
├─ Viewer:  Representing the simulation results.
│  └─ ...
└─ SCERPA.m
```

Figure 3.1. SCERPA directory structure.

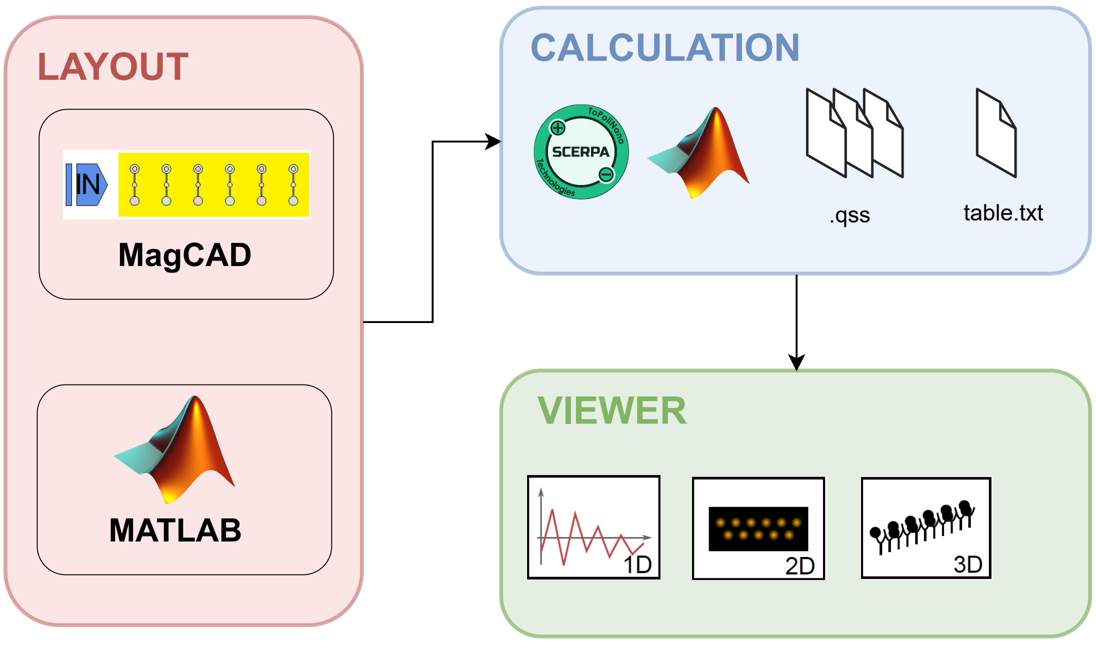


Figure 3.2. Schematic of SCERPA highlighting the main sections.

where `Dr_1` and `out_y` denote the driver and output of the circuit, respectively, while integers specify the clock phase of standard molecules. The driver logic values can be set using the `circuit.Values_Dr` field. Additionally, the `circuit.shift` and `circuit.rotation` attributes specify shifts and rotations of individual molecules, which can help model the presence of process variations and more realistic device manufacturing scenarios. To define the molecule to be used for the circuit, `circuit.molecule` is employed.

Each molecule is fully characterized by a `info.txt` file stored in the `Database` folder. For example, the bis-ferrocene molecule is described as:

```
CHARGES 4
-3.622 -5.062 -0.094
-3.588 +5.083 -0.094
+3.133515 -0.011731 -0.755
+11.776298 -0.053777 +0.409

ASSOCIATION 3
1 3
2 3
3 4

CLOCKDATA 3
ck1.txt -2 V 39 values
ck2.txt 0 V 39 values
ck3.txt +2 V 37 values
```

The `CHARGES` section lists the coordinates of each atom in the molecule, while the `ASSOCIATION` part specifies the connections among atoms for graphical representation. The `CLOCKDATA` section contains references to the transcharacteristics of the molecule obtained through ab initio calculations under different clock conditions.

Time is discretized into a predefined number of steps $\{t_0, \ldots, t_T\}$. The layout defines clock specifications for each time step through the `circuit.stack_phase` setting. The mapping between each time step and its clock value is stored in a clock field matrix. Its generic element $\{i, \tau\}$ contains the electric clock field $E^{\tau}_{clk,i}$ of *Molecule i* at the instant t_τ.

Alternatively, MagCAD `qll` description provides a more intuitive way to define a sub-set of layout-oriented parameters, including molecule type, position, shifts, rotations, and clock regions, through a graphical interface. Section 4.2 provides an example of `qll` content.

3.2 The algorithm

The pseudocode of SCERPA is reported in Algorithm 1. The procedure is divided into four main steps: initialization, selection, evaluation, and output.

Initialization step

The algorithm initiates by importing the information generated during the layout phase, along with any algorithmic-oriented options of the procedure not strictly related to layout details. These options include settings that influence how the simulation evolves and stops, such as approximation, precision, and convergence. During this phase, the molecule transcharacteristics are reshaped over a predetermined number of voltages to improve performance and reduce interpolation costs. Initially, all the molecules are set with a null voltage, and the effect of the driver D on each *Molecule i* is computed as $V_{D,i}$, using Equation (2.13). Moreover, the algorithm creates a distance matrix that stores the distances among every molecule to generate the Interaction Radius (IR) list.

Selection step

At each time step t_τ, the procedure translates the interactions among all the molecules in the circuit into a nonlinear system, solved during the evaluation step. More specifically, the input voltage of *Molecule i* is computed as:

$$V^{\tau}_{in,i} = V^{\tau}_{D,i} + \sum_{\substack{j=1 \\ j\neq i}}^{N_{AC}} V^{\tau}_{j,i}\left(V^{\tau}_{in,j}, E^{\tau}_{clk,j}\right) \tag{3.1}$$

Evaluation Step

The non-linear system reported in Equation (3.1) is solved iteratively as:

$$V^{k,\tau}_{in,i} = F_i\left(V^{k-1,\tau}_{in,1}, \ldots, V^{k-1,\tau}_{in,i-1}, V^{k-1,\tau}_{in,i+1}, \ldots V^{k-1,\tau}_{in,N}\right) \tag{3.2}$$

where k is the self-consistent field (SCF) step of the loop and F_i refers to Equation (3.1). At each generic k, Equation (3.2) illustrates how all the molecules in the circuit affect one another. Furthermore, the voltage variation between two consecutive steps is computed as $dV_i = |V^{k}_{in,i} - V^{k-1}_{in,i}|$, and leveraged to evaluate the convergence of step k.

In particular, if $dV_i < \varepsilon_{max}$, it indicates that consistency between AC and voltage is reached and the molecules no longer influence each other. Once every molecule gains stability, the algorithm is considered to have converged to a solution for the specific time step t_τ. The evaluation stage ends when convergence is achieved for all time steps T.

During the process, charges to compute $V^{k}_{in,i}$ are derived by applying the VACT.

Output Step

The procedure generates a set of output files according to the options specified in the initialization step. These files include:

- `qss` files: They contain the charge distributions of each molecule in the circuit for each time step t_τ. The viewer can read these files to generate their graphical representation in different forms.
- `txt` files: They contain additional data used during the simulation or by the viewer.
- `jpg` files: They are produced as combinations of the two data above and include:

- 1D charge figures: They represent how the charge distribution evolves in a one-dimensional perspective.
- 3D charge figures: The same as 1D charge figures, but in a 3D fashion.
- Potential figures: They show the voltage generated by the molecular charge distribution.
- Logic figures: They show the information propagation through the circuit using standard QCA encoding.
- Waveform figures: They represent the drivers, outputs, and clock values in the form of waveforms, easing the verification of the circuit logic correctness.

3.3 The optimizations

Although the use of VACT to evaluate the charge-voltage relationship resultes in a significant reduction in simulation time by eliminating ab initio calculations, the algorithm still has two nested loops, resulting in a $O(N^2)$ complexity. This complexity can make the algorithm computationally expensive for circuits with a large number of molecules. To address this issue, two optimization techniques are used to reduce simulation time: Interaction Radius (IR) and Active Region (AR) [11, 10].

3.3.1 Interaction Radius

The Interaction Radius (IR) technique takes advantage of the fact that faraway molecules have a negligible influence on each other due to the inverse proportionality of the Coulomb potential expression to the distance. During the selection phase for a given *Molecule i*, only neighboring molecules within a maximum d_{IR} are considered.

A large d_{IR} reduces the number of interactions considered for the computation of the *Molecule i* input voltage, thereby reducing the required simulation time. Despite that, the final solution is less accurate since it approximates what occurs in a real scenario. On the other hand, a smaller d_{IR} produces a larger interaction radius list since more neighboring elements influence the *Molecule i*. In this case, the solution is more precise, but it requires a higher amount of simulation time.

To strike a balanced d_{IR} value, the authors suggest that the interaction radius should be approximately six intermolecular distances, providing a reasonable trade-off between simulation time and solution precision [6].

3.3.2 Active Region

The Active Region (AR) is based on the experimental observation that the polarization of a *Molecule i* is mainly influenced by its neighboring molecules. Therefore, the input voltage of *Molecule i* is approximated as null if its neighbors are not polarized. In addition, the authors in [11] show that once a molecule becomes polarized, its input voltage in a given step is very similar in the immediate next steps. As a result, the

Algorithm 1 SCERPA pseudocode

```
 1: Import simulation settings.                                   ▷ Initialization stage
 2: Import molecular layout.
 3: Import and reshape molecule transcharacteristics.
 4: Compute distance matrix.
 5: for each molecule as $i$ do
 6:     for each molecule as $j$ do
 7:         Evaluate distance between $i$ and $j$.
 8:         if distance $< d_{IR}$ then
 9:             Add $j$ in the IR list of $i$.
10:         end if
11:     end for
12: end for
13: for each time step as $\tau$ do                               ▷ Selection stage
14:     Evaluate driver effects.
15:     Update driver effects.
16:     Update clock effects.
17:     while $|dV_i| < \varepsilon_{max}$ do                      ▷ Evaluation stage
18:         Create active region list as AR based on $dV_i > V_{AR}$.
19:         for each molecule in AR list as $i$ do
20:             Evaluate driver effect on $i$ as $V_{in,i} = V_{D,i}$.
21:             for each molecule in IR list of $i$ as $j$ do
22:                 Evaluate $j$ effect on $i$ as $V_{i,j}$.
23:                 Update effect on $i$ as $V_{in,i} = V_{in,i} + V_{i,j}$.
24:             end for
25:         end for
26:         Evaluate voltage variation $dV_i$.
27:     end while
28:     Output charge distributions.                              ▷ Output stage
29:     Output energy consumption.
30: end for
31: Generate simulation graphs.                                   ▷ Viewer
```

evaluation step of the procedure is completed only for those molecules that are considered active.

An active element is a *Molecule i* whose voltage variation $dV_i = |(V_{in,i}^{k} - V_{in,i}^{k-1})|$ between two consecutive steps k and $k-1$, is greater than a certain value, V_{AR}. All active elements for a given molecule are organized in the active region list of *Molecule i* determined during the initialization phase. By setting a small V_{AR}, the computation for the specific step k can be accelerated. Nevertheless, just like with the d_{IR} parameter, the V_{AR} values should be chosen carefully to avoid obtaining poor quality results as the approximation made by the optimization may not hold for larger values of V_{AR}.

While interaction radius and active region techniques do not directly affect the overall complexity of the algorithm, they can significantly reduce the number of calculations required, resulting in an almost N-independent computational cost [11]. As the number of molecules N increases, the increase in computational time will be much less than N^2, allowing for faster and more efficient simulations. By leveraging the two optimizations together, the algorithm can minimize the computational cost while still providing accurate results. A combination of the two is particularly advantageous for large circuits with many molecules, where the reduction in simulation time can be critical for achieving efficient and effective simulations.

3.4 The convergence

As explained in Section 4.1, SCERPA terminates when a solution to the nonlinear problem represented by Equation (3.2) is found. The problem is simplified to a fixed-point numerical task whose convergence is ensured whenever $||J_F|| < 1$ for some matrix norm [53]. J_F represents the Jacobian matrix of function F and shows how the *Molecule i* is influenced by a *Molecule j* in terms of input voltage. The complete expression is: $\{J_F\}_{i,j} = \delta V_{in,i}/\delta V_{in,j}$ [11]. The author of [3] demonstrates that the algorithm convergence is mainly influenced by charge distances and transcharacteristics slope. In particular:

- The convergence is improved by larger intermolecular distances.
- The convergence is disfavored by larger V_{in}–AC transcharacteristic (VACT) slope.

The first point suggests that molecular FCN systems with short intermolecular distances are disfavored in terms of convergence. Experimental results show this tendency [3]. In this scenario, SCERPA can favor its convergence by using a damping parameter $0 \le \xi \le 1$, which reduces the voltage variations between two consecutive steps of the iterative procedure. Equation (3.2) becomes:

$$V_{in,i}^{k,\tau} = \xi V_{in,i}^{k-1,\tau} + (1-\xi)\, F_i\left(V_{in,1}^{k-1,\tau}, \dots, V_{in,N}^{k-1,\tau}\right) \tag{3.3}$$

The damping can be deliberately set as a value in the range $[0-1]$ considering a higher damping ξ can enhance convergence but also increase the necessary simulation time. For these reasons, the ξ parameter should be chosen considering the needs of the specific circuit under analysis or whether it is required to favor the convergence or to reduce time analysis. The two limit cases $\xi = 0$ and $\xi = 1$ should be avoided as:

- having $\xi = 0$ implies Equation (3.3) to become $V_{in,i}^{k,\tau} = F_i\left(V_{in,1}^{k-1,\tau}, \ldots, V_{in,N}^{k-1,\tau}\right)$. In this case, the damping is disabled, preventing any of its benefits.
- having $\xi = 1$ implies Equation (3.3) to become $V_{in,i}^{k,\tau} = V_{in,i}^{k-1,\tau}$. In this case, the convergence loop is prevented from continuing, thus producing a final solution.

The main drawback of this implementation is, in general, the necessity to perform multiple tests to come up with a suitable value. As explained before, this is particularly true for situations where the circuit is prone to convergence problems due to critical operating conditions or molecules in use. In addition, once the sub-optimal ξ is set, the same parameter is statically applied from the first step down to the last of the procedure iterative loop, regardless of the convergence level reached throughout the analysis. To overcome these limitations, Section 6.1 proposes a dynamic damping alternative.

Chapter 4

The ToPoliNano EDA framework

The field of Electronic Design Automation (EDA), also known as Electronic Computer-Aided Design (ECAD), refers to any software tool aimed at designing, verifying, and testing electronic systems, including integrated circuits (ICs) and printed circuit boards (PCBs). Over the past half-century, the integration of EDA and CAD has played a leading role in the evolution of silicon-based systems, providing designers with essential technologies for the development and production of high-volume integrated circuits efficiently and predictably.

One of the EDA and CAD most remarkable characteristics has been the capacity to combine interdisciplinary knowledge from a wide variety of science fields, mathematics, material sciences, physics, and chemistry in computation theory. This multidisciplinary approach has resulted in an impressive ability to abstract low-level physical properties of devices, realize a separation of concerns, and develop a well-defined and scalable flow to support the work of designers [14].

Furthermore, the collaboration between the EDA industry and the academic community has strengthened over the years as they both address the challenges of the semiconductor industry. Through extensive research, the EDA community has continuously developed new algorithms and optimization techniques to cope with the increasing complexity of designing and testing new integrated circuits. The industry's success is a testament to the contribution of the EDA community [14].

It is safe to say that Moore's law would probably have never come true without the tools provided by the EDA domain to pursue continued growth and innovation in the semiconductor industry.

Nonetheless, as the semiconductor industry scaling trend slows down, it becomes increasingly important to explore innovative computing paradigms. Therefore, it is crucial to develop new EDA and CAD tools that can assist in the development, verification, and testing of electronic systems based on emerging technologies.

In this context, research remains central in collaborating with the industry to bridge the gap between academic and real-world applications. One example is the ToPoliNano EDA framework, a promising set of tools for studying non-conventional computing paradigms based on Field-Coupled Nanocomputing technology.

4.1 ToPoliNano EDA framework

The ToPoliNano framework [55, 54, 27] provides a comprehensive suite of EDA software tools for designing, simulating, and testing circuits based on beyond CMOS technologies. The central idea of the suite is to provide a top-down approach as similar as possible to the one provided by traditional EDA software.

The suite is developed in C++ using the Qt libraries by the VLSI Lab group at the Department of Electronics and Telecommunications of Politecnico di Torino. It is constantly under development to enlarge the support to other emerging technologies. As for now, it supports magnetic FCN, both in-plane and out-of-plane implementations, and partially molecular FCN.

The framework is composed of three nanoelectronic applications: MagCAD, a graphical 3D editor used to design emerging computing devices; ToPoliNano, a CAD tool for studying and simulating these systems, recently embedded with SCERPA; and FCN-viewer, a 3D visualization program used to represent and check for correct information propagation and elaboration. Thanks to the portability offered by the Qt libraries, all three software are available on Windows, Mac, and Linux. Figure 4.1 reports a schematic representation of the suite.

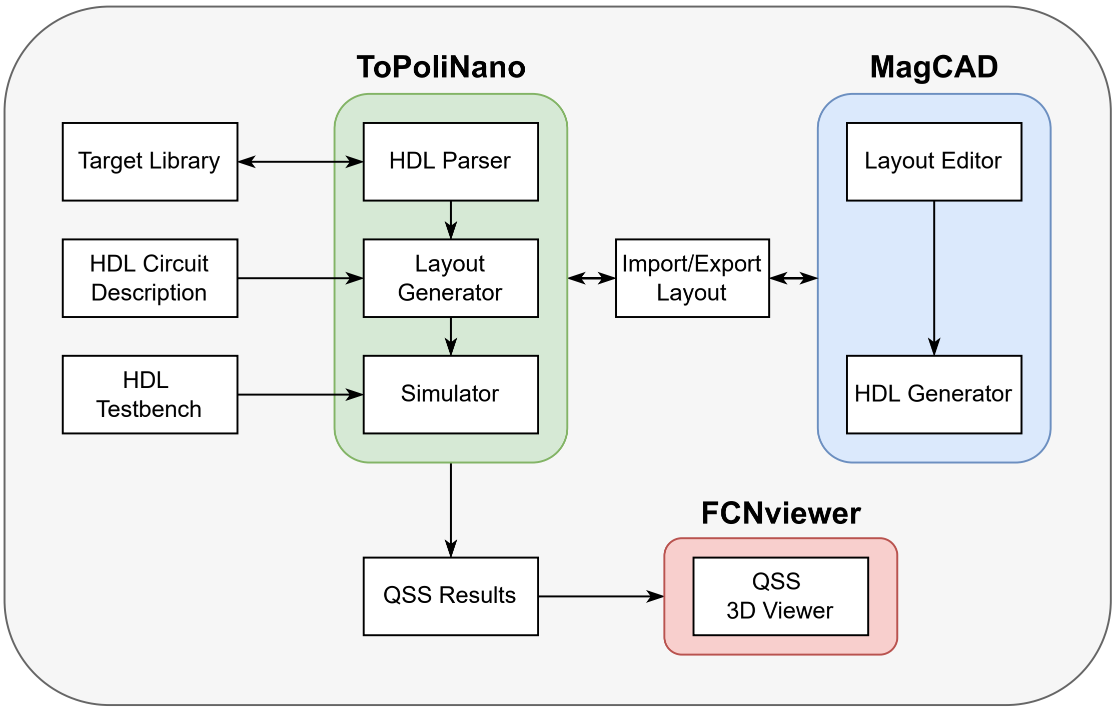


Figure 4.1. The ToPoliNano EDA framework.

4.1.1 ToPoliNano simulator

ToPoliNano [55] is a software tool designed to simulate devices based on cutting-edge technologies that extend beyond the current silicon approach, including magnetic and molecular FCN implementations. The tool is developed with a top-down approach that closely mirrors the workflow offered by commercial digital EDA tools, providing a familiar and intuitive environment to analyze novel circuits.

To perform simulations, the tool can import circuits in one of two ways, depending on the technology. For magnetic, it requires a structural/hierarchical HDL (Hardware Description Language) format, while for both magnetic and molecular circuits, it can import a graphical representation created in MagCAD.

The software employs VHDL testbenches to instantiate elements as digital components, thus reinforcing its affinity with commercial tools simulation workflows. Moreover, ToPoliNano offers an intuitive and straightforward graphical interface that enables users to define all the physical parameters and analysis settings.

Compared to research-oriented simulators like SCERPA, the simulator provides a single, comprehensive tool for analyzing multiple novel technologies. Its C++ implementation delivers better efficiency and performance compared to MATLAB coding. Additionally, its user-friendly graphical interface makes it a valuable candidate for commercial beyond-CMOS software in the near future.

On the other hand, ToPoliNano may currently lack support for additional in-depth functionalities that are more easily explored using a scientific and research-oriented approach with coding languages like MATLAB. Therefore, the best is to adopt a collaborative combination of academic and EDA tools like SCERPA and ToPoliNano. The former can be used to gain a more comprehensive and in-depth understanding of the technology, while the latter can assess the integrability of the research insights and determine which features are truly useful for more commercial-oriented software.

4.1.2 MagCAD graphical editor

MagCAD [54, 27] is a 3D graphical editor to sketch circuits based on emerging technologies. At the current time, the supported technologies are Nano Magnet Logic (NML), in-plane (iNML) and perpendicular (pNML) versions, and molecular FCN (mFCN). The tool focuses on simplifying the circuit design, providing an intuitive Graphical User Interface (GUI).

In addition, the editor can realize hierarchical circuits by reloading and reusing the previously extracted components. Once the technology is decided, the layout realization process can start by directly placing the building blocks of the target technology library in the desired location.

In the case of magnetic logic technology, the elementary blocks consist of nanomagnets interacting through an anti-ferromagnetic fashion, while in the case of molecular FCN, the elementary blocks are mQCA cells made of a couple of molecules interacting electrostatically.

All the parameters of the specific emerging technology, design rules, and physical constraints are set by the designer and internally handled by the software. Once the design is completed, the tool saves the circuit description that can be imported into a simulator tool like ToPoliNano to asses its correctness and evaluate its performance.

4.1.3 FCNviewer

Field-Coupled Nanocomputing (FCN) Viewer is software to be used in conjunction with the output simulation files generated by the ToPoliNano simulator. The tool creates a free explorable 3D graphical representation of the entire circuit, allowing one to asses its intended behavior. It currently supports magnetic and molecular visualizations.

Figure 4.2 displays the complete ToPoliNano framework flow to design, simulate and validate mFCN circuits.

4.2 Detailed overview of molecular ToPoliNano

The molecular ToPoliNano simulator enables the analysis of mFCN circuits. A parser function explores the `qll` layout designed in MagCAD and extracts all relevant characteristics, including general information such as intermolecular distances, clock phases, global circuit dimensions, technology type, and employed molecules. Furthermore, specific information about each cell composing the circuit is imported, such as its coordinates, clock phase belonging, name, ID, shifts, rotations, and whether it acts as a driver. As an example, Figure 4.3 depicts a majority voter designed in the MagCAD environment, while the correspondent `qll` is provided in Listing 4.1.

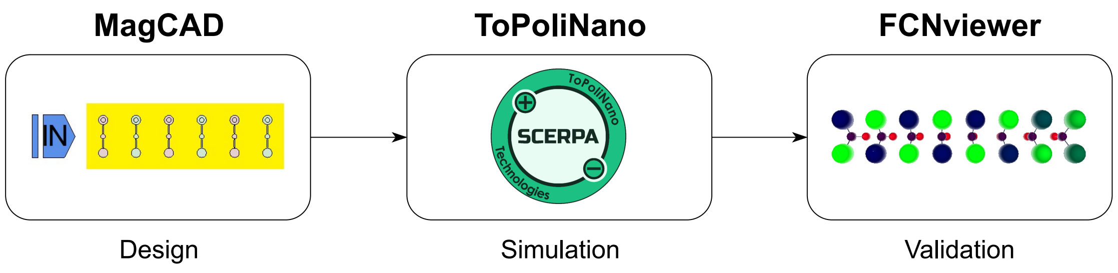


Figure 4.2. ToPoliNano's EDA framework flow to desing, simuale and validate mFCN circuits.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--File describing the layout of a QCA circuit-->
<qcalayout>
    <technologies>
        <settings tech="MolFCN">
            <property name="Layoutwidth" value="6"/>
            <property name="Intermolecular Distance" value="1000"/>
            <property name="PhaseNumber" value="3"/>
```

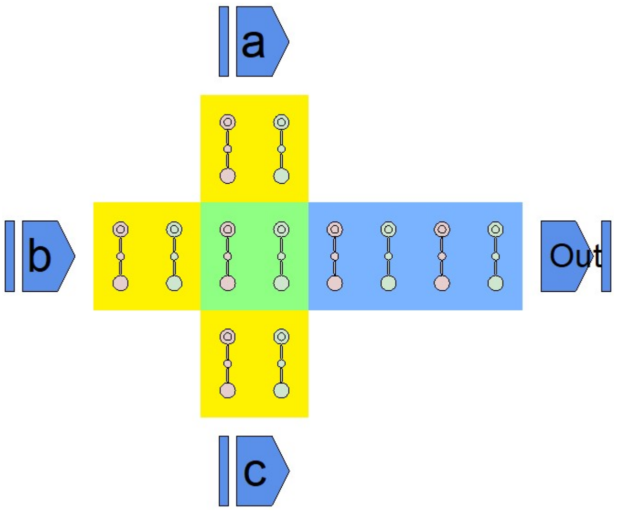


Figure 4.3. Single-line mFCN majority voter layout implemented with MagCAD editor.

```
            <property name="Layoutheight" value="5"/>
            <property name="CZSequence" value="4"/>
            <property name="layersEnabled" value="false"/>
        </settings>
    </technologies>
    <components>
        <item tech="MolFCN" name="Bisferrocene"/>
    </components>
    <layout>
        <pin tech="MolFCN" name="Out" direction="1" id="1" x="6" y="3"
layer="0"/>
        <pin tech="MolFCN" name="C" direction="0" id="2" x="3" y="5"
layer="0"/>
        <pin tech="MolFCN" name="B" direction="0" id="3" x="1" y="3"
layer="0"/>
        <pin tech="MolFCN" name="A" direction="0" id="4" x="3" y="1"
layer="0"/>
        <item comp="0" id="5" x="5" y="3" layer="0">
            <property name="phase" value="2"/>
        </item>
        <item comp="0" id="6" x="4" y="3" layer="0">
            <property name="phase" value="2"/>
        </item>
        <item comp="0" id="7" x="3" y="3" layer="0">
            <property name="phase" value="1"/>
        </item>
        <item comp="0" id="8" x="3" y="4" layer="0">
            <property name="phase" value="0"/>
        </item>
        <item comp="0" id="9" x="3" y="2" layer="0">
            <property name="phase" value="0"/>
        </item>
        <item comp="0" id="10" x="2" y="3" layer="0">
            <property name="phase" value="0"/>
        </item>
    </layout>
</qcalayout>
```

Listing 4.1. The `qll` file representing the majority voter reported in Figure 4.3.

A VHDL file acting as a testbench is required to instantiate the device under test and associate input values for the simulation. Listing 4.2 displays the VHDL testbench for a majority voter.

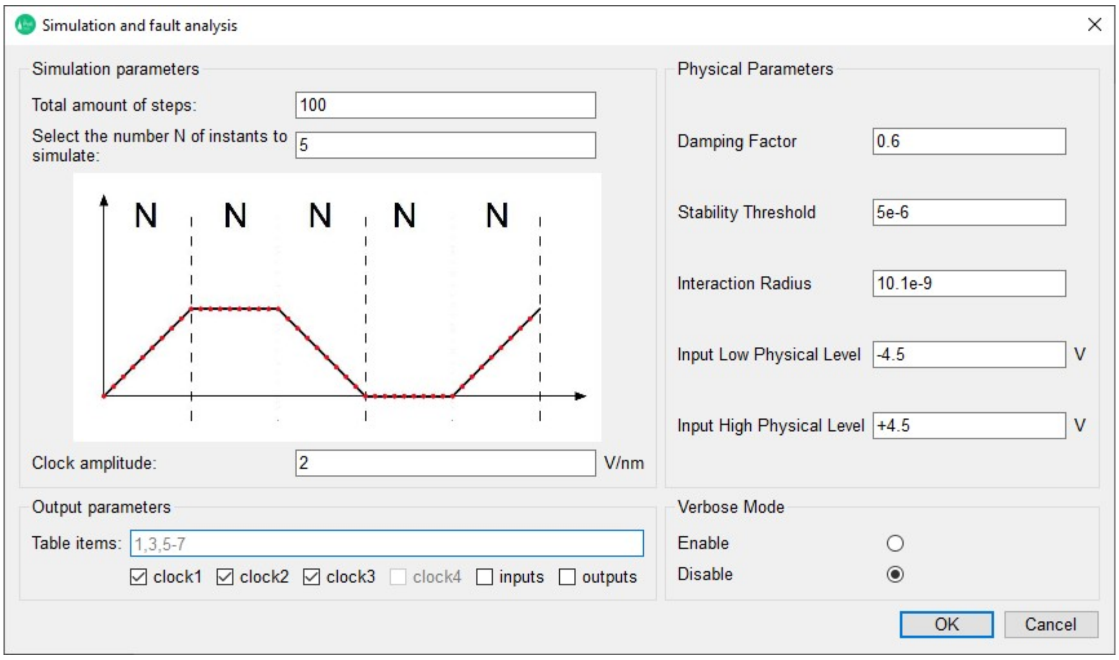
```
entity testbench is
end testbench;

architecture behavioral of testbench is
  signal a, b, c, o: std_logic;

  component test is
      port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            o: out std_logic);
        end component;
begin
  DUT: test port map(a, b, c, o);

    INP_PROC: process
    begin
        a <= '0';
        b <= '1';
        c <= '1';
        wait for 200ns;
  end process;
end behavioral;
```

Listing 4.2. The VHDL testbench for simulating the majority voter layout.

Once the circuit layout and its respective testbench are associated, the user configures the analysis operating on the GUI shown in Figure 4.4. The available options include simulation parameters, such as the number of steps and N, representing the number of points used to interpolate the clock waveforms. The former impacts the simulation duration, while the latter, if increased, can enhance the accuracy of the simulation, bringing it closer to a complete adiabatic solution.

Moreover, physical parameters, including damping factor, interaction radius, clock voltage levels, and stability threshold, are also specified, following the same meanings as in the original SCERPA.

The tool also allows for a verbose mode, which prints out every charge distribution of the self-consistent loop for each not-yet-converged step. Users may also specify any output parameters to be represented, including clock waveforms and the charge distribution of specific molecules. A `table.txt` file saves this information, which the FCNviewer can import to generate graphical representations.


Simulation and fault analysis

Simulation parameters

Total amount of steps: 100

Select the number N of instants to simulate: 5

Clock amplitude: 2 V/nm

Output parameters

Table items: 1,3,5-7

☑ clock1 ☑ clock2 ☑ clock3 ☐ clock4 ☐ inputs ☐ outputs

Physical Parameters

Damping Factor	0.6
Stability Threshold	5e-6
Interaction Radius	10.1e-9
Input Low Physical Level	-4.5 V
Input High Physical Level	+4.5 V

Verbose Mode

Enable [✓ Disable]

Figure 4.4. ToPoliNano's GUI to specify simulation parameters.

ToPoliNano employs the methods of the `MolecularSimulationController` class to carry on the molecular analysis. The pseudocode of the molecular engine is reported in Algorithm 2. The simulation begins with the `ReadMolecules()` method, which imports the transcharacteristics of the supported molecules. Unlike the original SCERPA, ToPoliNano's transcharacteristics are pre-interpolated over a predetermined number of values, reducing computational costs. Afterward, the controller applies the user-specified physical parameters.

Following this, the controller initializes clock entities based on the number of phases in the circuit via the `createClocks()` function. Next, the controller iterates over each `QcaItem` in the layout, distinguishing between drivers and non-drivers. For each item, two `MolecularElements` are created by assigning molecule initial charges and local positions within the cell. If specified, shifts and rotations are applied. At this point, all molecules of the original circuit are internally conceived as nodes in a graph. The controller completes the layout modeling by creating edges to model the influence interactions between molecules. In particular, the `buildInteractions()` method iterates over all the molecules, computing distances between each pair. If the gap is smaller than the interaction radius, an edge represents the interaction, with directions and distances between every dot. The process provides a comprehensive representation of every interaction within the circuit, similar to what SCERPA does, as explained in Section 3.2.

The simulation begins with the `startSimulation()` method, which first generates clock waveforms based on the GUI parameters. Then, a loop iterates over every simulation step, with the `setInputValue()` procedure setting the charge values of the driver cells according to the digital inputs specified in the testbench.

For each molecule in a given simulation step, the `evaluateInteraction()` method computes the effect of all other molecules within the specified interaction radius. In particular, the `computeNewValue()` method determines the voltage difference across Dot1 and Dot2 of each molecule. The voltage is then linked with the applied clock field to obtain the aggregated charges through the transcharacteristics. Then, the ACs are multiplied by the damping factor to favor convergence and applied to the molecule in its new state.

The `advanceStep()` function evaluates the convergence of each step by comparing the charge values between two consecutive steps. If the difference is above the stability threshold, the step is not yet stable, and the solution requires refinements until convergence.

ToPoliNano differs from SCERPA in the approach used to verify convergence. While SCERPA exploits the voltage difference between consecutive steps, ToPoliNano employs aggregated charges linked to the voltages through the transcharacteristics.

Once each step has reached convergence, the `qssWriteOnFile()` method saves all the molecules' charge distributions throughout the simulation in separate `qss` files for each step. Additionally, a `table.txt` file, created by `tableWriteOnFile()` function, stores data about the clock. Moreover, if any molecule IDs were specified in the table items field of the GUI, the correspondent charge values are also included in the `table.txt` file.

The FCNviewer can import the `qss` and `table.txt` files to provide graphical representations of how information propagates through the simulation, clock, and charge behavior. For instance, Figure 4.5 illustrates the molecular majority voter displayed in Figure 4.3, simulated at four different time instances: (a) when all cells, except drivers, are in reset, (b) the propagation of the driver value to the central cell, (c) the evaluation of the central cell based on the most recurrent input, and (d) the final propagation of the result toward the output. The `table.txt` content visualized in Figure 4.6 shows the three clock waveforms and the charge variations of the output cell during the simulation.

Algorithm 2 ToPoliNano's molecular core pseudocode.

```
 1: Import molecules transcharacteristics.                          ▷ ReadMolecules()
 2: Initialize clocks.                                              ▷ CreateClocks()
 3: Import molecular layout.                       ▷ MolecularSimulationController()
 4: Import simulation settings.
 5: Create and initialize molecules.
 6: for each molecule as $i$ do                                ▷ BuildInteractions()
 7:     for each molecule as $j$ do
 8:         Evaluate distance between $i$ and $j$.
 9:         if distance $< d_{IR}$ then
10:             Add $j$ in the IR list of $i$.
11:         end if
12:     end for
13: end for
14: Generate clocks waveforms.                                       ▷ createClock()
15: for each time step as $\tau$ do                              ▷ StartSimulation()
16:     Update driver value.                                      ▷ setInputValue()
17:     while $\tau$ convergence is not reached do
18:         for each molecule as $i$ do                      ▷ EvaluateInteraction()
19:             for each molecule within IR of $i$ as $j$ do   ▷ computeNewValue()
20:                 Evaluate voltage effect of $j$ on $i$.
21:                 Apply transcharacteristic to retrieve charge values.
22:                 Update effect on $i$.
23:             end for
24:         end for
25:         Evaluate charge variation.                              ▷ advanceStep()
26:     end while
27:     Output charge distributions.                            ▷ qssWriteOnFile()
28: end for
29: Output clocks information.                                ▷ tableWriteOnFile()
```

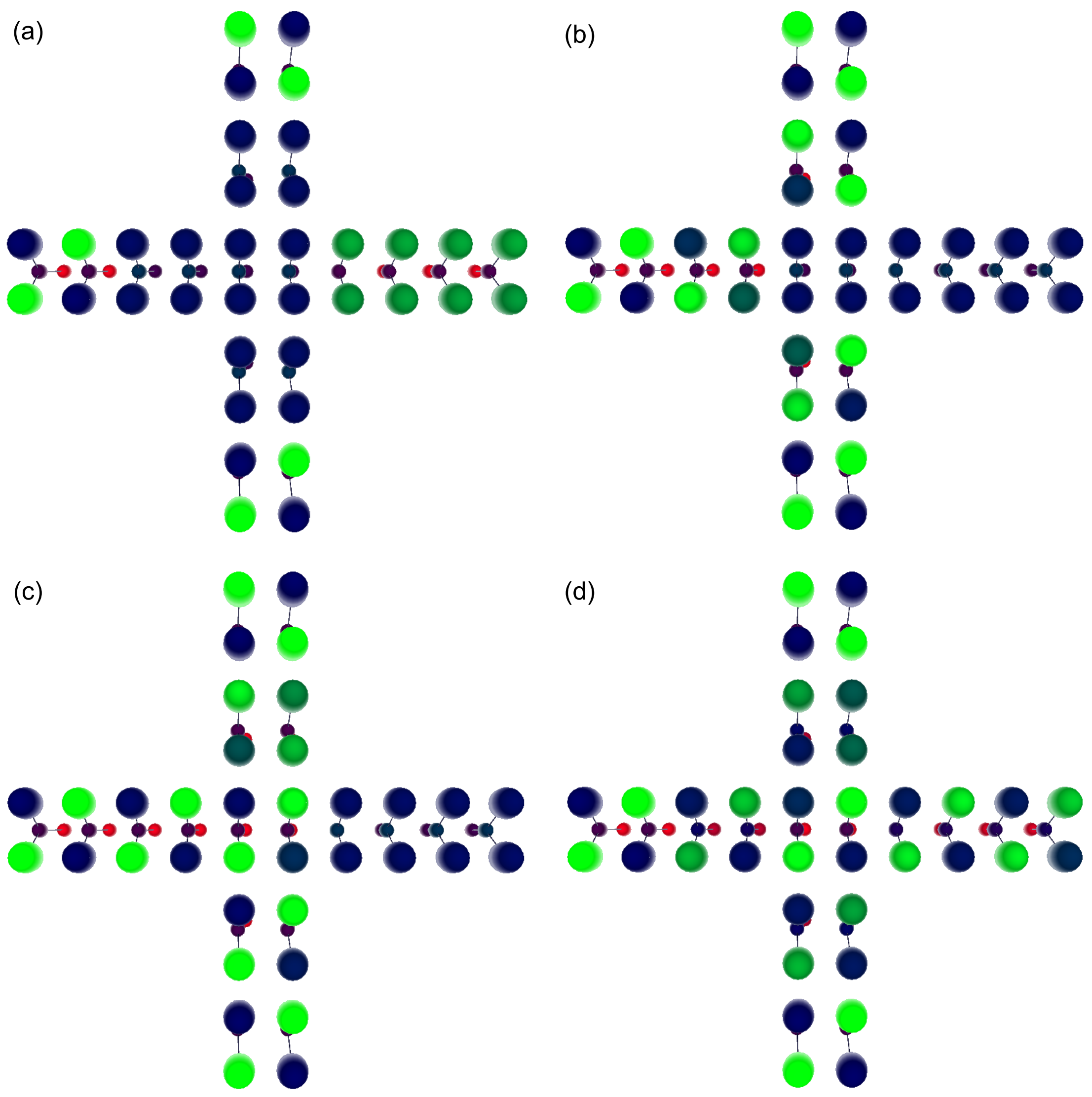


Figure 4.5. FCNviewer graphical representation of the majority voter simulated at four different time instances: (a) when all cells, except drivers, are in reset, (b) the propagation of the driver value to the central cell, (c) the evaluation of the central cell based on the most recurrent input, and (d) the final propagation of the result toward the output.

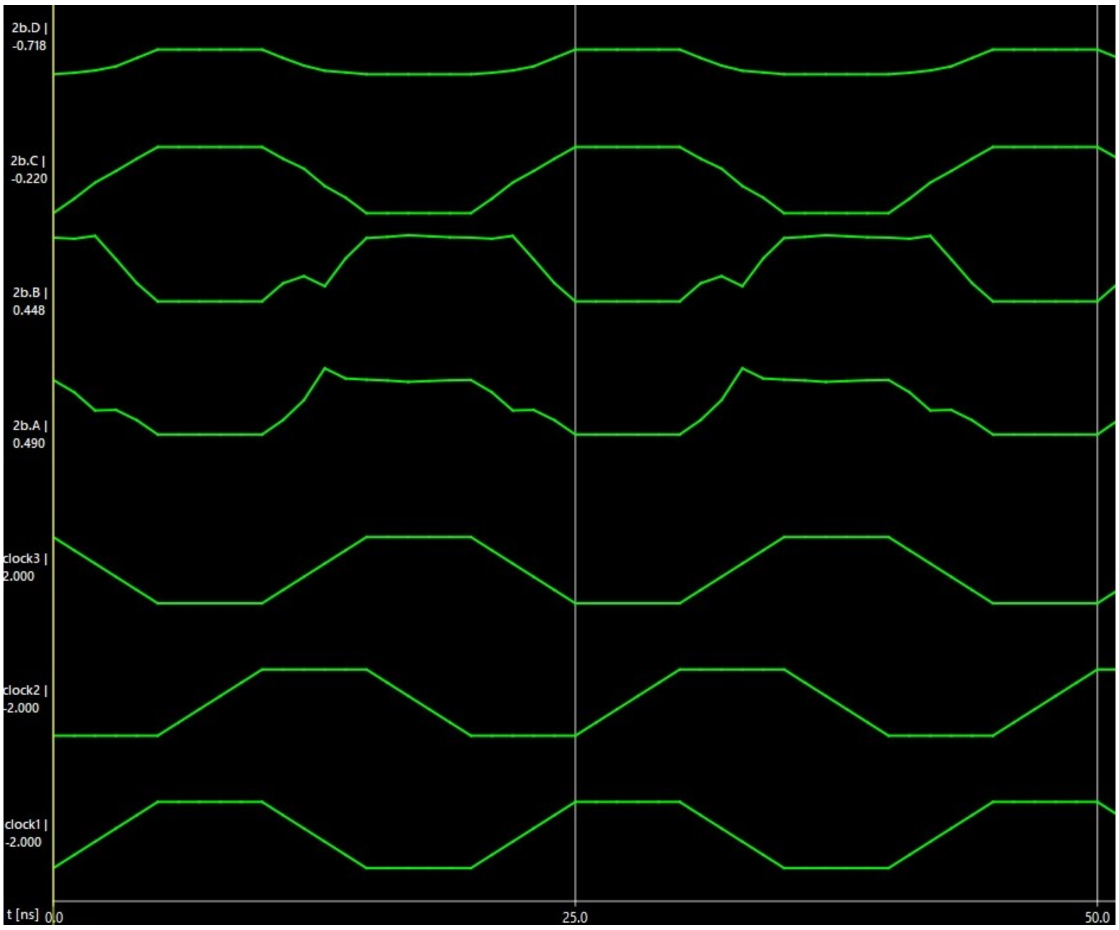


Figure 4.6. FCNviewer graphical representation of the clock waveforms and the charge variations of the output cell dots during the simulation.

Chapter 5

Comparative analysis of the molecular FCN simulators

One of the main focuses of this work is to compare and validate the results generated by the two SCERPA versions described previously:

- The research-oriented version that allows for importing circuits using either a MATLAB matrix textual description or a graphical MagCAD representation.

- The EDA/CAD-oriented version that is already integrated into the ToPoliNano simulator.

To accomplish the goal, a two-stage validation methodology, taking the SCERPA academic version with the MATLAB importer as the point of reference, is proposed.

The first stage aims at assessing consistency between the two SCERPA importers and identifying discrepancies between them. Both importers are expected to behave similarly regardless of the circuit definition mode. The two versions share the same molecular analysis core, with the only differences lying in how the two versions import a circuit and extract the required information. Therefore, both versions are foreseen to produce the same charge distributions throughout the simulation with minimal differences for every step.

On the contrary, the second stage focuses on comparing the SCERPA-MagCAD outputs that have been validated in the first stage with the ones generated by the CAD counterpart. The ToPoliNano simulator and the MagCAD graphical editor are part of the same EDA framework, suggesting their compatibility and consistency at the circuit level. Indeed, both tools share the same libraries, which enable the generation of a `qll` layout in MagCAD and its import in ToPoliNano. However, to ensure correct functioning between the original SCERPA version and its CAD counterpart, it is still necessary to validate them to identify and address any potential issues or discrepancies that may have arisen during the integration process.

5.1 Methodology

To ensure a well-structured and organized validation and testing process for the different SCERPA versions, a systematic methodology is derived:

- Define a set of molecular circuits to be analyzed.
- Apply shifts and rotations.
- Define a framework to automate the validation and testing process.

A standardized set of molecular circuits as benchmarks helps maintain consistency across all the SCERPA version comparisons. For this purpose, it includes layouts with an increasing level of complexity:

- Single molecular cell.
- Single phase wire of six molecular cells.
- Three phases wire of twelve molecular cells.
- Two-line three phases wire of twelve molecular cells.
- Two-line T-connection.
- Two-line L-connection.
- Two-line inverter logic gate.
- Two-line majority voter logic.
- XOR logic gate.

Shifts and rotations are employed to simulate process variations that may occur in practical molecular circuit implementations, which can prevent proper information propagation. Therefore, multiple random combinations of shifts and rotations are applied to molecules of a given cell. This approach verifies that all versions of SCERPA can correctly simulate circuits with molecules that are displaced from their original positions.

A framework consisting of a set of Bash scripts is developed to automate the validation and testing process. The Bash language is chosen for its high portability, enabling the framework to be used on any Unix/Linux system without modifications. Additionally, Bash provides powerful built-in tools for manipulating textual files and efficient automation capabilities to automate repetitive tasks, making it a suitable choice for this task. All the scripts are executable from the `Testing` directory located in the SCERPA folder.

5.2 Evaluating SCERPA importers

Each QCA cell is associated with a label that contains an integer serving as its identifier within the circuit. A suffix of either “a” or “b” is appended to the cell label to differentiate between the two molecules within the same cell. As explained in Section 3.1, the SCERPA tool provides two methods for importing molecular layouts: a textual description stored in a `qll` file generated by the MagCAD graphical editor or a textual matrix directly defined in the algorithm caller script. Cell IDs are assigned according to the way the circuit is designed. In the MagCAD case, cells retain the same ID assigned by the editor according to the order of creation. Conversely, in the MATLAB case, cells are labeled as an increasing integer based on their position in the layout, following the order of their location in the circuit plane. Both methods preserve the same suffix to distinguish each molecule within a QCA cell.

A single caller script initializes two consecutive SCERPA calls to simultaneously generate MagCAD and MATLAB simulations. For each version, a folder named after the circuit is created in the main SCERPA directory, with two subfolders, “magcad” and “matlab” containing the corresponding simulation results.

To accurately compare the charge distributions of each molecule across time steps, it is necessary to establish a way to associate the molecules between the two cases. One possible approach to realize associations is by exploiting the spatial coordinates of the Dot4 of each molecule in both layouts. For this purpose, a MATLAB function named `saveLabelsAndDot4Coord` is added to the SCERPA `Algorithm` folder. Its pseudocode and complete code are reported in Algorithm 3 and Appendix A.1, respectively. The function prints the label and Dot4 spatial coordinates of every molecule in two separate files, depending on the type of circuit. It is only called at the end of the simulation if the corresponding setting is flagged. For a six-cell wire layout defined in MATLAB and MagCAD, an example of the function output is reported in Listings 5.1 and 5.2, respectively.

A script named `generateMatchingMolecules.sh` was developed to automate the associations. Algorithm 4 and Appendix A.2 outline its pseudocode and complete code, respectively. The script is executed from the `Testing` folder with the command:

```
./generateMatchingMolecules.sh circuitName
```

where `circuitName` is the name of the circuit.

At first, the script imports the previously collected data on the circuit name provided as input. Subsequently, for each molecule in the MATLAB simulation, the script iterates over every molecule in the MagCAD simulation. If the coordinates of the two molecules match, the program writes their corresponding IDs to the output file. The inner loop is terminated to reduce execution time. For reference, the matches between molecules in the example provided in Listing 5.1 and 5.2 are reported in Listing 5.3.

```
Dr   014 11.7763 -0.0538 0.4090
Mol  01a 11.7763 -0.0538 10.4090
Mol  01b 11.7763 -0.0538 20.4090
Mol  02a 11.7763 -0.0538 30.4090
Mol  02b 11.7763 -0.0538 40.4090
Mol  03a 11.7763 -0.0538 50.4090
Mol  03b 11.7763 -0.0538 60.4090
Mol  04a 11.7763 -0.0538 70.4090
Mol  04b 11.7763 -0.0538 80.4090
Mol  05a 11.7763 -0.0538 90.4090
Mol  05b 11.7763 -0.0538 100.4090
Mol  06a 11.7763 -0.0538 110.4090
Mol  06b 11.7763 -0.0538 120.4090
```

Listing 5.1. ID molecules and Dot4 coordinates of a MATLAB layout.

```
Dr   010 11.7763 -0.0538 0.4090
Mol  07b 11.7763 -0.0538 100.4090
Mol  14a 11.7763 -0.0538 30.4090
Mol  05b 11.7763 -0.0538 60.4090
Mol  02a 11.7763 -0.0538 70.4090
Mol  13b 11.7763 -0.0538 20.4090
Mol  08a 11.7763 -0.0538 110.4090
Mol  05a 11.7763 -0.0538 50.4090
Mol  14b 11.7763 -0.0538 40.4090
Mol  02b 11.7763 -0.0538 80.4090
Mol  13a 11.7763 -0.0538 10.4090
Mol  07a 11.7763 -0.0538 90.4090
Mol  08b 11.7763 -0.0538 120.4090
```

Listing 5.2. ID molecules and Dot4 coordinates of a MagCAD layout.

```
014 010
01a 13a
01b 13b
02a 14a
02b 14b
03a 05a
03b 05b
04a 02a
04b 02b
05a 07a
05b 07b
06a 08a
06b 08b
```

Listing 5.3. MATLAB-MagCAD labels associations.

The comparison of matched molecules between the MATLAB and MagCAD simulations is performed using a script named `checkMatlabMagcadQSS.sh`, whose pseudocode and complete code are provided in Algorithm 5 and Appendix A.3, respectively. The script is executed from the `Testing` folder with the command:

```
./checkMatlabMagcadQSS.sh circuitName
```

where `circuitName` is the name of the circuit whose data are to be validate and used to retrieve the location of the simulation `qss`.

The script compares the aggregate charges of each molecule in the MATLAB and MagCAD simulations directories for every `qss` simulation step. If the molecule IDs match and at least one aggregated charges differ, the program prints the information about the mismatch to the output file and terminates the inner loop. The script outputs the execution time in the end. Listing 5.4 shows an example of print mismatches.

```
Found a mismatch between MATLAB 012.qss and MagCAD 012.qss files:
MATLAB reports:      03a 0.8453435 0.0851060 -0.2134495 -0.717
MagCAD reports:      05a 0.8687645 0.0844106 -0.2078541 -0.717
Differences:             0.0234210 0.0006954  0.0055954  0

```

```
Found a mismatch between MATLAB 018.qss and MagCAD 018.qss files:
MATLAB reports:       02a 0.0107268 0.9142049 -0.2079317 -0.717
MagCAD reports:       14a 0.0094517 0.9264848 -0.2074253 -0.717
Differences:              0.0012751 0.0122799  0.0005064  0
```

Listing 5.4. Example of mismatching printings.

Algorithm 3 Pseudocode for printing molecules IDs and Dot4 coordinates.

```
Require: Drivers and molecules stacks.
Ensure: Molecules IDs and Dot4 coordinates for MATLAB and MagCAD simulations.
 1: Determine output file name based on the simulation type.
 2: for each driver do
 3:     Write identifier and Dot4 spatial coordinates to the output file.
 4: end for
 5: for each molecule do
 6:     Write identifier and Dot4 spatial coordinates to the output file.
 7: end for
```

Algorithm 4 Pseudocode for matching molecule IDs in MATLAB and MagCAD.

```
Require: Molecules IDs and coordinates for MATLAB and MagCAD simulations.
Ensure: Molecules IDs matches and script execution time.
 1: Remove previous output files.
 2: Read the molecule IDs and coordinates of the MATLAB simulation.
 3: Read the molecule IDs and coordinates of the MagCAD simulation.
 4: Create an output file for listing the identified correspondences.
 5: for each molecule in MATLAB simulation do
 6:     for each molecule in MagCAD simulation do
 7:         if coordinates are the same then
 8:             Write the corresponding IDs to the output file.
 9:             Terminate inner loop to reduce execution time.
10:         end if
11:     end for
12: end for
13: if some IDs remain unmatched then
14:     Return an error.
15: end if
16: Print script execution time.
```

Algorithm 5 Pseudocode for comparing MATLAB and MagCAD ACs.

```
Require: Molecules IDs and coordinates for MATLAB and MagCAD simulations and
    circuit name.
Ensure: Mismatches file and script execution time.
 1: Remove any previous output files.
 2: Define MATLAB simulation circuit location.
 3: Define MagCAD simulation circuit location.
 4: for each step $\tau$ in simulation do
 5:     Use circuit name and $\tau$ to define name of the MATLAB qss.
 6:     Use circuit name and $\tau$ to define name of the MagCAD qss.
 7:     for each aggregate charge in MATLAB do
 8:         for each aggregate charge in MagCAD do
 9:             if molecule IDs are equal then
10:                 if aggregated charges are not equal then
11:                     Print information about the mismatch to the output file.
12:                     Terminate the inner loop to reduce execution time.
13:                 end if
14:             end if
15:         end for
16:     end for
17: end for
18: Print script execution time.
```

5.2.1 Result discussion

The above-described scripts were used to analyze the increasingly complex molecular FCN circuits discussed in Section 2.3. As expected, the simulation core remained consistent regardless of the circuit definition, resulting in identical aggregate charge distributions for each molecule between the two circuit modes. However, when molecules were subjected to shifts or rotations were analyzed, differences in the imported molecule coordinates became evident.

Specifically, applying shifts along the x-y-z directions resulted in the MagCAD frame of reference inverting the x and z axis compared to the MATLAB version. As a result, any shifts in the x direction in the graphical-defined circuit led to shifts in the z direction in the MATLAB version, and vice versa. Additionally, employing different shift values to the two molecules of the same QCA cell caused inconsistent coordinates in relation to the applied shifts. The issue arose from an inversion between the two molecules during the import of the MagCAD circuit.

Since the MagCAD editor only supports rotation along the z direction, simulations were made with the correspondent rotation in MATLAB. Nonetheless, even in this case, the application of rotation in the graphical circuit was not coherent with the one generated in MATLAB due to an error in the computation of the center molecules.

Finally, the solutions previously introduced to solve the issues resulted in some graphs no longer being consistent with the new ordering of the molecule stack, thereby generating incorrect graphical representations. As a result, consistency between the `qll` importer and the viewer was restored.

5.3 Evaluating SCERPA and ToPoliNano simulators

Unlike the validation conducted for the two circuit importers, the ToPoliNano simulator and MagCAD editor share the same library core to create and parse `qll` circuits, thus eliminating the need to generate molecule matches between the two SCERPA versions.

Since the ToPoliNano clock waveforms are strictly tied to the parameters specified in its GUI and less personalization is allowed, such values are statically assigned to the clock phase structure of SCERPA.

To automate the comparison process between the charge distributions generated by the two simulators, the script named `checkMatlabTopolinanoQSS` was developed. It follows a similar approach to Algorithm 5, with the pseudocode and complete code provided in Algorithm 6 and Appendix A.4, respectively. However, to avoid the detection of minimal hypothetical charge mismatches as actual discrepancies while still accounting for minor differences that may arise due to the different implementations of the two simulators, the new algorithm includes an additional input that specifies the maximum allowable distance. The script is executed from the `Testing` folder with the command:

```
./checkMatlabTopolinanoQSS.sh circuitName maxAllowedOffset
```

where:

- `circuitName` is the name of the circuit whose data are to be validated and used to retrieve the location of the simulation `qss`.
- `maxAllowedOffset` is the value that determines whether a difference between the charge distributions is considered an actual mismatch or not. This value is used as a threshold, above which a charge mismatch is deemed significant enough to warrant attention.

Listing 5.5 shows an output generated by the script when a mismatch is found.

```
Found a mismatch between MATLAB 005.qss and MagCAD 005.qss files:
Maximum +/- allowed discrepancy: 0.1.
MATLAB reports:       09a 0.125541 0.806824 -0.215365 -0.718
ToPoliNano reports:   09a 0.573384 0.363899 -0.219975 -0.718
Differences are:          0.447843 0.442925  0.00461   0

Found a mismatch between MATLAB 005.qss and MagCAD 005.qss files:
Maximum +/- allowed discrepancy: 0.1.
MATLAB reports:       04b 0.7195854 0.215076 -0.217662 -0.717
ToPoliNano reports:   04b 0.0114838 0.913637 -0.208121 -0.717
Differences are:          0.7081016 0.698561  0.009541  0
```

Listing 5.5. Example of mismatching printings.

5.3.1 Result discussion

The earlier presented scripts were essential in analyzing the charge distributions produced by SCERPA and ToPoliNano for the molecular FCN circuits described in Section 2.3. However, during the initial testing phase, some unusual behavior was observed, resulting in incorrect aggregated charges in particular molecules with values exceeding the charge unit.

Upon further investigation, a problem affecting how transcharacteristics were used to retrieve aggregated charges was identified. Additionally, there was an error affecting coordinate assignments during the creation of each molecule. Furthermore, another issue causing incorrect waveform was found. The error resulted in comparisons that did not consider a threshold for determining if two double-precision floating-point numbers were close enough to be considered equal. To address the problem `std::numeric_limits<double>::epsilon()` was employed to return the smallest possible difference between two double-precision floating-point numbers, leading to accurate comparisons and correct clock waveforms.

To improve the aggregated charges precision, the bis-ferrocene transcharacteristics were substituted with ones with more decimals. Following these corrections, all simple propagation circuits, including the single-cell system, wires, and T- and L-connections, exhibited precise and accurate charge distributions in both simulators. The differences in the four molecule charge dots remained below a mean value of 0.01, which is consistent with the tools' distinct implementations.

On the other hand, when simulating more complex circuits involving computations such as the majority voter, large inverter, and XOR gate, differences in produced charges

Algorithm 6 Pseudocode for comparing ToPoliNano and SCERPA ACs.

```
Require: Molecules IDs and coordinates for ToPoliNano and SCERPA simulations,
    circuit name and maximum allowed distance.
Ensure: Mismatches file and script execution time.
 1: Remove previous output files.
 2: Define ToPoliNano simulation circuit location.
 3: Define SCERPA simulation circuit location.
 4: for each step $\tau$ in simulation do
 5:     Use circuit name and $\tau$ to define name of the ToPoliNano qss.
 6:     Use circuit name and $\tau$ to define name of the SCERPA qss.
 7:     for each aggregate charge in ToPoliNano do
 8:         for each aggregate charge in SCERPA do
 9:             if molecule IDs are equal then
10:                 if aggregated charges are not equal then
11:                     Compute charges differences.
12:                     if charges difference > maximum allowed difference then
13:                         Print information about the mismatch to the output file.
14:                         Terminate the inner loop to reduce execution time.
15:                     end if
16:                 end if
17:             end if
18:         end for
19:     end for
20: end for
21: Print script execution time.
```

persisted between the two software, particularly for small values of N. Despite this, from a graphical perspective, the FCNviewer always displayed consistent propagation and computation patterns for each device. A possible explanation is that increasing the N value generates clock waveforms that help provide a nearly adiabatic solution.

To investigate the reason for the results discrepancies, a thorough comparison between all elementary functions used in both tools. The analysis confirmed that all functions behave equivalently in both simulators, yet the cause for the differences in produced charge remains unknown.

5.4 Molecule encoding checker

To streamline the efficiency of the validation process, a supplementary script named `moleculeEncodingChecker` was developed. Although not directly involved in the comparison methodology, the script is particularly useful for verifying the correct charge distribution of a specific molecule in a circuit layout, thereby determining the encoding state of the corresponding cell without relying on the graphical representation generated by the FCNviewer. Its pseudocode and complete code are provided in Algorithm 7 and Appendix A.5, respectively. It significantly reduced the effort required for verifying the functionality of the more complex circuits described in Section 2.3 and Chapter 8. The script can be executed from the folder containing all the `qss` generated by ToPoliNano. The procedure receives the list of the x and y coordinates of the molecules whose encoding state has to be checked, along with the digital values the cell is required to be consistent with. The script can be called from the simulation folder of the circuit to be validated using the following command:

```
./moleculeEncodingChecker.sh qssNumber xCoord yCoord digitalValues
```

where:

- `qssNumber`: The number of the `qss` file identifying the simulation step to be checked.
- `xCoord`: A list of the x-coordinates of the molecules to be checked.
- `yCoord`: A list of the y-coordinates of the molecules to be checked.
- `digitalValues`: A list of the digital values (‘0’ or ‘1’) that the cells are expected to be consistent with.

For example, the procedure can be called to check the encoding state of the molecules at coordinates $(32, 0)$ and $(32, 13)$ in a given circuit at the 90th simulation step, with expected digital values of ‘0’ and ‘1’, respectively, using the following command:

```
./moleculeEncodingChecker.sh 90 "32 32" "0 13" "0 1"
```

The script scans the current directory to retrieve the `qll` layout description. Then, it searches for the corresponding molecule IDs based on the provided coordinates and extracts the aggregate charge of the first dot. The value is compared with predetermined thresholds of 0.8 for a digital '0' and 0.3 for a digital '1' to determine whether the encoding state of the molecule matches the expected value. These thresholds were determined by statistically inspecting the aggregated charge values corresponding to clear 1 or 0. Finally, the script prints the result of the check. An instance of successful output is:

```
qss file to be validated: 0090.qss
Molecule 65a to be validated for a digital 1.
Molecule 93a to be validated for a digital 0.

Molecule 65a correctly encodes a digital 1.
Molecule 93a correctly encodes a digital 0.
```

Algorithm 7 Pseudocode for validating cell encoding for a given molecule at step τ.

```
Require: Step $\tau$, molecules' coordinates to be validated, expected digital values.
Ensure: Validation of the encoding state for the given molecule.
 1: Obtain the directory path of the qll layout.
 2: Parse the layout to retrieve the coordinates of the desired molecules.
 3: Define the charge thresholds for digital '0' and '1'.
 4: for each pair of coordinates do
 5:     Retrieve the corresponding molecule IDs.
 6:     for each molecule do
 7:         Extract ACs of the first Dot1.
 8:         Compare ACs with the predetermined thresholds.
 9:         Determine whether the encoding state matches the expectation.
10:         Print the result of the check for each molecule.
11:     end for
12: end for
```

Chapter 6

Enhancing the SCERPA algorithm

The extensive comparison of the research SCERPA version and its implementation in ToPoliNano identifies some differences between the tools. This chapter aims to describe how SCERPA was enhanced to improve it and make it more compatible with the CAD counterpart.

6.1 Dynamic damping

Section 3.4 discusses how SCERPA uses a static damping value within the range $(0,1)$ to help the self-consistent loop convergence in each simulation step τ and convergence step k expressed by Equation (3.3):

$$V_{in,i}^{k,\tau} = \xi V_{in,i}^{k-1,\tau} + (1-\xi)\, F_i\left(V_{in,1}^{k-1,\tau}, \dots, V_{in,N}^{k-1,\tau}\right) \tag{6.1}$$

One of the main issues with the current damping implementation is the need to balance between selecting a damping factor that is large enough to facilitate the convergence of the generic SCF step and avoiding choosing a value that is too large, which can increase the number of steps required for the procedure to converge. Moreover, a single damping value applied uniformly throughout every step of the algorithm, regardless of convergence level, may hinder the ability to produce a final solution.

To address these issues, a new dynamic damping approach that considers the history of voltage variations of molecules, leading to a potentially more efficient procedure, was derived. For a given simulation step τ, the voltage variation in the current step $V_{in,i}^{k}$ is compared to that in the previous step $V_{in,i}^{k-1}$ using the ratio of the two. Due to the unavailability of an earlier voltage variation in the first convergence step k of each simulation step τ, a static damping value is employed. This value can either be specified or left as a default one. Additionally, the first voltage variation becomes the previous voltage variation for the next step, and the same applies to subsequent SCF steps, where the current voltage variation becomes the previous for the next step. In general, a ratio smaller than 1 ensures convergence:

$$\left(\frac{V_{in,i}^{k}}{V_{in,i}^{k-1}}\right) < 1 \tag{6.2}$$

Previously, a static damping was employed to reduce the voltage variation produced by each molecule and improve convergence. In this new approach, a parameter μ is introduced to reduce the reported ratio:

$$\mu\left(\frac{V_{in,i}^{k}}{V_{in,i}^{k-1}}\right) < 1 \tag{6.3}$$

As a result, μ can be obtained by:

$$\mu = \left(\frac{V_{in,i}^{k-1}}{V_{in,i}^{k}}\right) \tag{6.4}$$

Equation (6.4) generates a vector of future possible damping values based on the number of molecules in the system. To assure consistency among all molecules, a single damping value is required. Therefore, the minimum absolute value of the damping vector is used to achieve a single positive candidate damping value. The associated molecule is the one farthest from reaching convergence consequently, the most critical element in the system.

Equation (6.4) becomes:

$$\mu = min\left|\left(\frac{V_{in,i}^{k-1}}{V_{in,i}^{k}}\right)\right| \tag{6.5}$$

To ensure that the damping value remains at an appropriate level, which would prevent convergence performance from decreasing, a precision parameter named Σ is introduced. The precision parameter Σ ranges from 0 to 1, where a value of 1 indicates maximum simulation precision but a higher risk of convergence failure, and a value of 0 indicates minimum simulation precision and an inability to progress with the procedure.

If μ is greater or equal to the precision, the final damping ξ is set to the precision value to ensure that the convergence process is not sped up excessively, especially for solutions with a slow convergence. On the other hand, if μ is less than Σ, the final damping ξ is computed as the product of μ and Σ. This aspect ensures that the voltage variations are dumped to accelerate the convergence process and improve overall performance.

Therefore, Equation 6.4 becomes:

$$\xi = \begin{cases} \Sigma & \text{if } \mu \geq \Sigma \\ \mu \cdot \Sigma & \text{otherwise} \end{cases} \tag{6.6}$$

The earlier discussed dynamic damping was integrated into the SCERPA main self-consistent loop. To enable this feature, the `importSettings.m` file was modified was showed in Listing 6.1. The added lines specify the default damping mode as static. In the case of autodamping, default values for the first convergence step damping and precision are set if the user does not deliberately set them. The complete code is provided in Appendix B.1.

```
% 0 static damping; 1 dynamic damping
scerpaSettings.autodamping = setDefault(userSettings,'autodamping', 0);
if scerpaSettings.autodamping == 1
    scerpaSettings.autodampingPrecision =
    setDefault(userSettings, 'autodampingPrecision', 0.99);
    scerpaSettings.autodampingFirstStep =
    setDefault(userSettings, 'autodampingFirstStep', 0.4);
end
```

Listing 6.1. Added source code in `importSettings()` function to enable dynamic damping.

6.1.1 Assessing the impact of dynamic damping performance

Increasing the precision parameter is theoretically expected to improve both the total number of convergence steps and the global simulation time, with the best performance achieved when $\Sigma = 1$. On the contrary, a $\Sigma = 0$ should correspond to the worse solution or better to a non-solution since the convergence is unreachable.

Figures 6.1 and 6.2 offer a visual representation of how the inverter and majority voter circuits perform under dynamic damping with varying precision parameters. In particular, four key metrics are reported as the precision parameter increases: total simulation time, mean SCF step, maximum SCF step, and median SCF step. To help comparisons, red dotted lines are included to provide a reference point and show the performance of the default damping value of 0.4.

The simulation results demonstrate that higher precision values lead to better performance, with the best results achieved when $\Sigma = 1$. Conversely, the simulations using $\Sigma = 0$. correctly prevented the circuit simulator from generating a solution since convergence cannot be reached.

To further examine the impact of precision on simulation time, Figures 6.3 and 6.4 focus on the percentage improvement and worsening of simulation time compared to the static damping reference point. The figures clearly show that positive increments in performance begin at a precision of around 0.55 for the inverter and about 0.45 for the majority voter. The maximum improvement achieved is 36% for the inverter and 40.8% for the majority voter, with a mean improvement of 19.08% and 21.1%, respectively, considering only positive increments.

6.2 Charge convergence

SCERPA's ability to generate a solution at each time step τ, is based on converting Equation (3.1) into Equation (3.2), as explained in Section 3.2. During each self-consistent field (SCF) step, the voltage variation of the specific *Molecule i* is compared with the

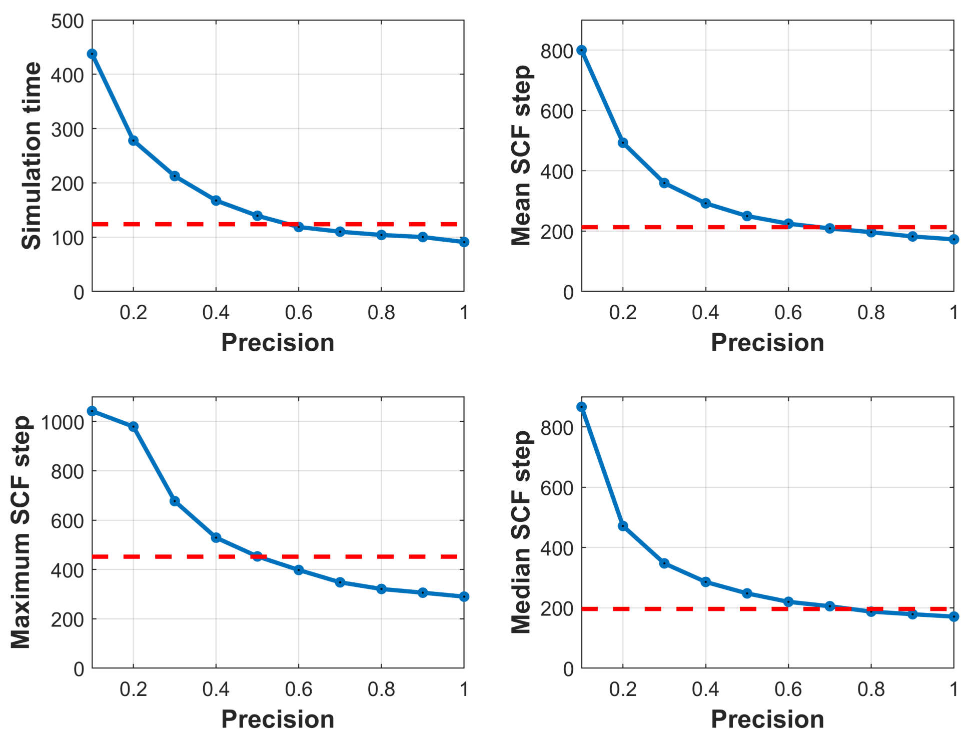


Figure 6.1. Dynamic damping analysis for the inverter gate.

stability threshold in the previous SCF step. If the difference is less than the threshold, the SCF has achieved convergence, and the time step ends. Conversely, if the difference is higher than the threshold, the procedure proceeds until convergence is accomplished. By utilizing this approach, SCERPA ensures that it generates an accurate solution at each time step while maintaining computational efficiency.

In contrast, as reported in Section 4.2, ToPoliNano verifies the stability of each step based on the variation between the charges of two consecutive steps. To ensure compatibility between the two simulators, SCERPA is enriched with the charge convergence mode of ToPoliNano. The integration starts by introducing an additional convergence setting in the `importSettings.m` file, as shown in Listing 6.2. The new line sets the default convergence mode to be the voltage mode unless differently specified.

```
% 0 voltage convergence; 1 charge convergence
scerpaSettings.convergence_mode =
    setDefault(userSettings, 'convergence_mode', 0);
```

Listing 6.2. Enabling the new charge convergence mode in `importSettings.m`

The new charge convergence mode involved modifying the self-consistent loop. In the first SCF, the aggregated charges of the molecules are stored as the previous charge state. In subsequent steps, when the molecules retrieve their new state, their ACs are

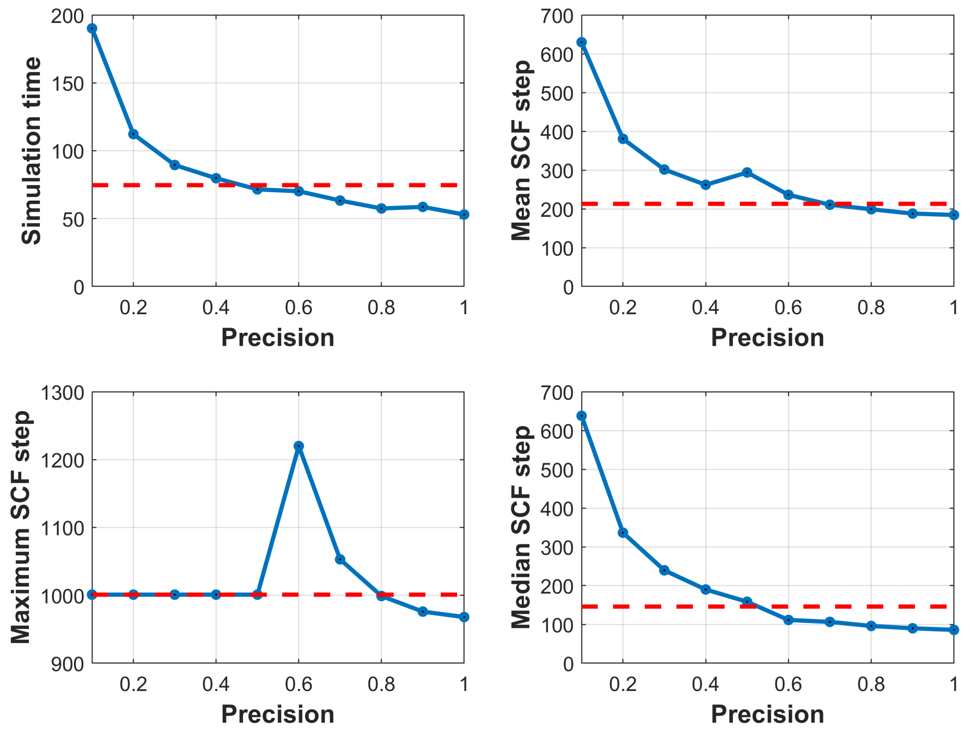


Figure 6.2. Dynamic damping analysis for the majority voter gate.

saved as the current charge state. Convergence is verified by computing the difference between the molecule's current and previous charge states. If it exceeds the stability threshold, the step is not stable yet, and refinement is required. The current charge distribution of all molecules is saved as the new previous state for the next iteration. This process is repeated until all molecules in every simulation step reach convergence.

The two modes were compared for several circuits, and in all cases, the difference in the final charge distributions was negligible, typically in the order of 0.001. Therefore, they remain broadly equivalent in achieving stable solutions. The complete code is provided in Appendix B.2.

6.3 ToPoliNano clock waveforms

The `circuit.stack_phase(i,:)` structure is used by SCERPA to specify the clock phases for simulating the circuit. The clock phases can be defined by either manually setting the clock values for each step or using the `linspace` MATLAB method.

An example of how SCERPA defines the clock phases in MATLAB is:

```
%clock values definitions
clock_low = -2;
```

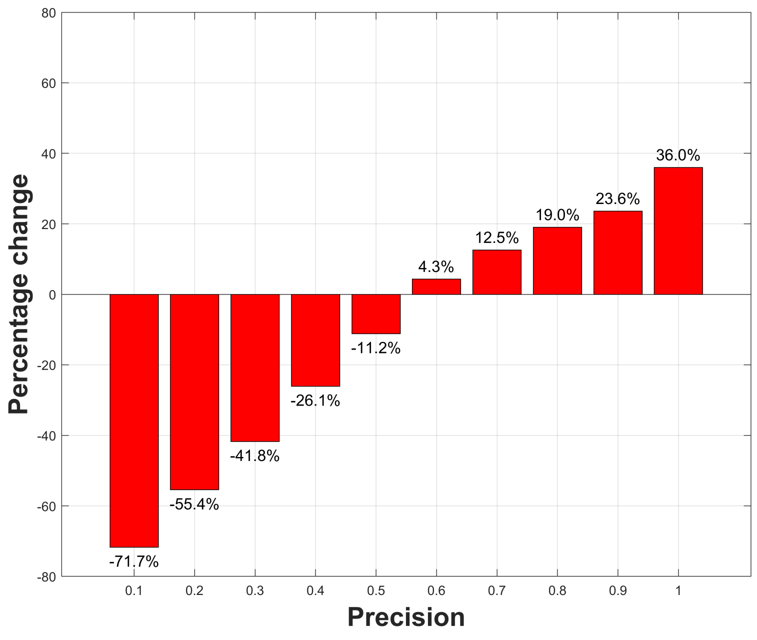


Figure 6.3. The effects of dynamic damping on simulation time for the inverter gate compared to static damping.

```
clock_high = +2;
clock_step = 3;

%clock phases definitions
pSwitch = linspace(clock_low,clock_high,clock_step);
pHold =  linspace(clock_high,clock_high,clock_step);
pRelease = linspace(clock_high,clock_low,clock_step);
pReset =  linspace(clock_low,clock_low,clock_step);

pCycle = [pSwitch pHold pRelease pReset];

circuit.stack_phase(1,:) = [pCycle pReset pReset pReset];
circuit.stack_phase(2,:) = [pReset pCycle pReset pReset];
circuit.stack_phase(3,:) = [pReset pReset pCycle pReset];
circuit.stack_phase(4,:) = [pReset pReset pReset pCycle];
```

The `linspace` function generates `N` equally spaced points between the `start` and `end` parameters. Once the `pCycle` is defined, the first phase is typically obtained as `pCycle` appended with a number of `pReset` phases. Other clock zones are generated by shifting the first by one, two, or three `pReset` phases. generated

In contrast, ToPoliNano generates clock waveforms in a static pre-defined manner by using the number of simulation steps `nSteps` and the number of points `N` to interpolate each clock waveform, along with two transients identifying the switch and release phases

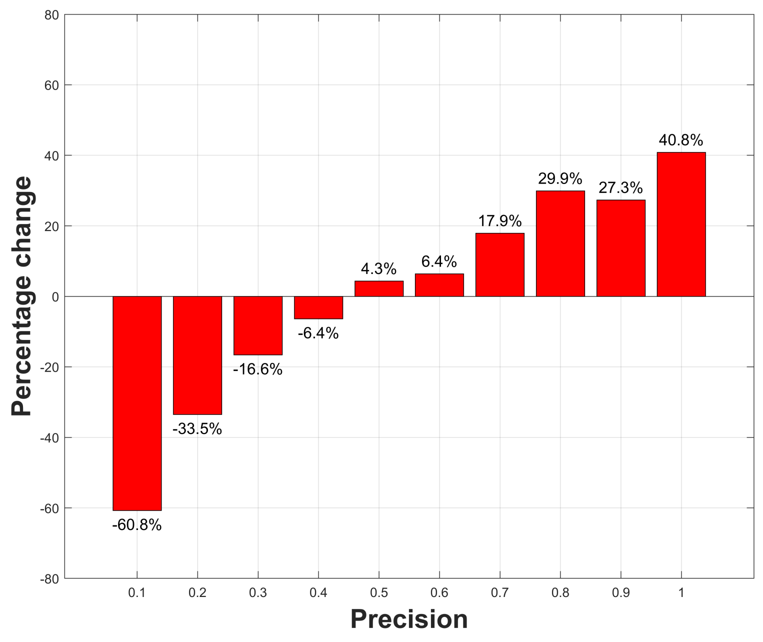


Figure 6.4. The effects of dynamic damping on simulation time for the majority voter gate compared to static damping.

and two horizontals defining the hold and reset zones. The number of points in each 1/4 clock period is equal to `N`, without considering the upper limit and the different phases are shifted by $\pi/2$. As a reference, ToPoliNano clock waveforms generated with `nSteps = 20` and `N = 4` are shown in Figure 6.5.

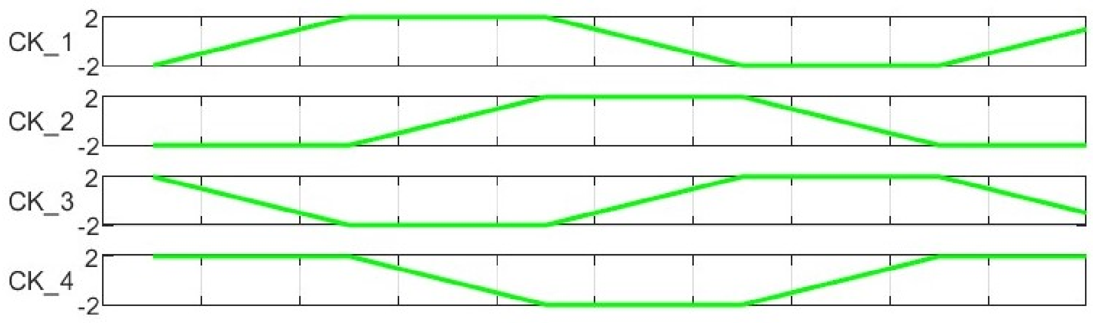


Figure 6.5. ToPoliNano clock waveforms generated with `nSteps = 20` and `N = 4`.

To create clock waveforms that match ToPoliNano's, given the total number of simulation steps `n_steps_topo` and a number of interpolating points `N_topo`, the `clock_step` variable in SCERPA is set to `N_topo + 1`, while the switch, hold, release and reset phases are obtained as before:

```
clock_step = N_topo + 1;

pSwitch = linspace(clock_low,clock_high,clock_step);
pHold = linspace(clock_high,clock_high,clock_step);
pRelease = linspace(clock_high,clock_low,clock_step);
pReset = linspace(clock_low,clock_low,clock_step);
```

To obtain the elementary ToPoliNano `pCycle`, the length of the two horizontal phases are reduced by removing their first and last points. Then, the four phases are concatenated to create `pCycle` as before:

```
    pHold = pHold(2:length(pHold)-1);
    pReset = pReset(2:length(pReset)-1);

    pCycle = [pSwitch pHold pRelease pReset];
```

The four elementary clock phases are generated based on the `pCycle` phase defined earlier. The first waveform coincides with `pCycle`, while the others are `pCycle` circularly shifted depending on the phase number and `N_topo`:

```
    topo_pCycle_1 = pCycle;
    topo_pCycle_2 = circshift(pCycle, N_topo);
    topo_pCycle_3 = circshift(pCycle, N_topo*2);
    topo_pCycle_4 = circshift(pCycle, N_topo*3);
```

To generate a complete waveform with a size of `n_steps_topo`, the number of repetitions of the elementary cycle `pCycle` is determined by dividing the total number of simulation steps by the length of `pCycle` and rounding up to the nearest integer:

```
    seq_length = length(pCycle);
    num_repeats = ceil(n_steps_topo/seq_length);
```

Finally, the four clock waveforms are stacked together and resized to `num_repeats` previously computed. Each clock matrix row corresponds to each clock waveform generated earlier:

```
    phase_i = repmat(topo_pCycle_i, 1, num_repeats);
    phase_i = phase_i(1:n_steps_topo);

    circuit.stack_phase(i,:) = phase_i;
```

Using the parameters `n_steps_topo = 20` and `N_topo = 4`, the SCERPA code presented before can generate clock waveforms that match the ones produced by ToPoliNano in Figure 6.5.

The complete code is provided in Appendix B.3.

Chapter 7

Enhancing the ToPoliNano EDA framework

The recent integration of SCERPA into the ToPoliNano framework is a significant step toward bridging the gap between research and EDA scenario, creating a promising candidate for commercial Beyond-CMOS tool in the future.

By leveraging the SCERPA's deep understanding of physical and technological details, the framework can design, simulate, and verify complex molecular architectures, employing a self-consistent iterative loop to solve electrostatic interactions without computationally expensive ab initio methods.

Although the integration of SCERPA has improved the molecular framework's capabilities, the ToPoliNano suite lacks support for features that could be useful for an effective EDA platform. Therefore, the current chapter describes the enhancements introduced to the ToPoliNano framework to bridge the gap between academic research and real-world EDA applications.

7.1 Parallelizing ToPoliNano simulator

Within the ToPoliNano simulator, the most computationally demanding task is the evaluation of molecular interactions that occur within a circuit. As the number of molecular cells grows, it rapidly becomes a significant limitation in the program's usability.

To address the issue, parallelization is a great way to improve performance by dividing the highly demanding task into smaller subtasks that can be executed simultaneously on multi-core systems. Qt libraries provide strong support for parallelization by offering many high-level APIs, like `QtConcurrent`. This framework enables several advantages, including the abstraction of many low-level details of parallel programming, such as synchronization, thread management, and data sharing.

In the ToPoliNano scenario, the most appropriate function to concurrently handle all the molecular interaction computations is the `QtConcurrent::map`. The routine simultaneously applies a function to each element of a collection according to the system specifications gathered using platform-specific APIs. In particular:

- It receives the input data to be handled in parallel, usually as an iterable collection, and the mapping function is responsible for the data processing.
- It splits the input data into smaller chunks and associates them to the available threads along with the mapping function.
- It returns the processed data as a `QFuture` object.
- It is often used in conjunction with the `waitForFinished()` method to coordinate with remaining data processing ensuring a sequential execution flow.

The code in Listing 7.1 demonstrates how the `evaluateInteraction()` method is parallelized. The variable `node_it` contains all the graph vertices representing every molecular interaction in the circuit. It is used as an iterator by `QtConcurrent::map` along with the `molecularEvaluator` working as a mapping function to compute the new molecule state. The code of the `molecularEvaluator` is reported in Appendix C.2. The processed interaction is stored in the `QFuture` object `future`. The method `waitForFinished()` on the `future` object blocks the new calling thread until all the dependencies of the given interaction are solved.

```
void MolecularSimulationController::evaluateInteractionParallel() {
    auto node_it = boost::vertices(system_.graph());
    QFuture<void> future = QtConcurrent::map(
        node_it.first,
        node_it.second,
        molecularEvaluator(
            &system_,
            simulation_time_,
            simulation_step_,
            &molecules,
            damping_factor_
        )
    );
    future.waitForFinished();
}
```

Listing 7.1. Source code of `evaluateInteractionParallel()` function.

7.1.1 Assessing the impact of parallelization

To investigate the advantages of parallelizing the ToPoliNano simulator, molecular circuits described in Section 2.3 were simulated using single-threaded and multi-threaded applications. The resulting simulation times in nanoseconds are shown in Figure 7.1, utilizing a logarithmic scale for more accurate visualization across differences. All simulations employed identical parameters, including simulation steps (100) interpolating points in each clock phase ($N = 5$), interaction radius ($d_{IR} = 10.1\text{e}{-9}$), and stability threshold ($\varepsilon_{max} = 5\text{e}{-6}$) and were carried out on an Intel Core i5-4690 processor running at 3.90 GHz with 8GB of RAM.

As expected, the results show a substantial enhancement in overall performance with reduced simulation times across all devices.

Figure 7.2 provides a more detailed analysis of the improvement percentages across all the devices. The best improvement is observed for the three-phase wire at 70.97%, while the two-line L-connection shows the worse enhancement at 33.66%. On average, the progress is around 54.82%. The reason why some devices are more prone to benefit from parallelization remains unclear.

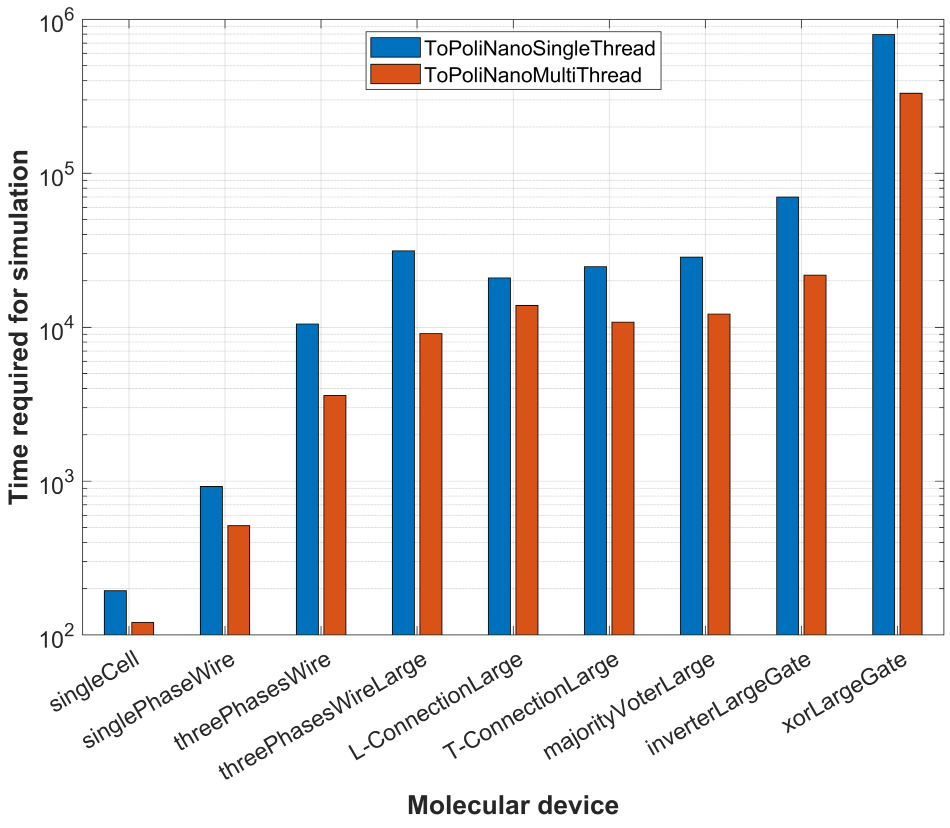


Figure 7.1. Multi-threaded and single-thread simulation times for a set mFCN circuits.

7.2 Introducing double-driver

In the MATLAB version of SCERPA, the molecular cell of the driver can be configured in two different ways: either as a single molecule that realizes a half mQCA cell or as two adjacent molecules that form a complete mQCA cell. To introduce double-driver support in the ToPoliNano framework, modifications were made to both the simulator and the viewer.

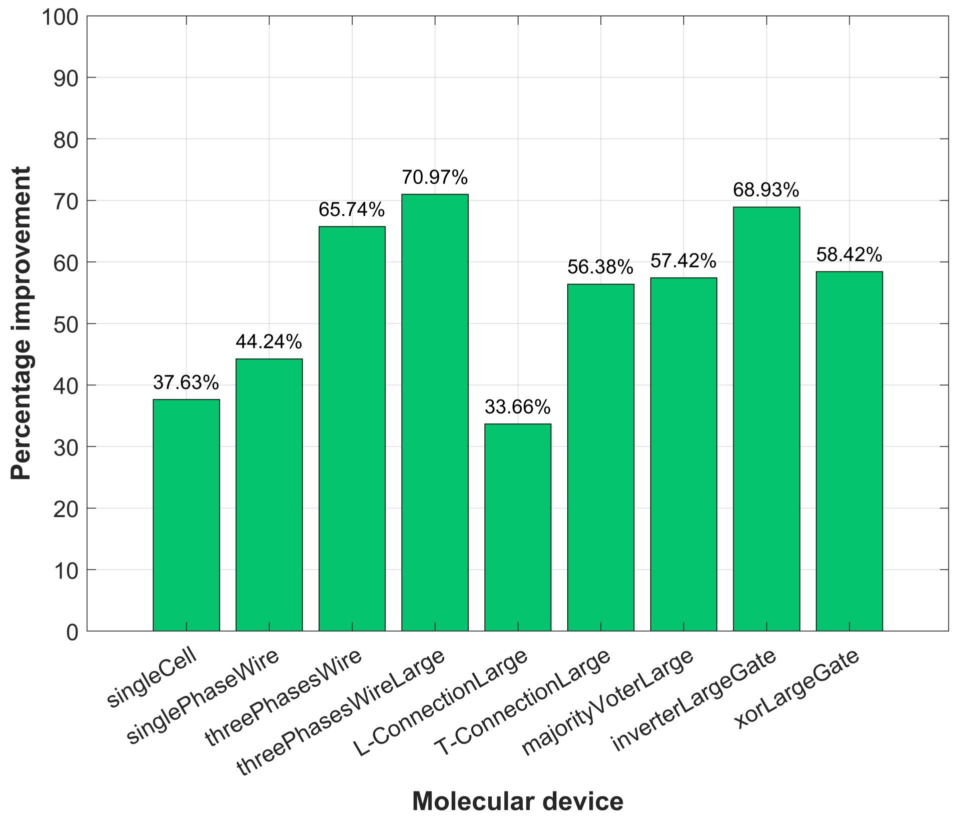


Figure 7.2. The effects of parallelization on simulation time for a set of mFCN circuits compared to single-thread.

ToPoliNano Simulator

As previously outlined in Section 4.2, the `molecularSimulationController` method performs a loop through the `qcaItems` in the imported circuit to identify whether each item is a driver or not. If a driver is detected, the method generates a single molecule. If not, the controller generates two molecules and appends “a” or “b” at the end of their IDs.

Consequently, to introduce two molecules for each driver, the controller was modified to mimic the behavior applied to non-driver cells, labeling their molecules with “a” and “b”. Throughout the simulation process, the driver charge state is updated via the `setInputValue()` method based on the digital input defined in the testbench. Currently, this is carried out for the single molecule of the driver cell only. Hence, the additionally introduced input molecule is treated as the opposite of the first. For instance, when the testbench indicates a logical ‘1’ for the input cell, a positive voltage is applied to the first molecule to retrieve an aggregated charge that mimics a state where the free charge is confined in Dot2, based on the transcharacteristics. Conversely, a negative voltage is applied to the second molecule of the same cell to move the charge toward Dot1.

The complete code is reported in Appendix C.3 and C.4.

FCNviewer

The FCNviewer was modified with the same intent. In particular, the `parseQll()` method of the `QSSReader` class is responsible for importing the `qll` circuit description and initializing the structures used by the openGL libraries to represent molecules of the circuit. When an input cell is detected, two molecules are created and positioned at the proper intermolecular distance, similar to non-driver cells. Figure 7.3 displays a comparison between a majority voter that uses single and double drivers.

The complete code is provided in Appendix C.5.

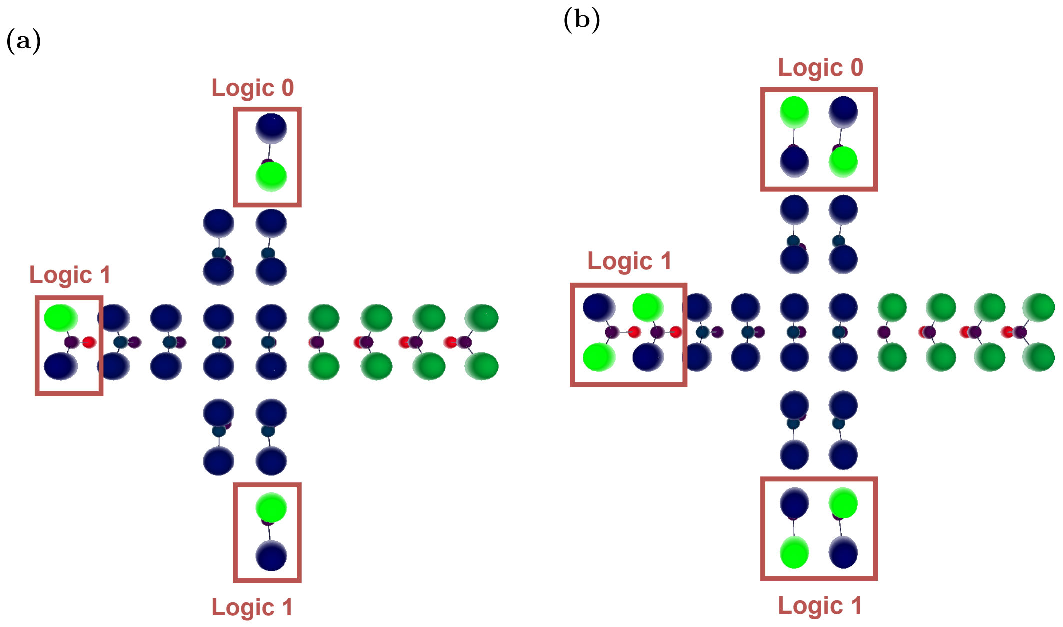


Figure 7.3. Single (a) and double-driver (b) representation of a majority voter.

7.3 Enabling hierarchical layouts

The basic idea behind hierarchical design of circuits is to divide complex architecture into smaller and more manageable parts which can be arranged in a hierarchical multilevel structure. Each level is designed and optimized independently from the others so that designers can focus on each level individually without caring about the complexity of the entire circuit. Other advantages include easier maintainability and an improved system understanding.

In MagCAD, hierarchical designs are based on the `component` entity, which can represent any set of molecular or magnetic supported items. A component creation starts with the drawing of the circuit to be converted in a black box and proceeds with its extraction in the form of a `xml` file with a `qcc` extension that contains all the data required for its use and graphical representation.

Components are organized in libraries the designer can fill as desired. A component definition always starts with the name of the library to which it belongs, its name, its dimensions, and an alphanumerical ID used to unequivocally reference it among other components. Then, it lists all the items composing the circuit layout specifying coordinates and phase numbers.

Listing 7.2 shows an example of a MagCAD component representing a molecular majority voter.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--File describing the QCA component-->
<qcacomponent Version="2" tech="MolFCN" lib="MolFCN_library" name="
    majorityVoter" ID="
    afb843f07d1747174b424fa30d8f10d0b8f7fe5ab0fedf1a5e2f54bbedb56a83"
    function="" maxX="3" maxY="2" MagnetsCount="6">
    <entity/>
    <components>
        <item tech="MolFCN" name="Bisferrocene"/>
    </components>
    <layout>
        <item comp="0" x="0" y="1" layer="0">
            <property name="phase" value="0"/>
        </item>
        <item comp="0" x="1" y="0" layer="0">
            <property name="phase" value="0"/>
        </item>
        <item comp="0" x="1" y="2" layer="0">
            <property name="phase" value="0"/>
        </item>
        <item comp="0" x="1" y="1" layer="0">
            <property name="phase" value="1"/>
        </item>
        <item comp="0" x="2" y="1" layer="0">
            <property name="phase" value="2"/>
        </item>
        <item comp="0" x="3" y="1" layer="0">
            <property name="phase" value="2"/>
        </item>
        <wiring/>
    </layout>
</qcacomponent>
```

Listing 7.2. A molecular majority voter described as a `qcc` component.

MagCAD currently supports hierarchical design for both molecular and magnetic circuits. Nevertheless, the ToPoliNano simulator does not support the analysis of molecular-based hierarchical circuits.

To enable this feature, ToPoliNano must first determine whether the `QcaItem` read from the `qll` is a `component` or not. If it is a component, ToPoliNano has to reference the component in the QCC library, retrieve the list of all the component sub-`QcaItems`, and proceed with the import procedure. Since a hierarchical layout can have an arbitrary number of nested levels, the optimal approach is to use a recursive function to explore the entire design and flatten it, allowing for every elementary molecule of the circuit to be included in the simulation. Therefore, the `exploreHierarchicalLayout()` function is added to the `molecularController`. It receives:

- The `QcaItem` list of the identified component, obtained as `component->items()`.
- The spatial coordinates of the component.
- The ID of the component, created by concatenating its spatial coordinates in the format x.y.z.
- The settings required to carry on with the import.

The structure of the `exploreHierarchicalLayout()` function is similar to that of the `MolecularController`, as it loops through all the `QcaItems` of the circuit and calls itself recursively for every nested component found. For each recursion level, the position of each item is adjusted with the offset of the component coordinates to determine each molecule's location in the layout.

In addition, to facilitate identifying a specific molecule in the output simulation results, the cell ID was modified from the one described in Section 4.2. Specifically, MagCAD no longer defines it according to the ordering of placement, but instead uses the cell coordinates in the layout in the form x.y.z.

As a result, for a one-level component placed at coordinates (1,0,0) and containing a cell at coordinates (1,1,0), the resulting molecule IDs would be:

```
1.0.0_1.1.0a
1.0.0_1.1.0b
```

The complete code is reported in Appendix C.6 and C.7.

The introduced hierarchical simulation support was tested using circuits in several conditions, ranging from single-level layouts to multiple layouts. Hierarchical designs offer particular benefits for complex systems, which can be broken down into smaller, more manageable components.

As a reference, the molecular XOR gate, shown in Figure 2.23, was realized using a one-level hierarchical approach, according to the mFCN inverter and majority voter fundamental blocks. The result is shown in Figure 7.4, highlighting the perspective offered by hierarchical design, with improvements in the readability and cleanliness of the layout.

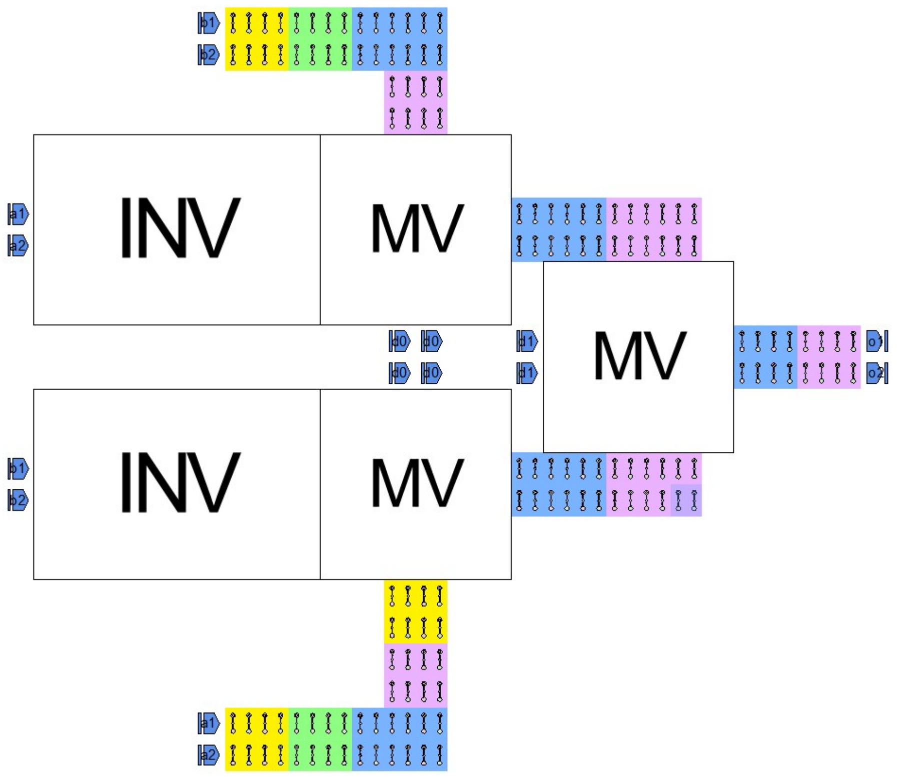


Figure 7.4. One-level hierarchical design applied to the XOR gate.

7.4 Toward information crossing

In molecular FCN, the interconnections play a crucial role in propagating information and enabling computations within the circuit. These interconnections facilitate communication among cells within the same layout and connections with other components, realizing complete and advanced devices. In molecular FCN interconnections are typically achieved using wires implemented using L- and T-connections.

To explore new opportunities for improving the paradigm, recent research [1] proposed to realize information crossing by taking advantage of the diagonal interaction in the traditional QCA paradigm. Building on this result, the authors in [5] investigated the possibility of transposing the same diagonal interaction to molecular FCN devices. The proposed layout, as shown in Figure 7.6, includes a vertical and horizontal wire that crosses at a shared cell called an evaluation cell. This intersection allows for information crossing and enables the propagation of the two driver values to their respective output cells. A correct layout functioning requires particular neutral molecules, therefore the authors introduced a neutral version of the commonly used bis-ferrocene presented in Section 2.1.3.

The associated molecular QCA cell consists of two juxtaposed neutral bis-ferrocene with positive and negative charges placed in the logical dots as reported in Figure 7.5.

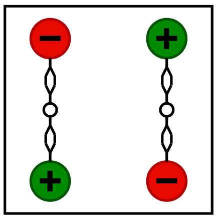


Figure 7.5. Neutralized bis-ferrocene molecular QCA cell.

The working principle of the *evaluation cell* depends on whether the two driver cells encode the same data or not. If they do, the cell propagates the information of the two branches to the outputs. If they do not, the cell remains non-polarized and does not affect the output branches. In the latter case, information crossing is achieved by exploiting the cells' neutrality and the fact that the horizontal output branch is primarily influenced by the vertical input branch due to their proximity and vice versa. As a result, the horizontal output cell is pushed to encode the logical value opposite to the vertical one. Moreover, to minimize the effect of the evaluation cell on the output branches when the logic values are opposite, the cell is subjected to a reduced clock field [5].

Figure 7.6 and Table 7.1 show the molecular layout and the truth table of the crosswire circuit proposed in [5], respectively.

A	**B**	**Out1**	**Out2**
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

Table 7.1. Crosswire truth table.

7.4.1 The new molecule in the ToPoliNano EDA framework

To introduce the neutralized oxidized bis-ferrocene molecule into the ToPoliNano simulator, its name, and x-y-z spatial coordinates were added to the `defineChargePosition()` method of the `MolecularElement` class. This method sets the position of each dot whenever the `molecularController()` function creates a new molecule.

Once the molecule is defined, the tool imports the correspondent transcharacteristics. As explained in Section 4.2, ToPoliNano requires the transcharacteristics to be interpolated on a predetermined number of voltage values. Therefore, the SCERPA reshape function converts the transcharacteristics obtained from ab initio simulations into the required format. Figure 7.7 shows the transcharacteristics of the molecule under two clock conditions.

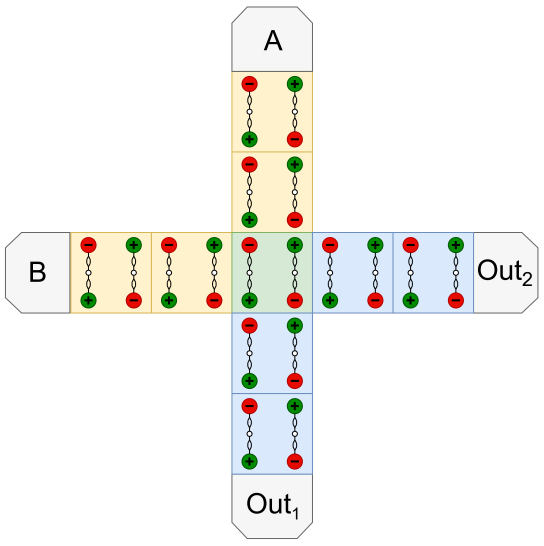


Figure 7.6. mFCN crosswire layout made of neutral oxidized bis-ferrocene molecules.

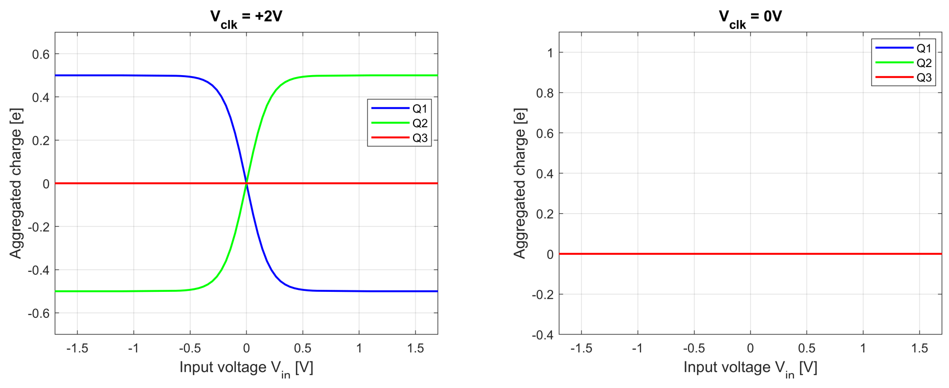


Figure 7.7. V_{in}–aggregated charge transcharacteristics of the neutralized oxidized bis-ferrocene molecule subjected to two clock fields.

The molecular controller retrieves molecules differently depending on whether they belong to a driver cell. For non-driver cells, the `item->name()` method returns the name of the molecule defined in the `qll` circuit, which is later used to determine the number of dots and select the appropriate transcharacteristic.

Conversely, ToPoliNano currently has a hardcoded bis-ferrocene molecule describing driver cells. To allow the use of the new molecule, a new variable `driverMolecularName` was introduced and treated in the same manner as other simulation parameters. This variable stores the molecule name used for driver cells during initialization by the ToPoliNano controller.

Integrating the new molecule into the GUI

To associate the `driverMolecularName` variable with a specific molecule chosen by the user, the GUI of the ToPoliNano simulation window was expanded. A new section was added to the window, featuring a drop-down list of available molecule options.

In particular, a new widget named “Driver settings” was created and attached to the molecular analysis window. It is implemented as a `QGroupBox` and contains a `QComboBox` drop-down list that displays the available driver molecules for selection, with the default option set to "Bisferrocene".

Listings 7.3 and 7.4 shows the source code of the `createDriverSettingsBox()` and `addComboDriverMoleculeList()` which generates the settings box and the drop-down list, respectively. Figure 7.8 displays the new ToPoliNano molecular GUI with the new driver option.

```
void createDriverSettingsBox() {
    m_GroupDriverSettings.setTitle("Driver settings");
    m_driverMoleculeLabel.setText("Driver molecule name");

    addComboDriverMoleculeList(m_ComboDriverMoleculeList);

    m_GridLayoutDriverSettings.addWidget(&m_driverMoleculeLabel, 1, 0);
    m_GridLayoutDriverSettings.addWidget(&m_ComboDriverMoleculeList, 1,
    1);

    m_GroupDriverSettings.setLayout(&m_GridLayoutDriverSettings);
}
```

Listing 7.3. Source code to create the driver settings box

```
void addComboDriverMoleculeList(QComboBox &box) {
    box.addItem("Bisferrocene");
    box.addItem("NeutralizedOxBisferrocene");

    box.setCurrentIndex(0); // Default is Bisferrocene
}
```

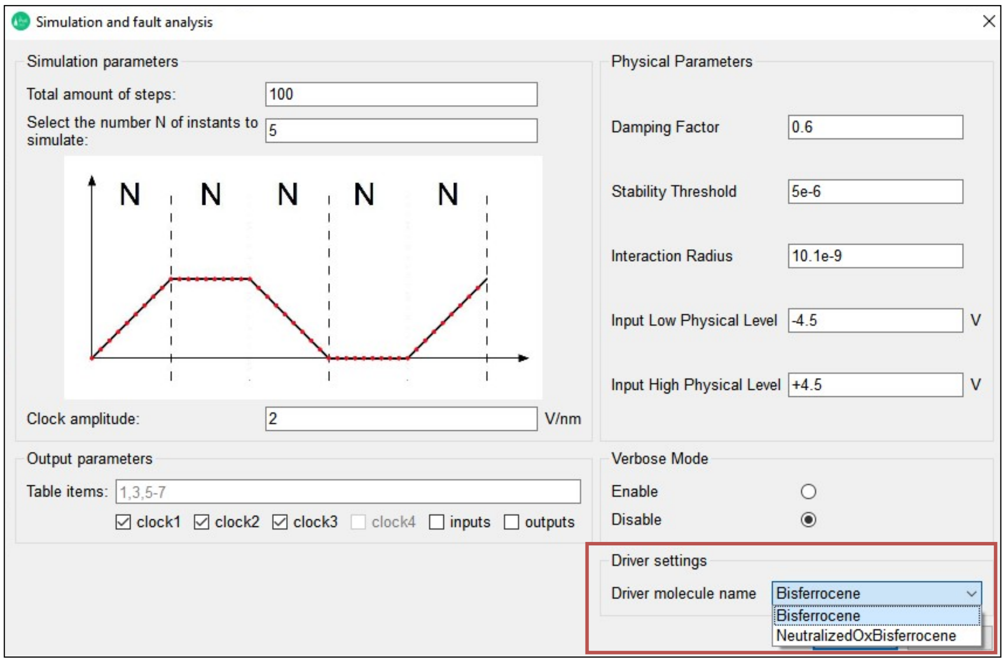
Listing 7.4. Source code to realize the drop-down molecules list

7.4.2 Simulating the crosswire with ToPoliNano

The authors in [5] devised a method to reduce the clock applied to the evaluation cell to reduce its influence on the output branches when two opposite inputs are propagated.

However, ToPoliNano currently does not allow such precise customization of clock waveforms, making it impossible to achieve correct information crossing in this case. As a workaround, two layouts were developed: one for the same inputs and another for different inputs, as illustrated in Figure 7.9a and Figure 7.9b, respectively.

When both drivers are equal, the input branches are synchronized by the same clock phase, providing the evaluation cell with data to be evaluated on the next clock phase. The output branches share the same clock to forward propagate data crossing.


Simulation and fault analysis

Simulation parameters

Total amount of steps: 100

Select the number N of instants to simulate: 5

Clock amplitude: 2 V/nm

Output parameters

Table items: 1,3,5-7

✓ clock1 ✓ clock2 ✓ clock3 ☐ clock4 ☐ inputs ☐ outputs

Physical Parameters

Damping Factor 0.6

Stability Threshold 5e-6

Interaction Radius 10.1e-9

Input Low Physical Level -4.5 V

Input High Physical Level +4.5 V

Verbose Mode

Enable ○

Disable ●

Driver settings

Driver molecule name Bisferrocene

Bisferrocene

NeutralizedOxBisferrocene

Figure 7.8. The new ToPoliNano molecular GUI with the driver molecule settings.

In contrast, when inputs are opposite, the input and output branches are synchronized with the same clock phase, allowing the last cell of one input branch to influence the first cell of the opposite branch and vice versa. Meantime, the evaluation cell is subjected to the next clock zone to not interfere with this interaction and enable the information crossing.

To assess layouts' functionality, simulations were performed using ToPoliNano. As a reference, Figure 7.10a shows the successful propagation of drivers encoding logical '0' to the outputs after the crossing, while Figure 7.10b illustrates the instance where the input branches, carrying logical '1' and '0', are causing a complementary state to be imposed on the output branches due to the deactivation of the evaluation cell.

The new neutralized oxidized bis-ferrocene has successfully demonstrated a practical approach for supporting new molecules in the ToPoliNano EDA framework. Moreover, this integration paves the way for possible future exploration of hybrid designs that combine different molecules in the same circuit, leveraging their unique electrical and physical properties to enhance performance and functionality.

7.5 Consistent visualization in the FCNviewer

The FCNviewer was modified to comply with the same frame of reference as the SCERPA viewer while retaining the x-z direction inversion in MagCAD. Previously, when rotations and shifts were applied along different directions, they produced comparable charge distributions over time, but the graphical representation between the two viewers was

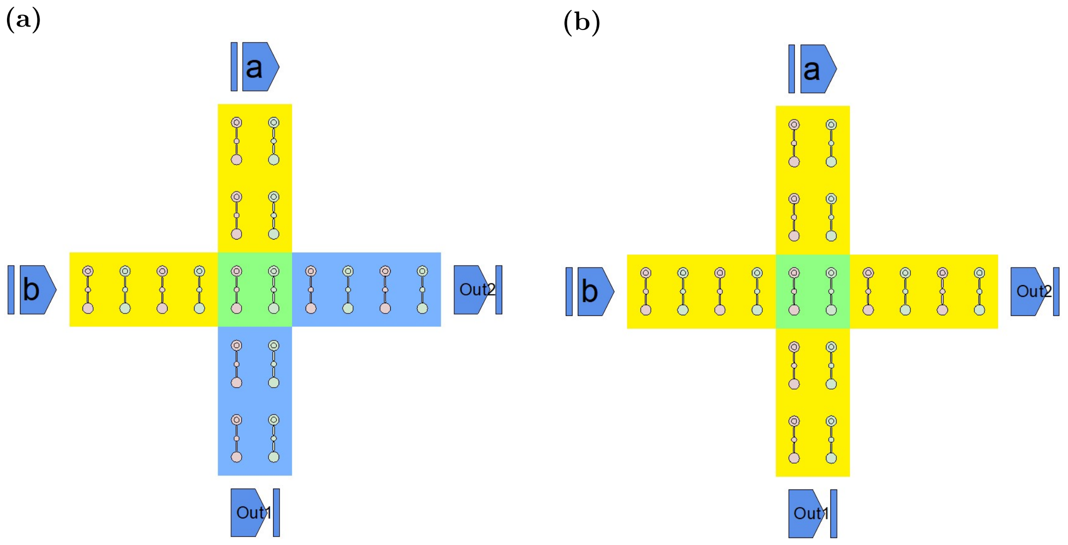


Figure 7.9. Two mFCN crosswire layouts. (a) mFCN crosswire layout for same inputs. The input branches are synchronized by the same clock phase, and the output branches share the same clock to forward propagate the data crossing. (b) mFCN crosswire layout for opposite inputs. The input and output branches are synchronized with the same clock phase, allowing the last cell of one input branch to influence the first cell of the other branch without interference from the evaluation cell.

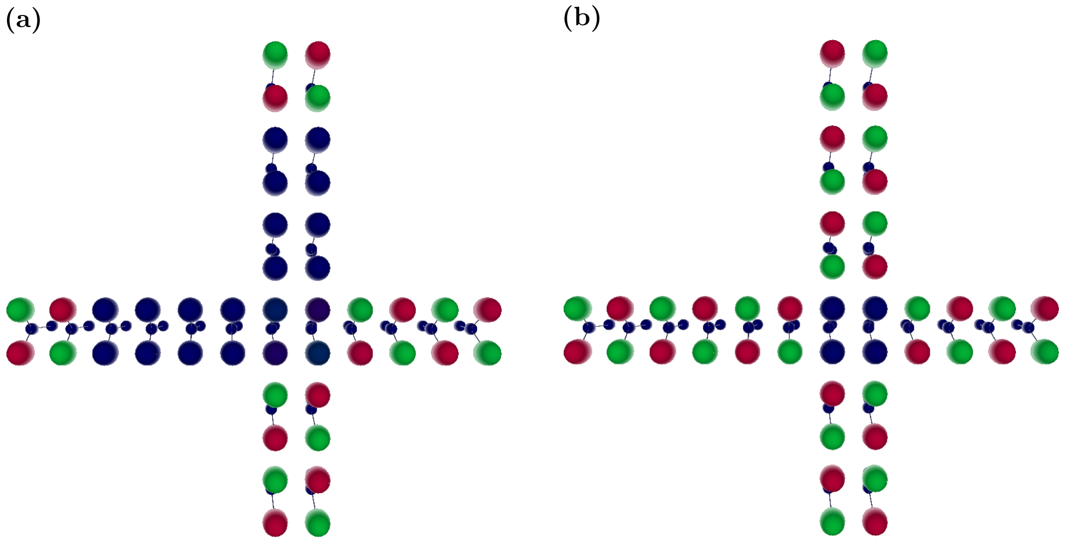


Figure 7.10. ToPoliNano simulations for the crosswire layouts. (a) Drivers encode same logic '0' state. (b) Drivers encode two opposite logic states.

not the same.

In particular, the function `getChargePosition()` was updated to use the coordinates of the ToPoliNano simulator to define the position of each molecule's dot in the circuit. The `parseQll()` function, which imports the circuit layout and identifies any shifts or rotations, was also modified to align with the conventions used in the SCERPA viewer.

Furthermore, due to the x-z axis inversion, the view of the circuit placed the Dot4 at the top and the Dot1 and Dot2 at the bottom. Therefore, the `resetRotation()` function was modified to reset the x-axis position to 180 degrees, ensuring the correct orientation. Appendix C.8 and C.9 report the complete code.

To verify the efficacy of the modifications, Figure 7.11 showcases a MagCAD layout with multiple shifts and rotations applied to a molecular wire. In particular:

- Mol_1 shifted along the x-axis by 200 pm.
- Mol_2 shifted along the x-axis by -200 pm.
- Mol_3 shifted along the z-axis by 500 pm.
- Mol_4 shifted along the z-axis by -500 pm.
- Mol_5 shifted along the y-axis by 200 pm.
- Mol_6 shifted along the y-axis by -200 pm.
- Mol_7 rotated by 30 degrees.
- Mol_8 rotated by -30 degrees.

The correspondent layout was simulated in both SCERPA and ToPoliNano. Figure 7.12 presents the SCERPA 3D graphical circuit view, while Figures 7.13 and 7.14 display the top and lateral views of the layout generated by FCNviewer.

All shifts and rotations match between the two viewers, demonstrating the effectiveness of the modifications.

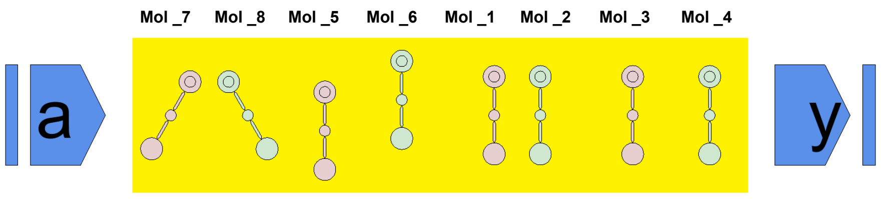


Figure 7.11. mFCN wire layout generated by MagCAD with shifts and rotations applied to molecules.

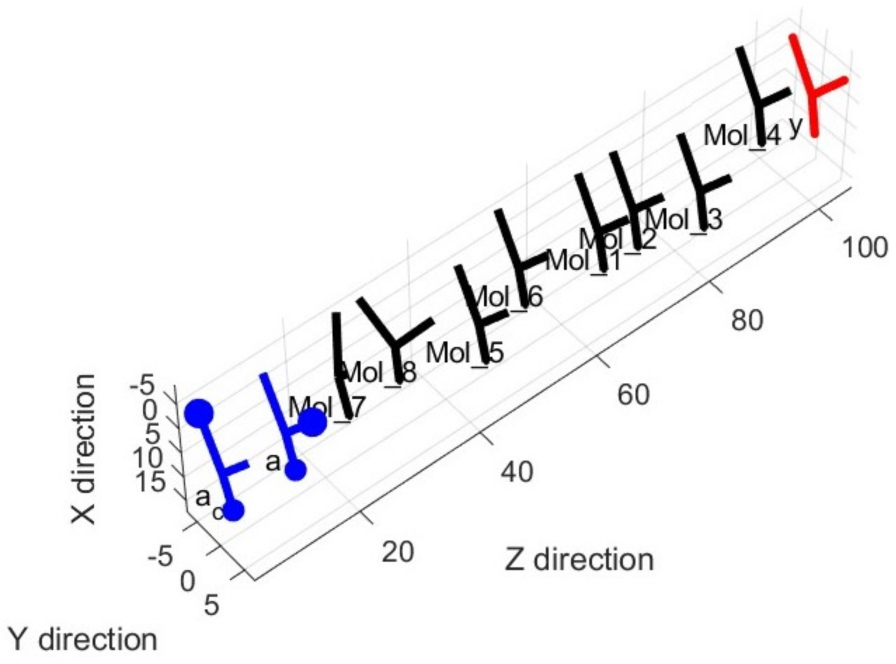


Figure 7.12. mFCN wire layout represented by the SCERPA 3D viewer with shifts and rotations applied to molecules.

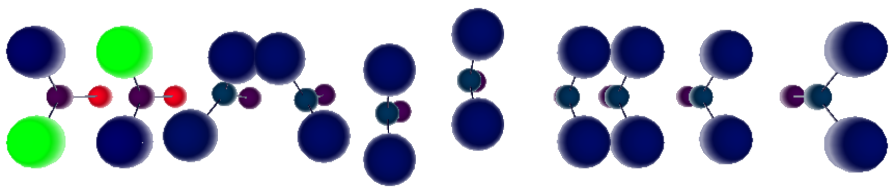


Figure 7.13. Top view of the modified mFCN wire layout generated by the FCN viewer.

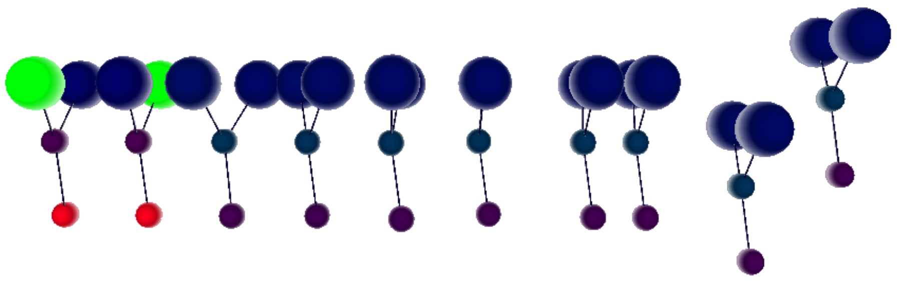


Figure 7.14. Lateral view of the modified mFCN wire layout generated by the FCN viewer.

Chapter 8

Advanced molecular FCN circuit design with the ToPoliNano EDA framework

The ToPoliNano framework has made remarkable progress in enhancing its functionality and performance. In particular, a crucial advancement has been achieved through the parallelization of molecular analysis, particularly in the computation of molecule interactions within a layout. This aspect enables the ToPoliNano suite to investigate more complex molecular circuits more rapidly and realistically, bringing it closer to practical Beyond-CMOS EDA applications.

Furthermore, the framework has been validated through extensive testing, including comparison with its research SCERPA counterpart, which gives good hope of its capability in converting digital architectures into their molecular counterparts while ensuring their correctness.

The following chapter introduce a range of molecular FCN circuits, including the 1-bit and 2-bit full adders, alongside some innovative layouts that have yet to be explored in the molecular FCN scenario. One such example is the ISCAS C17 benchmark digital circuit, which was engineered in its molecular equivalent using the MagCAD editor from scratch. The aim is to evaluate the extent of architecture complexity SCERPA can analyze.

To guarantee consistent and reliable results, all devices underwent ToPoliNano analyses with the same physical and electrical parameters, comprising clock settings, interpolating points in each clock phase ($N = 5$), interaction radius ($d_{IR} = 10.1\text{e}{-9}$), and stability threshold ($\varepsilon_{max} = 5\text{e}{-6}$). The simulations were carried out on an Intel Core i5-4690 processor running at 3.90 GHz with 8GB of RAM.

Finally, the results were validated by combining the FCNviewer and the validation script described in Section 5.4.

Figure 8.1 reports the schematic flow adopted to design, simulate and test the novel molecular architectures.

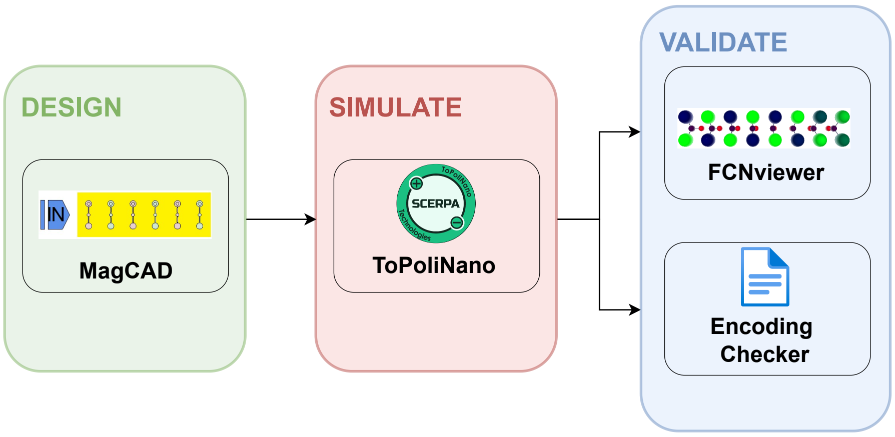


Figure 8.1. Schematic flow adopted to design, simulate and test the novel molecular architectures.

8.1 QCA adders

In digital electronics, addition is a fundamental operation performed using various methods. One of the most common and straightforward ways is through a full adder, which adds two binary numbers alongside an input carry, producing a sum and a carry-out. Table 8.1 reports the truth table of a 1-bit full adder.

A modular approach exploiting multiple 1-bit full adders can be taken to implement an N-bit full adder. Each 1-bit adder's carry-out is connected to the carry-in of the next, and the sum output of each module is combined to produce the N-bit result.

Researchers have been digging quantum-dot cellular automata (QCA) principles to design full adders. Like other molecular FCN logical devices, full adders can be realized by properly arranging the inverter gate and majority voter fundamental blocks. One of the earliest structures was presented in [60] and further explored in [64]. The structure consists of five majority gates and three inverters and resembles a carry look-ahead (CLA) structure as the carry-out signal is ready before the sum. Figure 8.2 illustrates the layout using fundamental QCA computational blocks.

More recently, [66] has demonstrated that:

$$C_{out} = MV(A, B, C_{in}) \tag{8.1}$$

$$Sum = MV(\overline{C_{out}}, C_{in}, MV(A, B, C_{in})) \tag{8.2}$$

proposing an optimized full adder layout that requires only three majority voters and two inverters according to the arrangement depicted in Figure 8.3.

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 8.1. 1-bit full adder truth table.

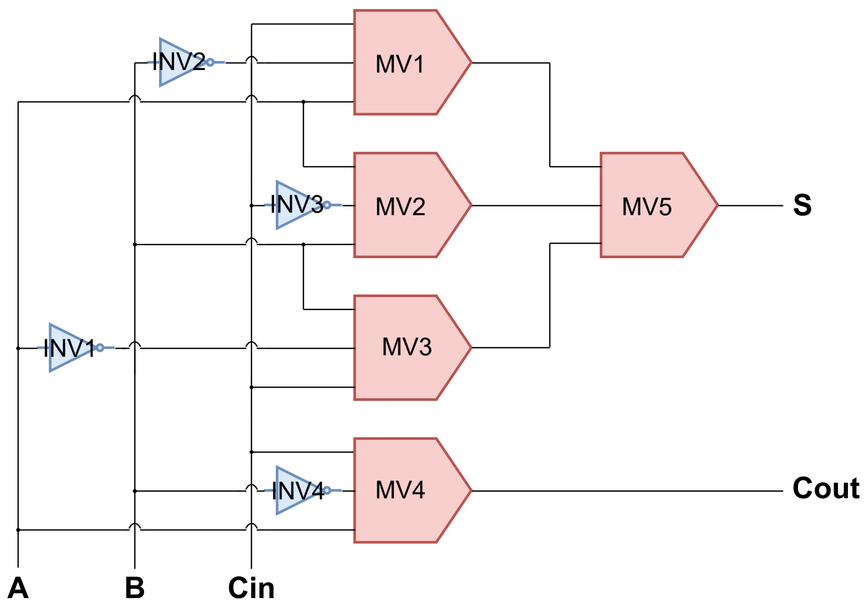


Figure 8.2. One of the earliest 1-bit full adder layouts proposed for QCA technology and implemented with a majority voter and inverter-based logic.

8.2 Molecular 1-bit full adder

The feasibility of transposing the optimized full adder layout proposed in [66] into the molecular domain has been already demonstrated in [5] by leveraging the bis-ferrocene molecule. In that context, the circuit was designed using the MagCAD graphical editor and successfully simulated with SCERPA.

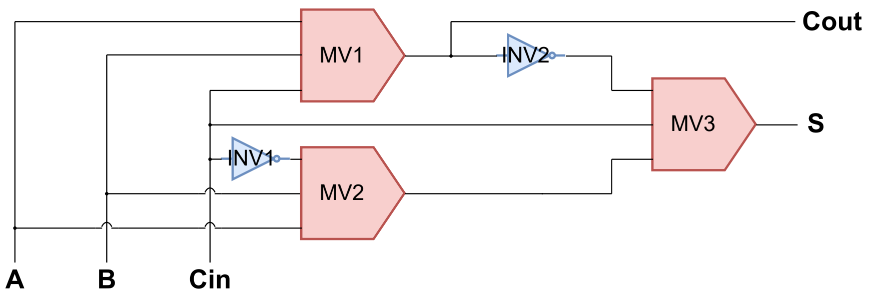


Figure 8.3. Optimized 1-bit full adder implemented with a majority voter and inverter-based logic.

8.2.1 Layout

Due to the circuit's complexity in terms of the number of cells, reliable data propagation with no border effects is ensured by designing the device with a uniform partition into the four-phase clocking system. The clock zones are maintained large enough to preserve correct transmission and avoid metastability. As a result, the structure has a four-pipeline fashion that makes the outputs available through the two critical paths propagating the carry-out and the sum, with a delay of $4T$, where T is the clock period. The design reported in Figure 8.2 follows a two-line fashion to enhance robustness against information loss and degradation. Each square in the figure corresponds to an mQCA cell, consisting of two bis-ferrocene molecules juxtaposed under the influence of one of the four-phase clocks.

Due to the lack of complete crossing information support, the layout require some of its inputs to be duplicated, increasing the total area.

8.2.2 Simulation and testing

The integration of parallel interaction computation into the CAD simulator tool provides new opportunities to investigate the enhanced efficiency of ToPoliNano. A comparison between the recent parallel version and the old single-threaded version in terms of required simulation time can provide valuable insights into the reduction of analysis time for not just full adder, but other molecular FCN devices as well.

The simulation times for the single-threaded and multi-threaded versions of ToPoliNano carried for 90 steps are as follows:

- Single-threaded: 6840 seconds.
- Multi-threaded: 2617 seconds.

It is noteworthy that the parallel version of ToPoliNano results in a significant 61.74% reduction in the required analysis time.

To ensure correctness, exhaustive testing was performed, generating a proper test-bench for each of the eight possible input combinations and verifying charge distributions on output molecules using flow reported in Figure 8.1. The layout synchronously produces carry-out and sum after 90 steps.

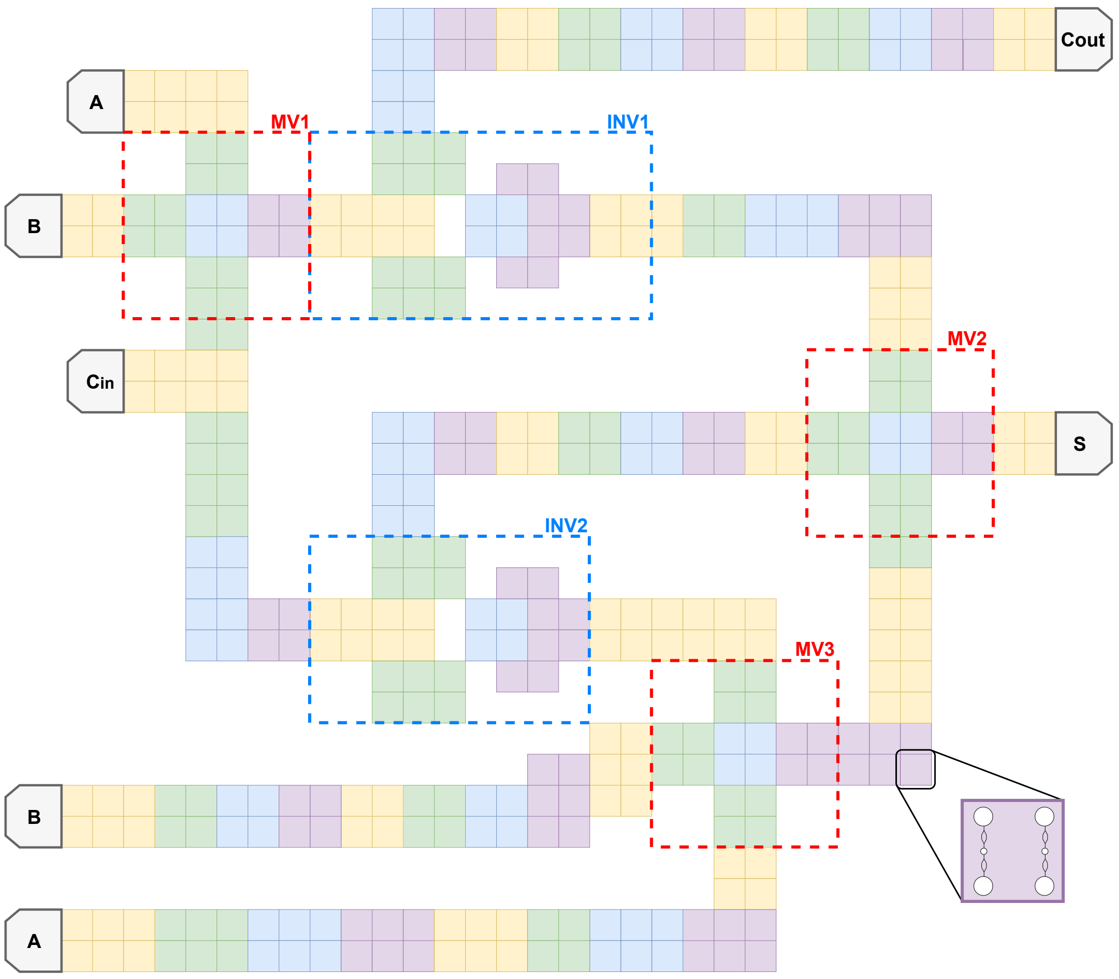


Figure 8.4. 1-bit molecular full adder implemented with a robust two-line fashion with a majority voter and inverter-based logic. Each square corresponds to an mQCA cell, consisting of two adjacent bis-ferrocene molecules.

8.3 Molecular 2-bit full adder

The 2-bit full adder operates similarly to its 1-bit counterpart, except it processes two bits for each input instead of one.

8.3.1 Layout

To construct the 2-bit version, two 1-bit full adders are connected in sequence using an L-connection, with the carry-out of the previous stage serving as the carry-in of the second as Figure 8.5 illustrates. The molecular equivalent is derived similarly by replicating two molecular 1-bit full adders joined in sequence, as shown in Figure 8.6.

The clocking system is maintained between the two stages to ensure information correctness throughout the circuit. The sum of the first two input bits is available after $4T$, while the sum of the remaining ones is ready after $10T$, where T represents the clock period. The L-connection links the carries of the two stages with a delay of $5/4T$. The critical path propagates and computes the carries, resulting in a global delay of $10T$.

8.3.2 Simulation and testing

The simulations were performed using standard physical parameters, and they took 3840 seconds to complete using the ToPoliNano multi-threaded version. To track the information propagation, 90 steps were required, and the outputs of the first and second stages are ready at steps 90 and 210, respectively.

It is worth noting that reducing the stability threshold can significantly reduce the required simulation time, albeit at the expense of lower result precision. For instance, by setting a stability threshold $\varepsilon_{max} = 5e - 3$, the simulation time was reduced to 724 seconds without affecting the circuit information propagation, at least from the graphical perspective.

Similarly to the 1-bit version, the 2-bit full adder underwent exhaustive testing for every possible input combination using proper testbenches, the validation script reported in Section 5.4, and the FCNviewer. The circuit performs as expected for all input combinations leveraging on the flow reported in Figure 8.1.

8.4 Molecular N-bit full adder

To create an N-bit full adder, the principle used to move from the 1-bit to the 2-bit version can be generalized. N adders are arranged in sequence and connected using $(N - 1)$ L-connections that propagate the carry from each block to the next. Each stage performs the addition between two single-bit inputs using the carry-out from the previous as the carry-in. The generalized N-bit full adder is shown in Figure 8.7.

The complexity of the circuit and the required analysis time depend on the number of stages. Specifically, the layout includes $3TN$ majority voters and $2TN$ inverters, where T is the clock period and N is the number of stages. The critical path of the circuit is determined by the carry propagation, producing the last sum after a delay of $10TN$.

The successful testing of the previous two full adders generates optimistic expectations for the performance of the N-bit full adder.

Additionally, due to its modular implementation, the N-bit full adder is an excellent example of a circuit that can exploit the hierarchical design by instantiating N black

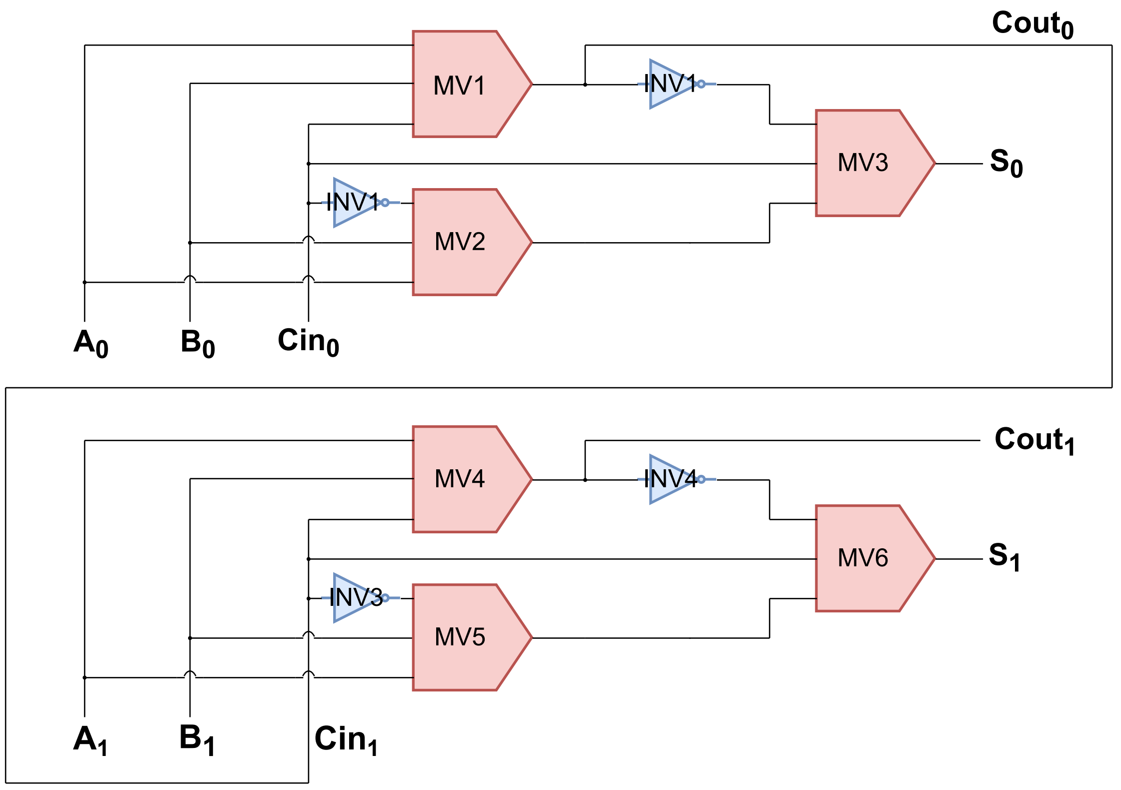


Figure 8.5. Optimized 2-bit full adder implemented with a majority voter and inverter-based logic.

boxes acting as the stages required by the adder. Figure 8.8 shows a 3-bit full adder implemented hierarchically with 1-bit full adders.

8.5 Molecular ISCAS C17

International Symposium on Circuits and Systems (ISCAS) is a prestigious electrical and electronics engineering annual convention, serving as a valuable occasion for academics, engineers, and industry professionals to collaborate and exchange knowledge on the latest research and cutting-edge technologies. The seminar covers several topics, emphasizing the design and implementation of analog and digital circuits and the importance of innovative validation and testing techniques to ensure high-quality and reliable electronic systems.

Through the years, it has cooperated closely with the EDA community by offering several circuit benchmarking collections to assess and compare the performance across logic simulation, synthesis, place-and-route, and verification tools. One of the most commonly used packages is the ISCAS'85 suite [18, 19], which consists of thirteen digital

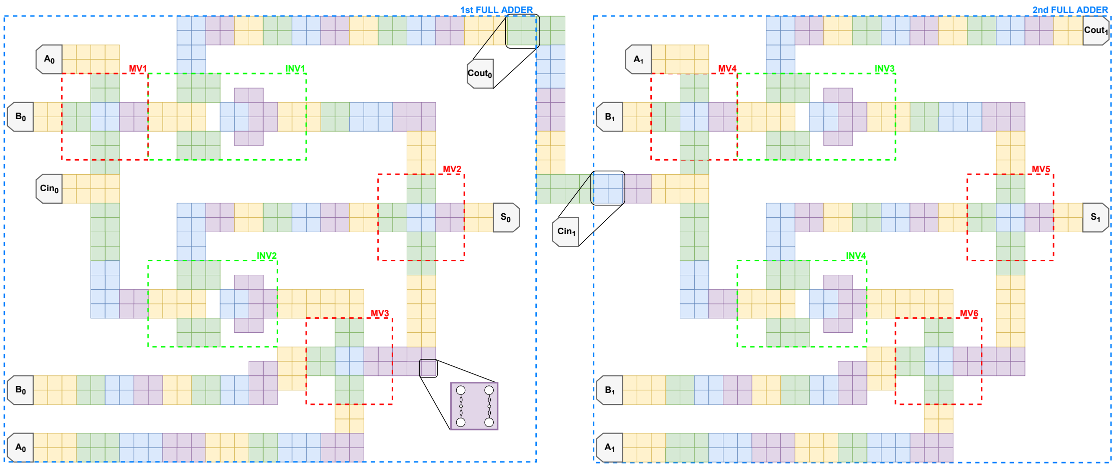


Figure 8.6. Optimized 2-bit full adder highlighting majority voter and inverter blocks and the double stage structure. Each square corresponds to an mQCA cell, consisting of two adjacent bis-ferrocene molecules.

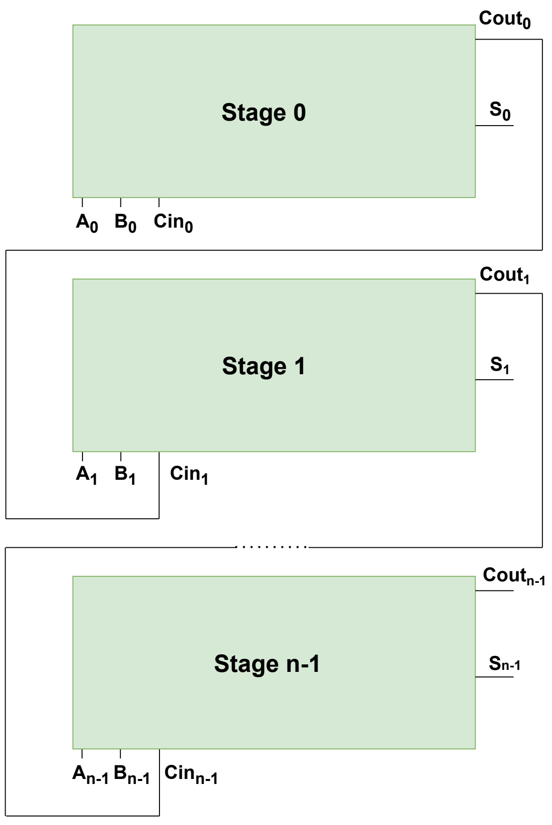


Figure 8.7. Schematic representation of an N-bit full adder layout.

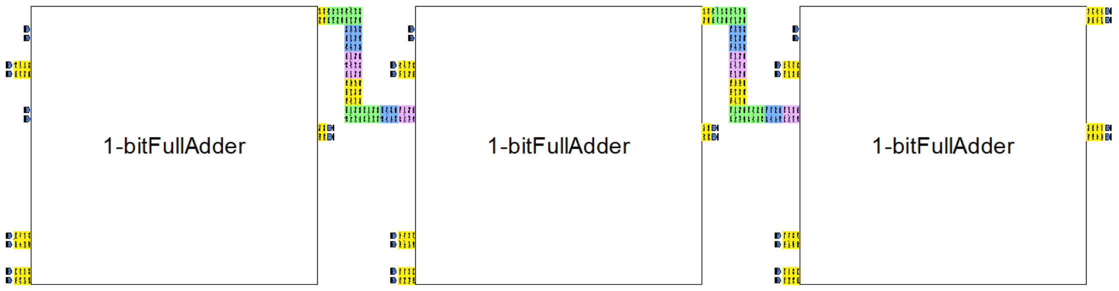


Figure 8.8. N-bit full adder hierarchical layout.

circuits of different complexity, spanning from simple basic logic gates to more sophisticated systems, such as comparators, arithmetic and logic units (ALUs), and multipliers.

Selecting the appropriate circuit for verifying the usability and effectiveness of the ToPoliNano EDA framework is essential. In the context of this work, the *C17* circuit is the best option as it strikes a suitable balance between complexity and testability.

8.5.1 Layout

Figure 8.9 displays the digital implementation of the ISCAS C17 circuit, which includes five inputs, two outputs, and six NAND logic gates. The outputs are generated according to the following Boolean expressions:

$$\begin{aligned} Out_1 &= (A \bar{\wedge} B) \bar{\wedge} ((B \bar{\wedge} C) \bar{\wedge} D) \\ Out_2 &= (D \bar{\wedge} (B \bar{\wedge} C)) \bar{\wedge} (E \bar{\wedge} (B \bar{\wedge} C)) \end{aligned} \tag{8.3}$$

where $\bar{\wedge}$ identifies the NAND operation. Table 8.3 displays the associated truth table.

The ISCAS C17 molecular implementation employs a 2-bit NAND logic gate as the fundamental building block whose truth table is provided in Table 8.2. The element uses a 3-input majority voter, with one input fixed at '0'. The result is complemented by a molecular inverter, which negates the output and implements the NAND function as:

$$A \bar{\wedge} B = NOT(MV(A, B, 0)) \tag{8.4}$$

where $\bar{\wedge}$, $NOT()$ and $MV()$ stands for NAND, NOT and majority voter operations, respectively. Figure 8.11 depicts the molecular 2-bit NAND logic gate, highlighting the majority voter and inverter gates.

However, due to the lack of complete support for information crossing, a direct transformation of the digital ISCAS C17 circuit displayed in Figure 8.9 is not feasible. As a result, signals that feed into multiple paths must be duplicated, resulting in the more complex configuration shown in Figure 8.10. The MagCAD editor is used to insert each NAND gate. Connections are realized by using vertical and horizontal wires arranged as L and T-connections. The circuit resembles a two-line structure to improve

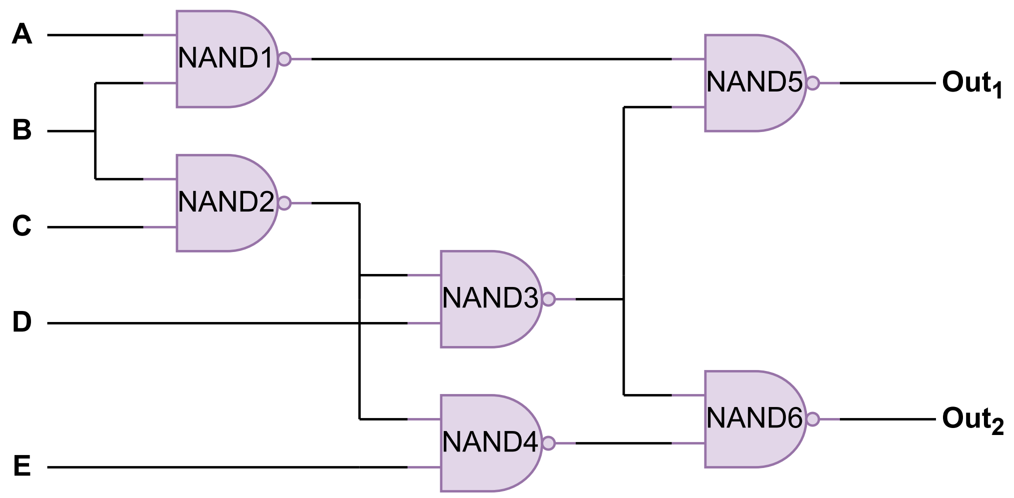


Figure 8.9. ISCAS C17 digital layout.

robustness, while the four-phase clocking scheme is carefully applied in each building block to guarantee adiabatic transmission [62] and ensure enough information encoding stability. In addition, horizontal and vertical propagation are implemented with no more than five consecutive cells with the same clock zones to avoid metastability issues. Furthermore, the phases are synchronized such that the two outputs are ready at the same time step. The complete molecular ISCAS C17 is reported in 8.12

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

Table 8.2. NAND truth table.

8.5.2 Simulation and testing

As per the previous circuits, the flow reported in Figure 8.1 is applied. A number of 100 steps are required to follow the entire information propagation from the inputs to the outputs. The simulations with the usual physical parameters and the multi-threaded version of ToPoliNano completed in 1324 seconds.

Exhaustive testing was employed to ensure circuit correctness for all possible inputs. In particular, the outputs are compared against the expected ones, using the truth table shown in 8.3 as a reference. Given the 2^5 combinations, instead of visually checking the output charge distributions using the FCNviewer, the script introduced in Section 5.4

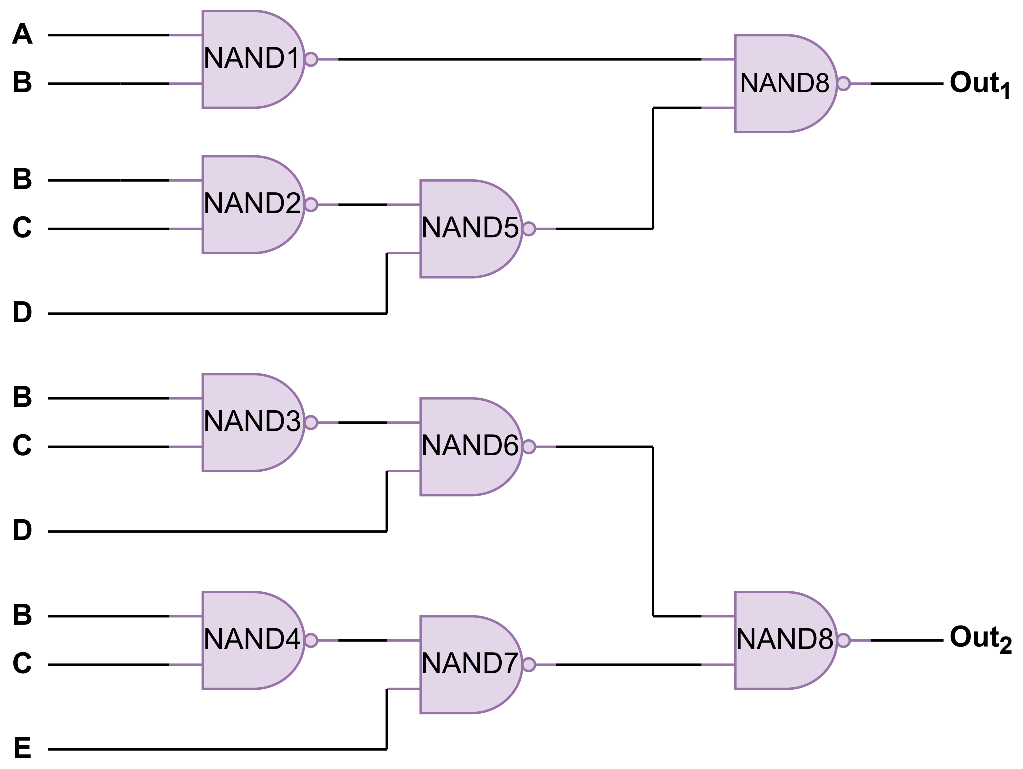


Figure 8.10. ISCAS C17 digital circuit implemented with duplicated inputs to overcome the need to use crosswire.

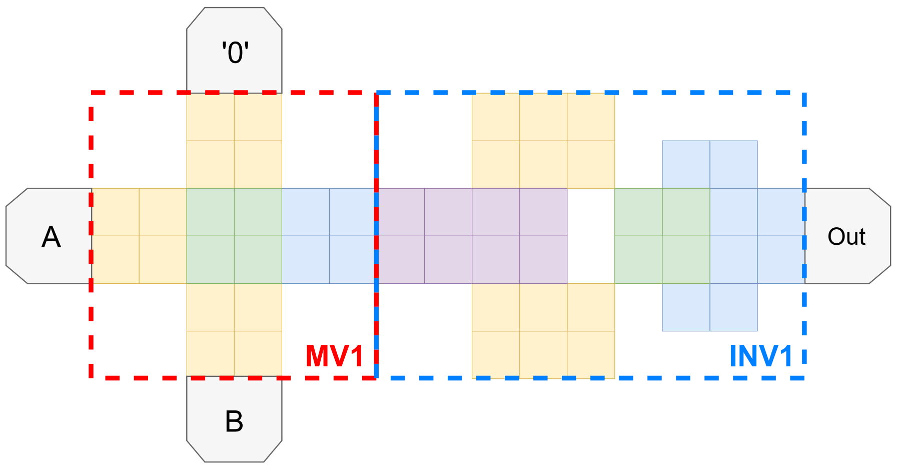


Figure 8.11. Two-line mFCN 2-input NAND layout employed a fundamental block to realize the mFCN ISCAS C17 circuit.

reduced the testing effort. The circuit performed as expected for all input combinations. In addition, after each NAND addition, a check for correct signal transfer throughout the layout was performed to streamline the global testing process.

Similar to previous circuits, the C17 could benefit from using information crossing to relax some design constraints. In particular, the crosswire could eliminate the need to duplicate some inputs, thereby saving global area. Additionally, a hierarchical design approach could be leveraged to instantiate NAND black box components, improving overall clarity and readability. As a reference, Figure 8.13 depicts the ISCAS C17 implemented hierarchically with NAND gates.

The successful implementation, testing, and verification of the molecular ISCAS C17 demonstrates the feasibility of transposing digital architectures into their molecular counterparts using the ToPoliNano framework.

A	**B**	**C**	**D**	**E**	**Out1**	**Out2**
0	0	0	0	0	0	0
0	0	0	0	1	0	1
0	0	0	1	0	0	0
0	0	0	1	1	0	1
0	0	1	0	0	1	1
0	0	1	0	1	1	1
0	0	1	1	0	1	1
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	0	1	0	1
0	1	0	1	0	0	0
0	1	0	1	1	0	0
0	1	1	0	0	1	1
0	1	1	0	1	1	1
0	1	1	1	0	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	0	0	1	0	1
1	0	0	1	0	0	0
1	0	0	1	1	0	1
1	0	1	0	0	1	1
1	0	1	0	1	1	1
1	0	1	1	0	1	1
1	0	1	1	1	1	1
1	1	0	0	0	1	0
1	1	0	0	1	1	1
1	1	0	1	0	1	0
1	1	0	1	1	1	0
1	1	1	0	0	1	1
1	1	1	0	1	1	1
1	1	1	1	0	1	0
1	1	1	1	1	1	0

Table 8.3. ISCAS C17 truth table.

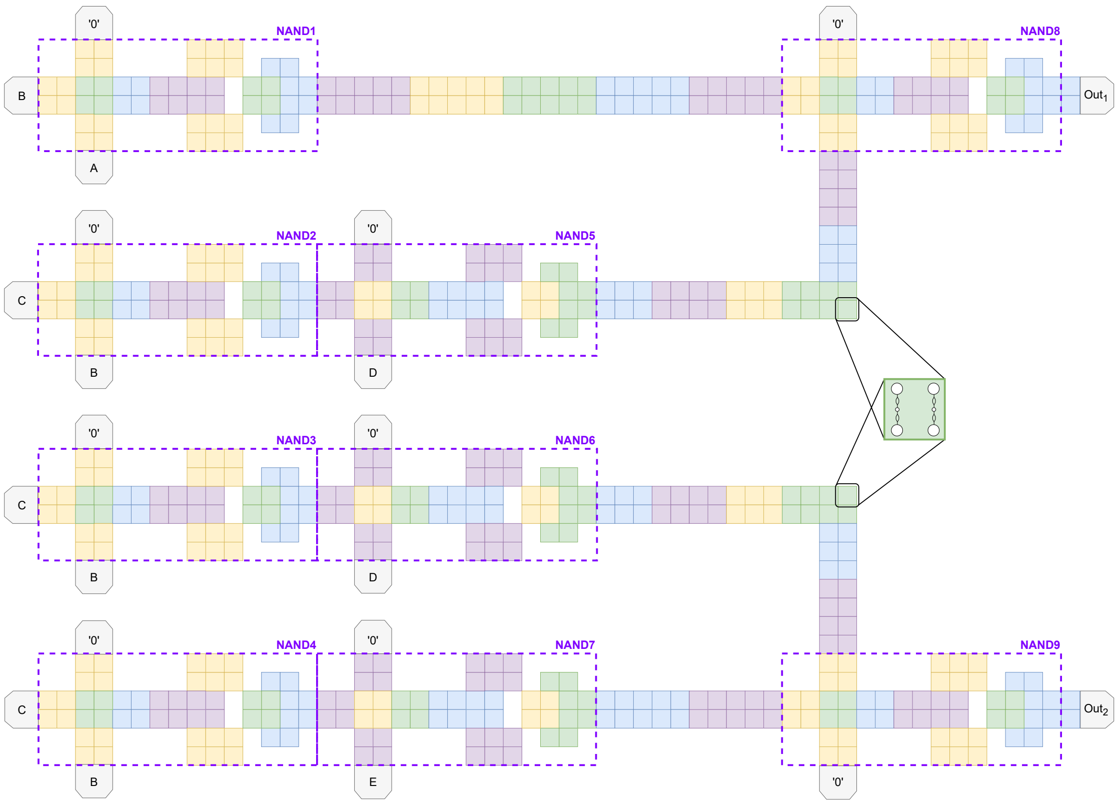


Figure 8.12. ISCAS C17 molecular circuit highlighting the fundamental NAND logic gates. Each square corresponds to an mQCA cell, consisting of two adjacent bis-ferrocene molecules.

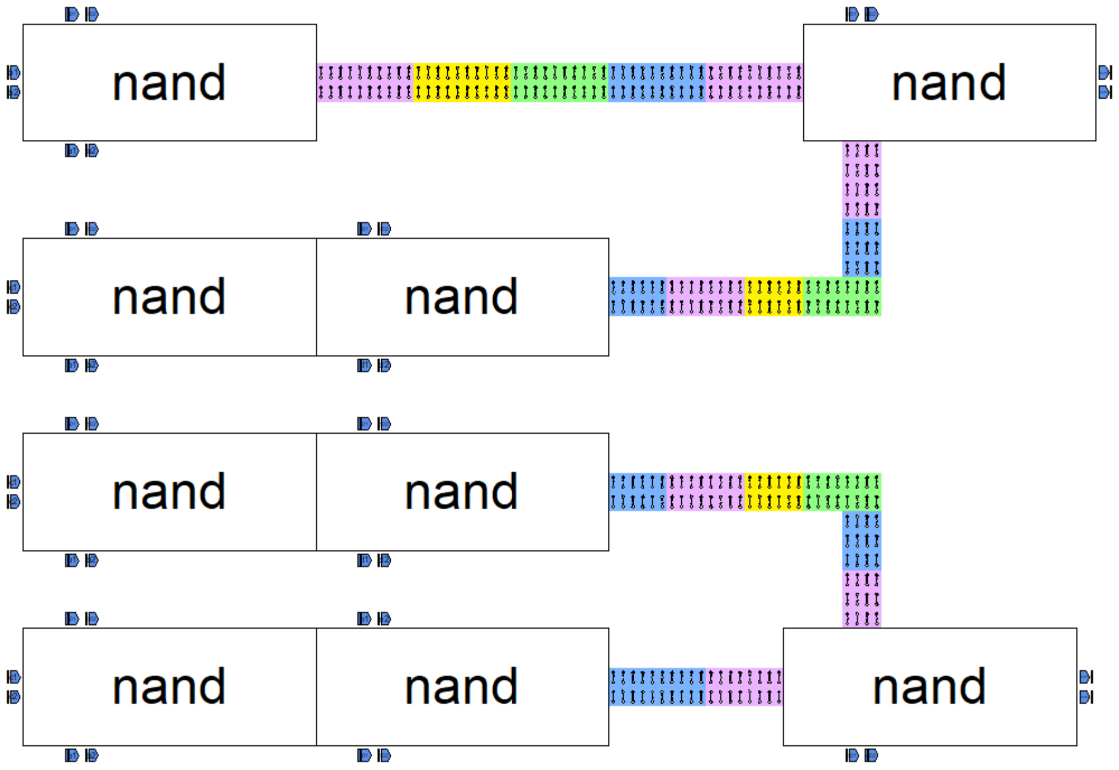


Figure 8.13. ISCAS C17 molecular circuit hierarchically implemented with NAND gates.

Conclusions and future perspectives

The presented work delves into the promising molecular Field-Coupled Nanocomputing (mFCN) paradigm, which offers numerous advantages over traditional silicon technology, including higher speed, lower power consumption, and greater scalability. However, practical device manufacturing still poses limitations that require further research.

To tackle these challenges, SCERPA has emerged as an effective tool for modeling molecules as electronic devices. It maintains a vivid connection to physical and technological perspectives and leverages ab initio calculations only for molecule characterizations. Specifically, SCERPA assesses the field generated by all the molecules in the circuit and links it to the charge, enabling a comprehensive analysis of signal propagation and computation.

In parallel, the ToPoliNano framework is being developed as an effective and intuitive EDA environment to offer a top-down approach similar to standard-silicon software for designing systems based on emerging technologies.

The recent integration of SCERPA into the ToPoliNano framework is a significant step toward bridging the gap between research and EDA scenario, creating a promising candidate for commercial Beyond-CMOS tool in the future. By leveraging the SCERPA's deep understanding of physical and technological details, the framework can design, simulate, and verify complex molecular architectures, employing a self-consistent iterative loop to solve electrostatic interactions without relying on computationally expensive ab initio methods.

This work objective is to validate the results produced by SCERPA and ToPoliNano, while simultaneously devising novel features and improvements for both tools, enhancing their functionality, usability, and compatibility.

A systematic methodology achieves the validation goal, proving highly effective in identifying and solving issues and discrepancies affecting the MagCAD importer of SCERPA and the ToPoliNano simulator. One significant advantage of this methodology is the ability to abstract the differences across the various SCERPA implementations, enabling a better understanding of potential discrepancies. In particular, the methodology is employed to compare the results of the two SCERPA importers, solving issues and demonstrating consistency between them. Similarly, the simulation results between research and EDA-oriented SCERPAs are validated for simple circuits. However, for complex layouts, discrepancies are observed. While a thorough comparison of every function of the two environments is conducted, the exact reason for the differences remains unknown. The degree of adiabaticity of the circuit solution could explain the reason for the differences. Increasing the number of interpolating points of the clock waveforms results in comparable results between the two tools.

Overall, the methodology has helped to build a more robust and reliable EDA platform, contributing to accurate and consistent circuit simulations.

This work introduces several new features and improvements to enhance the usability and effectiveness of ToPoliNano as a CAD tool for simulating mFCN systems.

One significant enhancement is the parallelization of the molecular engine responsible for computing interactions among molecules. This upgrade results in better performance compared to the previous single-threaded version. Increasingly complex molecular devices are simulated using the old single-threaded version and the new multi-threaded version to assess improvements. As expected, the results demonstrate a substantial enhancement in overall performance, with reduced simulation times across all devices. This aspect is particularly crucial in paving the way toward a commercial beyond-CMOS EDA tool capable of efficiently designing and simulating circuits regardless of the complexity.

In addition, this study adds hierarchical design support to the ToPoliNano simulator. The feature enables users to create nested sub-circuits and manage complex system designs more effectively, improving general clarity and readability. This aspect is particularly critical for a market-oriented EDA application that aims to abstract many low-level physical and electrical details of the simulation by providing an intuitive interface. The benefits of hierarchical layouts are assessed by applying them to the more complex circuits developed in this work. Specifically, the N-bit full adder benefits significantly by instantiating a number of 1-bit full adders, realizing an adder of any dimensions.

Moreover, to enhance circuit design flexibility, the newly introduced molecule support in ToPoliNano enables information exchange when two wires cross. This aspect increases the flexibility of circuit design, allowing for the creation of circuits with fewer constraints. To engineer the crosswire layout, MagCAD is employed to design it, while ToPoliNano is used to simulate it. However, due to the lack of low-level personalization of external clock fields, The evaluation cell may not always be situated in optimal conditions to generate the intended outputs. As a result, using the same crosswire layout proposed by the literature for all input combinations is impossible. Therefore, this work introduces two crosswire layouts to handle separately the cases in which the inputs are equal or not. This issue highlights the need for further work to introduce more fine-tuned clock personalization at the ToPoliNano level or explore new molecules capable of supporting information crossing regardless of external conditions. Overall, the successful integration of the new molecule and the flow used to introduce it demonstrates the potential for adding new molecules to ToPoliNano's capabilities and paves the way for the possibility of hybrid molecular circuits.

The study also emphasizes the refined and validated ToPoliNano suite to be used to analyze complex molecular circuits. One of the notable achievements is the successful transposition of the 1-bit full adder layout to a 2-bit version, which demonstrates the generalization of the N-bit version. Moreover, the framework's effectiveness is demonstrated by transposing digital architectures into molecular equivalents, such as the widely-used digital benchmark circuit ISCAS C17.

To improve its effectiveness and versatility as a state-of-the-art research algorithm for studying mFCN, SCERPA is enhanced with a dynamic damping feature that adapts the parameter already used to facilitate iterative loop convergence, resulting in faster analysis. A comparison with static damping evaluates performance improvement.

The research procedure now includes the same charge convergence mode as ToPoliNano to evaluate the convergence of the iterative loop, improving its comparability with its CAD counterpart. Furthermore, now SCERPA comprises the same clock waveforms as ToPoliNano to regulate information propagation through circuits.

Regarding future perspectives, there are several promising areas of investigation for both SCERPA and ToPoliNano.

The academic procedure could enhance the accuracy of performance, power estimations, and overall analysis by fully characterizing time-dependent properties such as electron movement through molecular dynamics concepts like Real-Time Time-Dependent Density Functional Theory (RT-TDDFT). Although the current self-consistent iterative loop provides valuable modeling, it does not fully describe the actual behavior of molecules over time. Incorporating a dynamic description of the system would offer a more comprehensive and realistic understanding of how molecules behave, a critical aspect in paving the way for mFCN prototype realization. Additionally, the algorithm could serve as a basis for exploring alternative methods for mFCN computation, such as employing molecule positions instead of charges or through a combination of magnetic and electrostatic external stimuli. Another encouraging research topic for SCERPA could be related to improving the characterization of process variations, which could help assess their effect on final devices and facilitate concrete device realization.

Similarly, the ToPoliNano framework could leverage SCERPA's deep understanding of the technology and evaluate the integrability of research insights to enhance practical EDA applications. For instance, the framework could integrate the SCERPA model to assess the energy dissipated by a molecular circuit. Meanwhile, other visualization methods that mimic the range of representations provided by the SCERPA viewer could enhance the FCNviewer. Moreover, the entire EDA framework could explore technologies across different domains, allowing for the design of hybrid emerging technology circuits.

References

[1] M. Ali and A. Esam. «A new approach to bypass wire crossing problem in QCA nano technology», in: *Circuit World* (2021).

[2] I. Amlani, A. O. Orlov, G. L. Snider, C. S. Lent, and G. H. Bernstein. «External charge state detection of a double-dot system», in: *Applied physics letters* 71.12 (1997).

[3] Y. Ardesi. «Investigation of Molecular FCN for Beyond-CMOS. Technology, design, and modeling for nanocomputing». PhD thesis. 2022.

[4] Y. Ardesi, G. Beretta, M. Vacca, Gianluca G. Piccinini, and M. Graziano. «Impact of Molecular Electrostatics on Field-Coupled Nanocomputing and Quantum-Dot Cellular Automata Circuits», in: *Electronics* 11.2 (2022).

[5] Y. Ardesi, U. Garlando, F. Riente, G. Beretta, G. Piccinini, and M. Graziano. «Taming Molecular Field-Coupling for Nanocomputing Design», in: *Emerging Technologies in Computing Systems* 19.1 (2022), pp. 1–24.

[6] Y. Ardesi, L. Gnoli, M. Graziano, and G. Piccinini. «Bistable propagation of monostable molecules in molecular field-coupled nanocomputing», in: *Proc. 15th Conf. Ph.D. Res. Microelectron. Electron* (2019), pp. 225–228.

[7] Y. Ardesi, M. Graziano, and G.Piccinini. «A Model for the Evaluation of Monostable Molecule Signal Energy in Molecular Field-Coupled Nanocomputing», in: *Journal of Low Power Electronics and Applications* 12.1 (2022).

[8] Y. Ardesi, A. Pulimeno, M. Graziano, F. Riente, and G. Piccinini. «Effectiveness of Molecules for Quantum Cellular Automata as Computing Devices», in: *Low Power Electronics and Applications* (2018).

[9] Y. Ardesi, A. Pulimeno, M. Graziano, F. Riente, and G. Piccinini. «Effectiveness of molecules for quantum cellular automata as computing devices», in: *Low Power Electronics and Applications* 8.3 (2018), p. 24.

[10] Y. Ardesi, G. Turvani, G. Piccinini, and M. Graziano. «SCERPA Simulation of Clocked Molecular Field-Coupling Nanocomputing», in: *IEEE Transactions. Very Large Scale Integration (VLSI) Systems* 29.3 (2021), pp. 558–567.

[11] Y. Ardesi, R. Wang, G. Turvani, G. Piccinini, and M. Graziano. «SCERPA: A Self-Consistent Algorithm for the Evaluation of the Information Propagation in Molecular Field-Coupled Nanocomputing», in: *IEEE Transactions Computer-Aided Design Integrated Circuits Systems* 39.10 (2020), pp. 2749–2760.

[12] V. Arima, M. Iurlo, L. Zoli, S. Kumar, M. Piacenza, F. Della Sala, F. Matino, G. Maruccio, R. Rinaldi, F. Paolucci, M. Marcaccio, P. Cozzi, and A. Bramanti. «Toward quantum-dot cellular automata units: Thiolated-carbazole linked bisferrocenes», in: *Nanoscale* 4 (2012), pp. 813–823.

[13] P. W. Atkins and J. D. Paula. «Physical Chemistry», in: (2006).

[14] R. Iris Bahar, Alex K. Jones, Srinivas Katkoori, Patrick H. Madden, Diana Marculescu, and Igor L. Markov. «Workshops on Extreme Scale Design Automation (ESDA) Challenges and Opportunities for 2025 and Beyond», in: *CoRR* (2020).

[15] Gary H. Bernstein, Alexandra Imre, V. Metlushko, Alexei O. Orlov, L. Zhou, L. Ji, György Csaba, and Wolfgang Porod. «Magnetic QCA systems», in: *Microelectronics Journal* 36.7 (2005), pp. 619–624.

[16] «Beyond CMOS», in: *International Roadmap for Devices and Systems* (2022).

[17] E. P. Blair. «Quantum-dot cellular automata: A clocked architecture for highspeed, energy-efficient molecular computing», in: *Unconventional Computation and Natural Computation* (2017), pp. 56–68.

[18] F. Brglez and H. Fujiwara. «A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran», in: *Int. Symposium on Circuits and Systems* (1985).

[19] F. Brglez, P. Pownall, and R. Hum. «Accelerated ATPG and Fault Grading via Testability Analysis», in: *Proc. IEEE Int. Symposium on Circuits and Systems* (1985).

[20] Y. S. Chauhan, D. D. Lu, S. Vanugopalan, S. Khandelwal, J.P. Duarte, N. Paydavosi, A. Niknejad, and C. Hu. *FinFET Modeling for IC Simulation and Design.* Academic Press, 2015.

[21] T. C. Chen. «Overcoming Research Challenges for CMOS Scaling», in: *Industry Directions", Proc. of 8th Intl. Conference on Solid-State and Integrated Circuit Technology* (2006), pp. 4–7.

[22] R. P. Cowburn and M. E. Welland. «Room temperature magnetic quantum cellular automata», in: *Science* (2000).

[23] B.D. Cullity and C.D. Graham. «Introduction to Magnetic Materials», in: *Wiley* (2009).

[24] M. A. Cusumano. «The Changing Economics and Strategy of the Semiconductor Industry», in: *IEEE Engineering Management Review* 45.3 (2017).

[25] P. Douglas and C. S. Lent. «Logical devices implemented using quantum cellular automata», in: *Applied Physics* 75 (1994).

[26] T. Farrelly. «A review of Quantum Cellular Automata», in: *Quantum 4* 4 (2020).

[27] U. Garlando, F. Riente, and M. Graziano. «FUNCODE: Effective Device-to-System Analysis of Field-Coupled Nanocomputing Circuit Designs», in: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.3 (2021), pp. 467–478.

[28] M. Graef. «More than Moore White Paper», in: *IEEE International Roadmap for Devices and Systems Outbriefs* (2021), pp. 1–47.

[29] M. Graziano, M. Vacca, and M. Zamboni. «Magnetic QCA Design: Modeling, Simulation and Circuits», in: *Cellular Automata - Innovative Modelling for Science and Engineering* (2011).

[30] M. Graziano, R. Wang, M. R. Roch, Y. Ardesi, F. Riente, and G. Piccinini. «Characterisation of a bis-ferrocene molecular QCA wire on a non ideal gold surface», in: *Micro & Nano Letters* 14 (2019), pp. 22–27.

[31] N. Z. Haron. «Why is CMOS scaling coming to an END?», in: *3rd International Design and Test Workshop* (2009), pp. 98–103.

[32] A. Ilachinski. «Cellular Automata: A Discrete View of the World», in: *World Scientific Publishing Co Inc.* (2001).

[33] C. S. Lent, B. Isaksen, and M. Lieberman. «Molecular quantum-dot cellular automata», in: *American Chemical Society* 125.4 (2003), pp. 1056–1063.

[34] C. S. Lent and Gregory L. Snider. «The development of quantum-dot cellular automata», in: *In Field-Coupled Nanocomputing, Springer Berlin Heidelberg* (1994), pp. 3–20.

[35] C.S. Lent and B. Isaksen. «Clocked molecular quantum-dot cellular automata», in: *IEEE Transactions on Electron Devices* 50.9 (2003), pp. 1890–1896.

[36] C.S. Lent, P.D. Tougaw, W. Porod, and G.H. Bernstein. «Quantum cellular automata», in: *Nanotechnology* 4 (1993), pp. 49–57.

[37] D. Litvinov and K. Ingersent. «Magnetic Quantum Cellular Automata», in: *Journal of Applied Physics* (2004).

[38] Y. Lu and C. S. Lent. «A metric for characterizing the bistability of molecular quantum-dot cellular automata», in: *Nanotechnology* 19.15 (2008).

[39] Y. Lu, M. Liu, and C.S. Lent. «Molecular quantum-dot cellular automata: From molecular structure to circuit dynamics», in: *Applied physics* (2007), p. 102.

[40] M. Macucci. *Quantum cellular automata: theory, experimentation and prospects.* London: Imperial College Press, 2006.

[41] F. P. Martinez, I. Farrer, D. Anderson, G. A. C. Jones, D. A. Ritchie, S. J. Chorley, and C. G. Smith. «Demonstration of a quantum cellular automata cell in a Ga As/ Al Ga As heterostructure», in: *Applied physics letters* 91.3 (2007).

[42] U. Mehta and V. H. Dhare. «Quantum-dot Cellular Automata (QCA): A Survey», in: *CoRR* (2017).

[43] G. E. Moore. «Cramming more components onto integrated circuits», in: *Electronics* (1965).

[44] «More Moore», in: *International Roadmap for Devices and Systems* (2022).

[45] «More than Moore», in: *International Roadmap for Devices and Systems* (2022).

[46] A. O. Orlov, I. Amlani, G. H. Bernstein, C. S. Lent, and G. L. Snider. «Realization of a functional cell for quantum-dot cellular automata», in: *Science* 277.5328 (1997).

[47] A. O. Orlov, R. K. Kummamuru, R. Ramasubramaniam, G. Toth, C. S. Lent, G. H. Bernstein, and G. L. Snider. «Experimental demonstration of a latch in clocked quantum-dot cellular automata», in: *Applied Physics Letters* 78.11 (2001).

[48] M. Ottavi, S. Pontarelli, and A. Salsano. «Modeling Magnetic Quantum-dot Cellular Automata by HDL», in: *11th IEEE International Conference on Nanotechnology* (2011), pp. 1139–1144.

[49] A. M. Pintus, A. Gabrieli, F. G. Pazzona, G. Pireddu, and P. Demontis. «Molecular QCA embedding in microporous materials», in: *Physical Chemistry Chemical Physics* 21 (15 2019), pp. 7879–7884.

[50] A. Pulimeno, M. Graziano, A. Antidormi, R. Wang, A. Zahir, and G. Piccinini. «Understanding a Bisferrocene Molecular QCA Wire», in: *Springer Berlin Heidelberg* 8280.2 (2014), pp. 307–338.

[51] A. Pulimeno, M. Graziano, D. Demarchi, and G. Piccinini. «Towards a molecular QCA wire: simulation of write-in and read-out systems», in: *Solid-State Electronics* 77 (2012), pp. 101–107.

[52] A. Pulimeno, M. Graziano, R. Wang, D. Demarchi, and G. Piccinini. «Charge distribution in a molecular QCA wire based on bis-ferrocene molecules», in: *IEEE International Symposium on Nanoscale Architectures* (2013), pp. 42–43.

[53] A. Quarteroni and F. Saleri. «Numerical Mathematics», in: *Springer-Verlag* (2006).

[54] F. Riente, U. Garlando, G. Turvani, M. Vacca, M. R. Roch, and M. Graziano. «MagCAD: A Tool for the Design of 3D Magnetic Circuits», in: *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 3 (2017), pp. 65–73.

[55] F. Riente, G. Turvani, M. Vacca, M. R. Roch, M. Zamboni, and M. Graziano. «ToPoliNano: A CAD Tool for Nano Magnetic Logic», in: 36.7 (2017), pp. 1061–1074.

[56] L. C. S. and P. D. Tougaw. «A device architecture for computing with quantum dots», in: *Proceedings of the IEEE* 85.4 (1997).

[57] E. Scott and Thompson et al. «In Search of 'Forever,' Continued Transistor Scaling One New Material at a Time», in: *IEEE Transactions on Semiconductor Manufacturing* 18.1 (2005), pp. 26–36.

[58] U. C. Singh and P. A. Kollman. «An approach to computing electrostatic charges for molecules», in: *Journal on Computational Chemistry* 5 (1984), pp. 129–145.

[59] R. O. Topaloglu. «More than Moore Technologies for Next Generation Computer Design», in: *Springer* (2016).

[60] P.D. Tougaw and C. S. Lent. «Logical devices implemented using quantum cellular automata», in: *Applied Physics* 75 (1994).

[61] J. M. Tour. «Molecular Electronics. Synthesis and Testing of Components», in: *Accounts of Chemical Research* 33 (2000).

[62] V. Vankamamidi, M. Ottavi, and F. Lombardi. «A line-based parallel memory for QCA implementation», in: *IEEE Transactions on Nanotechnology* 4.6 (2005).

[63] V. Vankamamidi, M. Ottavi, and F. Lombardi. «Clocking and Cell Placement for QCA», in: *In Proceedings of theIEEE-NANO* 1 (2006).

[64] A. Vetteth, K. Walus, V. S. Dimitrov, and G. A. Jullien. «Quatum-dot cellular automata carry-look-ahead adder and barrel shifter», in: *IEEE Emerging Telecommunications Technologies Conference* (2002).

[65] K. Walus, T. J. Dysart, G. A. Jullien, and R. A. Budiman. «QCADesigner: A rapid design and simulation tool for quantum-dot cellular automata», in: *Nanotechnology* 3.1 (2004), pp. 26–31.

[66] W. Wang, K. Walus, and G. A. Jullien. «Quantum-Dot Cellular Automata Adders», in: *IEEE Conference on Nanotechnology* (2003).

[67] A. Yuri, U. Garlando, F. Riente, G. Beretta, G. Piccinini, and M. Graziano. «Taming Molecular Field-Coupling for Nanocomputing Design», in: 19.1 (2022).

[68] L. Zoli. «Active Bis-Ferrocene Molecules as Unit for Molecular Computation». PhD thesis. 2010, pp. 1056–1063.

Appendix A

Comparative analysis of the molecular FCN simulators

This appendix includes the code developed to facilitate the comparison of the two SCERPA layout importers and the SCERPA and ToPoliNano simulators.

A.1 Evaluating SCERPA importers

Listing A.1. MATLAB source code of `SaveLabelsAndDot4Coord.m`.

```
function Function_SaveLabelsAndDot4Coord(stack_mol, stack_driver, circuit_name, out_path)

%file management
    if strcmp(circuit_name, 'matlabDrawing')
        fileLabelMatlab = sprintf('%s/labelsAndDot4CoordinatesMatlab.txt', out_path);
        fileIDMatlab = fopen(fileLabelMatlab,'wt');
    else
        fileLabelMagcad = sprintf('%s/labelsAndDot4CoordinatesMagcad.txt', out_path);
        fileIDMagcad = fopen(fileLabelMagcad,'wt');
    end

    %insert drivers
    for kk=1:stack_driver.num
        if strcmp(circuit_name, 'matlabDrawing')
          fprintf(fileIDMatlab,'Driver %s %.4f %.4f %.4f\n', stack_driver.stack(kk).identifier_qll,
 stack_driver.stack(kk).charge(4).x, stack_driver.stack(kk).charge(4).y, stack_driver.stack(kk).charge(4)
 .z);
        else
          fprintf(fileIDMagcad,'Driver %s %.4f %.4f %.4f\n', stack_driver.stack(kk).identifier_qll,
 stack_driver.stack(kk).charge(4).x, stack_driver.stack(kk).charge(4).y, stack_driver.stack(kk).charge(4)
 .z);
        end
     end

    %insert molecules
    for kk=1:stack_mol.num
        if strcmp(circuit_name, 'matlabDrawing')
          fprintf(fileIDMatlab,'Molecule %s %.4f %.4f %.4f\n', stack_mol.stack(kk).identifier_qll, stack_mol.
 stack(kk).charge(4).x, stack_mol.stack(kk).charge(4).y, stack_mol.stack(kk).charge(4).z);
        else
          fprintf(fileIDMagcad,'Molecule %s %.4f %.4f %.4f\n', stack_mol.stack(kk).identifier_qll, stack_mol.
 stack(kk).charge(4).x, stack_mol.stack(kk).charge(4).y, stack_mol.stack(kk).charge(4).z);
        end
    end

    %close file
    if strcmp(circuit_name, 'matlabDrawing')
        fclose(fileIDMatlab);
    else
        fclose(fileIDMagcad);
    end
end
```

Listing A.2. Bash source code of `generateMatchingMolecules.sh`.

```bash
#!/bin/bash
####################################################################
#Script Name  : generateMatlabMagcadTopolinanoCorrespondences.sh
#Description  : the script receives the name of the circuit to be tested.
#                 It requires to have already executed the corrispondent
#                 MATLAB testing file from the Testing folder.
#                 It generates the label correspondences between
#                 MATLAB and MagCAD(= ToPoliNano) outputs according
#                 to the molecules DOT4 coordinates
#Inputs         : name of the circuit to be tested (i.e. ThreePhasesWire)
#Outputs        : label matche between SCERPA-MATLAB and SCERPA-MagCAD
#Author         : Dario Castagneri
#####################################################################
start_time="$(date -u +%s)"

#Move to main SCERPA directory
cd ..

#Files generated by scerpa-MATLAB and scerpa-MagCAD
#storing the labels along with the coordinates of DOT4
matlab_file="./$1/matlab/SCERPA_OUTPUT_FILES/labelsAndDot4CoordinatesMatlab.txt"
magcad_file="./$1/magcad/SCERPA_OUTPUT_FILES/labelsAndDot4CoordinatesMagcad.txt"

#Check if correspondences file is already
#present due to a previous test.
#If yes, delete it.
correspondencesFile="./$1/labelsMatlabMagcadTopo.txt"
if [ -f "$correspondencesFile" ]; then
    rm $correspondencesFile
fi

nRows=0
nLabels=0
#Generate the correspondences between the labels of MATLAB and
#MagCAD by searching the labels with the same DOT4 coordinates.
while IFS=' ' read -r matCol1 matCol2 matCol3 matCol4 matCol5
do
    let nRows=nRows+1
    while IFS=' ' read -r magCol1 magCol2 magCol3 magCol4 magCol5
    do
        if [[ $matCol3 == $magCol3 &&
              $matCol4 == $magCol4 &&
              $matCol5 == $magCol5 ]]; then
            echo $matCol2 $magCol2 >> $correspondencesFile
            let nLabels=nLabels+1
            break                #Break the loop to save computation time.
        fi
    done < "$magcad_file"
done < "$matlab_file"

#Check wether the number of correspondences is
#equal to tne number of labels.
#If not, exit the program.
if [ $nRows -ne $nLabels ]; then
    echo "It was not possible to find all the correspondences"
    exit 1
fi

end_time="$(date -u +%s)"
elapsed="$(($end_time-$start_time))"
echo "Execution time is $elapsed seconds."
```

Listing A.3. Bash source code of `checkMatlabMagcadQSS.sh`.

```bash
#!/bin/bash
####################################################################
#Script Name  : checkMatlabMagcadQSS.sh
#Description  : the script receives the name of the circuit to be tested.
#                 It requires to have already executed the corrispondent
#                 MATLAB testing file from the Testing folder.
#                 It assumes that the generateMatlabMagcadTopolinanoCorrespondences.sh script
#                 has already been executed to generate labels between Matlab and Magcad (= Topolinano)
#                 It checks for any mismateches between matlab and magcad .qss files.
#Argument       : name of the circuit to be tested (i.e. ThreePhasesWire)
#Author         : Dario Castagneri
```

```bash
####################################################################
start_time="$(date -u +%s)"

#Move to main directory
cd ..

# Correspondences file is already generated by generateMatlabMagcadTopolinanoCorrespondences.sh script
corresponcencesFile="./$1/labelsMatlabMagcadTopo.txt"

#Check if mismatches file is already present due to a previous test.
#If yes, delete it.
mimsmatchesFile="./$1/mismatchesMatlabMagcad.txt"
if [ -f "$mimsmatchesFile" ]; then
    rm $mimsmatchesFile
fi


lineNumber=35      #Considering the header, lineNumber specifies from where the reading starts.
numberQss=39
i=0                #i specifies the number of QSS files to be compared.
while [ $i -le $numberQss ]
do
matlabQss="./$1/matlab/SCERPA_OUTPUT_FILES/00000$i.qss"
magcadQss="./$1/magcad/SCERPA_OUTPUT_FILES/00000$i.qss"

if [ $i -ge 10 ]; then
    matlabQss="./$1/matlab/SCERPA_OUTPUT_FILES/0000$i.qss"
    magcadQss="./$1/magcad/SCERPA_OUTPUT_FILES/0000$i.qss"
fi

echo "Checking: "
echo $matlabQss
echo $magcadQss

    #read MATLAB qss
    while IFS=' ' read -r matCol1 matCol2 matCol3 matCol4 matCol5
    do
            #read corresponcences file
            while IFS=' ' read -r corrMatlab corrMagcad;
            do
            if [[ $corrMatlab == $matCol1 ]]; then
                                #read MagCAD qss
                                while IFS=' ' read -r magCol1 magCol2 magCol3 magCol4 magCol5
                                    do
                                    if [[ $corrMagcad == $magCol1 ]]; then
                                        if [[ ${matCol2:0:5} != ${magCol2:0:5} || ${matCol3:0:5} != ${magCol3:0:5} || ${matCol4:0:5} != ${magCol4:0:5} || ${matCol5:0:5} != ${magCol5:0:5} ]]; then
                                                echo "Found a mismatch between MATLAB 0${i}.qss and MagCAD 0${i}.qss files:" >> $mimsmatchesFile
                                                echo "MATLAB reports: $matCol1 $matCol2 $matCol3 $matCol4 $matCol5" >> $mimsmatchesFile
                                                echo -e "MagCAD reports: $magCol1 $magCol2 $magCol3 $magCol4 $magCol5\n" >> $mimsmatchesFile
                                        fi
                                    fi
                                done < <(tail -n +$lineNumber $magcadQss)
            fi
            done < "$corresponcencesFile"
    done < <(tail -n +$lineNumber $matlabQss)
let "i=i+1"
done

end_time="$(date -u +%s)"
elapsed="$(($end_time-$start_time))"

echo "Execution time is $elapsed seconds."
```

A.2 Evaluating SCERPA and ToPoliNano simulators

Listing A.4. Bash source code of `checkMatlabTopolinanoQSS.sh`.

```bash
#!/bin/bash
####################################################################
#Script Name  : checkMatlabTopolinanoQSS.sh
#Description  : the script receives the name of the circuit to be tested and maximum offset.
#                 It directly compares the dot charges looking for the same molecule label in topolinano and magcad .qss files.
#                 To be used before the modifications to inital voltages and clock in topolinano (initial voltage 4.5V -> 0 and clock 0 -> -2)
```

```bash
#Argument       : name of the circuit to be tested (i.e. ThreePhasesWire), maximum offset allowed to consider a mismatch
#Author         : Dario Castagneri
#################################################################

start_time="$(date -u +%s)"


##############################################
computeDifferenceCharges () {
    #echo "$1 $2"

    if (( $(echo "$1 >= $2" | bc -l) )); then
        difference=$(echo "$1 - $2" | bc)
    else
        difference=$(echo "$2 - $1" | bc)
    fi
}
##############################################


offsetOK=$2

#Move to main directory
cd ..

#Check if mismatches file is already present due to a previous test.
#If yes, delete it.
mimsmatchesFile="./$1/mismatchesMagcadTopo.txt"
if [ -f "$mimsmatchesFile" ]; then
    rm $mimsmatchesFile
fi

topoDirectory="/mnt/a/Utenti/Dario/Desktop/topoWorkSpace/Work/MolQCAProject/Simulations/"
eval "cd $topoDirectory"
simName=$(eval "ls -t | head -n 1")
topoDir="$topoDirectory$simName"
eval "cd /mnt/c/scerpa"


lineNumberMatlab=35      #Considering the header, lineNumber specifies from where the reading starts.
lineNumberTopo=15
numberQss=39
i=0                      #i specifies the number of QSS files to be compared.
while [ $i -le $numberQss ]
do
matlabQss="./$1/magcad/SCERPA_OUTPUT_FILES/00000$i.qss"
topoQss="$topoDir/00000$i.qss"

if [ $i -ge 10 ]; then
    matlabQss="./$1/magcad/SCERPA_OUTPUT_FILES/0000$i.qss"
    topoQss="$topoDir/0000$i.qss"
fi

echo "Checking: "
echo $matlabQss
echo $topoQss

    #read MATLAB qss
    while IFS=' ' read -r matCol1 matCol2 matCol3 matCol4 matCol5
    do
        #read Topo qss
        while IFS=$'\t' read -r topoCol1 topoCol2 topoCol3 topoCol4 topoCol5
            do
            if [[ $matCol1 == $topoCol1 ]]; then
                if [[ $matCol2 != $topoCol2 || $matCol3 != $topoCol3 || $matCol4 != $topoCol4 || $matCol5 != $topoCol5 ]]; then

                        computeDifferenceCharges $matCol2 $topoCol2
                            DOT1difference=$difference

                        computeDifferenceCharges $matCol3 $topoCol3
                            DOT2difference=$difference

                        computeDifferenceCharges $matCol4 $topoCol4
                            DOT3difference=$difference

                        matCol5=$(echo "$matCol5" | tr -d $'\r')
                        topoCol5=$(echo "$topoCol5" | tr -d $'\r')

                        computeDifferenceCharges $matCol5 $topoCol5
                            DOT4difference=$difference

                        #difference is greater for one of the four parameters wrt to offset
                        if (( $(echo "$DOT1difference > $2" | bc -l) || $(echo "$DOT2difference > $2" | bc -l) || $(echo "$DOT3difference > $2" | bc -l) || $(echo "$DOT4difference > $2" | bc -l) )); then
                            echo "Found a mismatch between MATLAB 0${i}.qss and MagCAD 0${i}.qss files greater than maximum +/- allowed discrepancy:" >> $mimsmatchesFile
                            echo "Maximum +/- allowed discrepancy: $offsetOK" >> $mimsmatchesFile
                            echo -e "MATLAB reports: \t$matCol1 \t$matCol2 \t$matCol3 \t$matCol4 \t$matCol5" >> $mimsmatchesFile
                            echo -e "ToPoliNano reports: \t$topoCol1 \t$topoCol2 \t$topoCol3 \t$topoCol4 \t$topoCol5" >> $mimsmatchesFile
                            echo -e "Differences are: \t\t\t$DOT1difference \t$DOT2difference \t$DOT3difference \t$DOT4difference\n" >> $mimsmatchesFile
                        fi
                fi
            fi
        done < <(tail -n +$lineNumberTopo $topoQss)
    done < <(tail -n +$lineNumberMatlab $matlabQss)
let "i=i+1"
done

end_time="$(date -u +%s)"
elapsed="$(($end_time-$start_time))"

echo "Execution time is $elapsed seconds."
```

A.3 Molecule encoding checker

Listing A.5. Bash source code of `moleculeEncodingChecker.sh`.

```
#!/bin/bash
###################################################################
#Script Name  : checkTopolinanoMoleculeEncoding.sh
#Description  : The script receives a list of x and y coordinates of molecules whose encoding state needs to be checked.
#Argument       : The QSS to be checked, list of x-coordinates, list of y-coordinates, digital values.
#                 (i.e. 90 "32 32" "0 13" "0 1")
#Author         : Dario Castagneri
####################################################################
zeroThreshold=0.8
oneThreshold=0.3

qssToBeChecked=$1
echo "QSS to be checked: 00$qssToBeChecked.qss"
qssFile="$PWD/0000$qssToBeChecked.qss"

xCoordinates=($2)
yCoordinates=($3)
digitalValues=($4)

qllFile=$(find . -maxdepth 1 -name "*.qll" -type f)
moleculeIds=()

for (( i=0; i<${#xCoordinates[@]}; i++ ));
do
    tag=$(grep -m1 "x=\"${xCoordinates[$i]}\".*y=\"${yCoordinates[$i]}\"" "$qllFile")
    echo $tag
    id=$(echo $tag | sed -n 's/.*id="\([0-9]*\)".*/\1/p')
    moleculeIds+=($id)
done

for i in "${!digitalValues[@]}";
do
    echo "Molecule ${moleculeIds[$i]}a to be checked for a digital ${digitalValues[$i]}."
done

for i in "${!digitalValues[@]}";
do
        match="^${moleculeIds[$i]}a"
        line=$(eval "grep $match $qssFile")
        DOTaValue=$(echo $line | awk '{print $2}')

        if [[ ${digitalValues[$i]} -eq 0 ]]; then
            if (( $(echo "$DOTaValue < $zeroThreshold" | bc -l ) )); then  #a problem is found
                echo "Found a problem for molecule ${moleculeIds[$i]}a while checking it for digital 0. Exiting."
            else
                echo "Molecule ${moleculeIds[$i]}a correctly encodes a digital 0."
            fi
        else
            if (( $(echo "$DOTaValue > $oneThreshold" | bc -l ) )); then  #a problem is found
```

```
                echo "Found a problem for molecule ${moleculeIds[$i]}a while checking it for digital 1. Exiting."
            else
                echo "Molecule ${moleculeIds[$i]}a correctly encodes a digital 1."
            fi
        fi
done
```

Appendix B

Enhancing the SCERPA algorithm

This appendix includes the code developed to enhance SCERPA as a state-of-the-art research algorithm for studying mFCN.

B.1 Dynamic damping

The code implements the dynamic damping feature presented in Section 6.1 to promote faster analysis by favoring convergence.

Listing B.1. MATLAB source code implementing the dynamic damping in the main loop.

```matlab
if settings.autodamping == 1 % autodamping ON
    if scfStep==1 %at first step
        % in case of first step, the autodamping is statically
        % assigned to a standard value specified by the user or
        % by default
        deltaV_previous_step = Vout - preV_afterVoltageVariation;
        autodampingFirstStepValue = (1 - settings.autodampingFirstStep);
    else % not first step
        deltaV_current_step = Vout - preV_afterVoltageVariation;
        % in case of a normal step, the autodamping is
        % (deltaV_previous_step / deltaV_current_step) *
        % precision (how far is the solution wrt to ideal case with damping=1)
        autodampingValueVector = abs((deltaV_previous_step ./ deltaV_current_step));
        autodampingValue = min(autodampingValueVector);
        if (autodampingValue >= settings.autodampingPrecision)
            autodampingValue = settings.autodampingPrecision;
        else
            autodampingValue = autodampingValue * settings.autodampingPrecision;
        end
        deltaV_previous_step = deltaV_current_step; % assign last step deltaV to current step deltaV
    end
else % static damping
    damping = 1 - settings.damping;
end
if settings.autodamping == 2 % if real autodaming is on, apply the autodamping value
    if scfStep==1
        Vout = preV_afterVoltageVariation + autodampingFirstStepValue*(Vout - preV_afterVoltageVariation);
    else
        Vout = preV_afterVoltageVariation + autodampingValue*(Vout - preV_afterVoltageVariation);
    end
else
    Vout = preV_afterVoltageVariation + damping*(Vout - preV_afterVoltageVariation);
end
```

B.2 ToPoliNano clock waveforms

The code introduces the matched clock waveforms of ToPoliNano to regulate information propagation.

Listing B.2. MATLAB source code to enable ToPoliNano clock waveforms.

```matlab
clear variables
close all

generate_topo_clock = 1;
n_steps_topo = 20;
N_topo = 4;

%clock values definitions
clock_low = -2;
clock_high = +2;
clock_step = 3;

if generate_topo_clock == 1
    clock_step = N_topo + 1;
end

%layout (MagCAD)
file = 'circuit.qll';
circuit.qllFile = fullfile(pwd,file);
circuit.doubleMolDriverMode = 1;
circuit.magcadImporter = 1;

%drivers and clock
D0 = num2cell(+4.5*ones(1,clock_step*4));
D1 = num2cell(-4.5*ones(1,clock_step*4));
Dnone = num2cell(zeros(1,clock_step*4));

circuit.Values_Dr = {
    'a1_c'   +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5 +4.5
    'a1'     -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5 -4.5
};

%normal clock phases
pSwitch = linspace(clock_low,clock_high,clock_step);
pHold = linspace(clock_high,clock_high,clock_step);
pRelease = linspace(clock_high,clock_low,clock_step);
pReset = linspace(clock_low,clock_low,clock_step);

if generate_topo_clock == 1
    pHold = pHold(2:length(pHold)-1);
    pReset = pReset(2:length(pReset)-1);
end

pCycle = [pSwitch pHold pRelease pReset];

if generate_topo_clock == 1
    topo_pCycle1 = pCycle;
    topo_pCycle2 = circshift(pCycle, N_topo);
    topo_pCycle3 = circshift(pCycle, N_topo*2);
    topo_pCycle4 = circshift(pCycle, N_topo*3);

    seq_length = length(pCycle);
    num_repeats = ceil(n_steps_topo/seq_length);

    phase_1 = repmat(topo_pCycle1, 1, num_repeats);
    phase_1 = phase_1(1:n_steps_topo);
    circuit.stack_phase(1,:) = phase_1;

    phase_2 = repmat(topo_pCycle2, 1, num_repeats);
    phase_2 = phase_2(1:n_steps_topo);
    circuit.stack_phase(2,:) = phase_2;

    phase_3 = repmat(topo_pCycle3, 1, num_repeats);
    phase_3 = phase_3(1:n_steps_topo);
    circuit.stack_phase(3,:) = phase_3;

    phase_4 = repmat(topo_pCycle4, 1, num_repeats);
    phase_4 = phase_4(1:n_steps_topo);
    circuit.stack_phase(4,:) = phase_4;
else
    circuit.stack_phase(1,:) = [pCycle pReset pReset pReset];
    circuit.stack_phase(2,:) = [pReset pCycle pReset pReset];
    circuit.stack_phase(3,:) = [pReset pReset pCycle pReset];
    circuit.stack_phase(4,:) = [pReset pReset pReset pCycle];
end

%SCERPA settings
settings.damping = 0.6;
settings.verbosity = 0;
settings.dumpDriver = 1;
settings.dumpOutput = 1;
settings.dumpClock = 1;
settings.dumpEnergy = 1;
settings.evalConformationEnergy = 1;
settings.evalIntermolecularEnergy = 1;
settings.evalPolarizationEnergy = 1;
settings.evalFieldEnergy = 1;
settings.energyEval = 1;

%PLOT settings
plotSettings.plot_waveform = 1;
plotSettings.plot_3dfig = 0;
plotSettings.plot_1DCharge = 0;
plotSettings.plot_logic = 0;
plotSettings.plot_potential = 1;
plotSettings.plotSpan = 1;
plotSettings.fig_saver = 0;
plotSettings.HQimage = 0;

%copy outputh path from algorithm settings if specified by the user
if isfield(settings,'out_path')
    plotSettings.out_path = settings.out_path;
end

%%%%
diary on
this_path = pwd;
scerpa_path = fullfile('../');
cd(scerpa_path)
SCERPA('generateLaunchView', circuit, settings, plotSettings);
cd(this_path)
diary off
if isfield(settings,'out_path')
    movefile('diary',fullfile(settings.out_path,'logfile.log'))
end
```

Appendix C

Enhancing the ToPoliNano EDA framework

This appendix includes the code developed to enhance the ToPoliNano EDA framework as a emerging EDA platform to study molecular circuits based on mFCN.

C.1 Parallelizing the ToPoliNano simulator

The section provides the code to parallelize the ToPoliNano's function to compute new molecule state along simulation.

Listing C.1. Source code of the wrapper function `evaluateInteractionParallel` to synchronise parallel computation results during simulation.

```
void MolecularSimulationController::evaluateInteractionParallel() {
    auto node_it = boost::vertices(system_.graph());
    QFuture<void> future = QtConcurrent::map(node_it.first, node_it.second, molecularEvaluator(&system_,
     simulation_time_, simulation_step_, &molecules, damping_factor_));
    future.waitForFinished();
}
```

Listing C.2. Source code of the `molecularEvaluator` to compute new molecule state.

```
struct molecularEvaluator {
    molecularEvaluator(System<MolecularElement*, MolecularEdge*, AmorphicClockElement*>* system, double
     simulation_time, double simulation_step, QHash<QString, MoleculeParameters> *molecules, double
     damping_factor)
        : system_(system), simulation_time_(simulation_time), simulation_step_(simulation_step), molecules_(
     molecules), damping_factor_(damping_factor) { }

    void operator()(long node_it) {
        double vClock = 0;
        double v1 = 0, v2 = 0;
        QVector<double> neighbor_charge;
        auto current_desc = node_it;
        MolecularElement* current = system_->at(current_desc);
        // Do not compute step for input elements
        if(current->IsInput())
            return;
        Value current_value = current->getValue();
        for( auto clk : system_->clocks()) {
            Value clk_v = clk->getValueInPos(current->GetPosition());
            vClock += clk_v["Voltage"];
        }

        for (auto neighbor_it_pair = system_->inEdges(current_desc); neighbor_it_pair.first !=
     neighbor_it_pair.second; ++neighbor_it_pair.first) {
            auto edge_desc = *neighbor_it_pair.first;
            MolecularEdge* edge = system_->graph()[edge_desc];
            auto distances = edge->getAggregateChargeDistances();
            auto neighbor = system_->at(boost::source(edge_desc, system_->graph()));
            Value neighbor_value = neighbor->getValue();
            int neighbor_charge_number = molecules_->value(neighbor->getMoleculeName()).getNumberOfCharges();
            neighbor_charge.resize(neighbor_charge_number);
            for(int i = 0; i < neighbor_charge_number; i++) {
                neighbor_charge[i] = neighbor_value[QString("Q%1").arg((char)(i + 'a'))];
            }
            for(int i = 0; i < neighbor_charge_number; i++) {
                v1 += kVoltageFacroe * neighbor_charge[i] / distances[0][i];
                v2 += kVoltageFacroe * neighbor_charge[i] / distances[1][i];
            }
        }
        QVector<double> newState = molecules_->value(current->getMoleculeName()).getChargesValue(v1 - v2, vClock);
        Value newValue;
        for(int i = 0; i < newState.size(); i++) {
           newValue.AddValue(QString("Q%1").arg((char)(i + 'a')), newState[i] + damping_factor_ * (current_value[QString("Q%1").arg((char)(i + 'a'))] - newState[i]));
        }
        current->setNew_value(newValue);
    }

    System<MolecularElement*, MolecularEdge*, AmorphicClockElement*>* system_;
    double simulation_time_;
    double simulation_step_;
    QHash<QString, MoleculeParameters>* molecules_;
    double damping_factor_;
};
```

C.2 Introducing double-driver

The section lists the code that introduce the double-driver in the ToPoliNano simulator and FCNviewer.

Listing C.3. Source code portion of `molecularSimulationController()` method to generate double-driver cells.

```
QcaPin* pin = static_cast<QcaPin*>(item);
if(pin->direction() != QcaPin::OUTPUT) {
  for(int i = 0; i < 2; i++) {
      Geometry::Point center(physical_parameters_.offset + (item->pos().x() + 0.5 * i) * physical_parameters_.cell_distance,
                             physical_parameters_.offset + item->pos().y() * physical_parameters_.cell_distance,
                             physical_parameters_.offset + item->layer() * physical_parameters_.cell_distance);
      geometry = new MultiPoint(molecules.value(driverMoleculeName).getNumberOfCharges(), center);
      try {
          // done reading parameters: create the Element
          QVector<double> initial_value = molecules.value(driverMoleculeName).getChargesValue(kInitialInputVoltage, kInitialClockVoltage);
          Value value;
          for(int i = 0; i < initial_value.size(); i++) {
             value.AddValue(QString("Q%1").arg((char)(i + 'a')), initial_value[i]);
          }
          MolecularElement* element = new MolecularElement(geometry, driverMoleculeName, value);
          element->setNumberOfCharges(molecules.value(driverMoleculeName).getNumberOfCharges());
          element->defineChargePosition();
          element->setId(item->id() + (char)('a' + i));
          element->setIsInput(true);
          element->rotate(item->angle());
          element->setName(pin->name());
          system_.addElement(element);
     }
      catch (const std::runtime_error& e) {
          throw std::runtime_error("Error while creating an Element");
      }
  }
}
```

Listing C.4. Source code of `setInputValue()` function to assign inputs to double-driver cells during simulation.

```cpp
void MolecularSimulationController::SetInputValue(QString name, Value value) {
    for ( auto node_it = boost::vertices(system_.graph());
                node_it.first != node_it.second; ++node_it.first) {
        if(system_.at(*node_it.first)->getName() == name) {
            QChar id_letter = system_.at(*node_it.first)->getId().back();

            // first molecule driver.
            if(id_letter == (char) ('a')){
                double input = (value["Logic"] == 1) ? voltage_high_ : voltage_low_;
                QVector<double> state = molecules.value(system_.at(*node_it.first)->getMoleculeName()).getChargesValue(input, 2);
                Value inputValue;
                // Set the first driver molecule state charges according to the actual testbench value.
                for(int i = 0; i < state.size(); i++){
                   inputValue.AddValue(QString("Q%1").arg((char)(i + 'a')), state[i]);
                }
                system_.at(*node_it.first)->setValue(inputValue);
            }
            // second molecule driver as complementary of the
            // first molecule driver
            else {
                double input = (value["Logic"] == 1) ? voltage_low_ : voltage_high_;
                QVector<double> state = molecules.value(system_.at(*node_it.first)->getMoleculeName()).getChargesValue(input, 2);
                Value inputValue;
                // Set the second driver molecule state charges
                // according to the complementary testbench value.
                for(int i = 0; i < state.size(); i++)
                {
                   inputValue.AddValue(QString("Q%1").arg((char)(i + 'a')), state[i]);
                }
                system_.at(*node_it.first)->setValue(inputValue);
            }
        }
    }
}
```

Listing C.5. Source code portion of `parseQll()` function to instantiate two molecules when a driver is imported.

```cpp
QcaPin* pin = static_cast<QcaPin*>(item);
if (pin->direction() != QcaPin::OUTPUT) {
    int item_rotation = item->angle();
    QVector3D item_center(
        (item->pos().x() + 0.5f) * intermolecular_Dist,
        (item->pos().y() * intermolecular_Dist),
        item->layer() * intermolecular_Dist
    );
    for (int i = 0; i < 2; i++) {
        if (!item->getProperty(QString("disabled_%1").arg(char(i+'a')).toStdString().c_str()).toBool()) {
            float dx = item->getProperty(QString("xshift_%1").arg(char(i+'a')).toStdString().c_str()).toFloat();
            float dy = item->getProperty(QString("yshift_%1").arg(char(i+'a')).toStdString().c_str()).toFloat();
            float dz = item->getProperty(QString("zshift_%1").arg(char(i+'a')).toStdString().c_str()).toFloat();
            QVector3D offset((i-0.5f)*0.5f*intermolecular_Dist, 0, 0);
            QVector3D mol_center = item_center + rotateZ(offset, item_rotation) + QVector3D(dx, dy, dz);
            int rotation = item_rotation + item->getProperty(QString("angle_%1").arg(char(i+'a')).toStdString().c_str()).toInt();
            QString item_ID = QString::number(item->id()) + char(i+'a');
            drawing.insertMolecule(item_ID, mol_center, rotation);
            // find min and max x
            if(mol_center.x() < xmin) xmin=mol_center.x();
            else if(mol_center.x() > xmax) xmax=mol_center.x();
            // find min and max y
            if(mol_center.y() < ymin) ymin=mol_center.y();
            else if(mol_center.y() > ymax) ymax=mol_center.y();
            // find min and max z
            if(mol_center.z() < zmin) zmin=mol_center.z();
            else if(mol_center.z() > zmax) zmax=mol_center.z();
        }
    }
}
```

C.3 Enabling hierarchical layouts

The section provides with the code to simulate hierarchical layouts.

Listing C.6. Source code of `exploreHierarchicalLayout()` function call in `MolecularSimulationController()`.

```cpp
if(item->type()!=QcaItem::PIN) {
    if(item->type()==QcaItem::COMPONENT) {
        QcaComponent* component = static_cast<QcaComponent*>(item);
        QList<QcaItem*> list = component->items();

        // generate component id according to its x, y, layer
        // coordinates in the format x.y.layer
        QString component_id = QString::number(component->pos().x()) +
          "." + QString::number(component->pos().y()) +
          "." + QString::number(component->layer());

        exploreHierarchicalLayout(list,
                                  component->pos().x(),
                                  component->pos().y(),
                                  component->layer(),
                                  settings,
                                  component_id,
                                  driverMoleculeName);
    }
}
```

Listing C.7. Source code of `exploreHierarchicalLayout()`. recursive function to explore hierarchical layouts.

```cpp
void MolecularSimulationController::exploreHierarchicalLayout(QList<QcaItem*> list, double
     x_hierarchical_offset, double y_hierarchical_offset, double layer_hierarchical_offset, const
     MQCAFloorplanParameters &settings, QString component_id, QString driverMoleculeName) {
MultiPoint* geometry;
foreach (QcaItem * item, list) {
if(item->type()!=QcaItem::PIN) {
    if(item->type()==QcaItem::COMPONENT) {
        QcaComponent* component = static_cast<QcaComponent*>(item);
        QList<QcaItem*> list = component->items();
        exploreHierarchicalLayout(list, x_hierarchical_offset + component->pos().x(), y_hierarchical_offset +
      component->pos().y(), layer_hierarchical_offset + component->layer(), settings, component_id + "_" +
     QString::number(component->pos().x()) + "." + QString::number(component->pos().y()) + "." + QString::
     number(component->layer()), driverMoleculeName);
    }
    else {
        for(int i = 0; i < 2; i++) {
            qDebug() << item->getProperty(QString("disabled_%1").arg((char)(i + 'a')).toStdString().c_str()).
     toBool();
            qDebug() << item->getProperty(QString("angle_%1").arg((char)(i + 'a')).toStdString().c_str()).
     toInt();

            if(!item->getProperty(QString("disabled_%1").arg((char)(i + 'a')).toStdString().c_str()).toBool()
     ) {
                double dx = item->getProperty(QString("xshift_%1").arg((char)(i + 'a')).toStdString().c_str()
     ).toDouble() * 1e-12;
                double dy = item->getProperty(QString("yshift_%1").arg((char)(i + 'a')).toStdString().c_str()
     ).toDouble() * 1e-12;
                double dz = item->getProperty(QString("zshift_%1").arg((char)(i + 'a')).toStdString().c_str()
     ).toDouble() * 1e-12;

                Geometry::Point center(physical_parameters_.offset +
                                       (item->pos().x() + 0.5 * i + x_hierarchical_offset) *
     physical_parameters_.cell_distance + dx,
                                       physical_parameters_.offset +
                                       (item->pos().y() + y_hierarchical_offset) * physical_parameters_.
     cell_distance + dy,
                                       physical_parameters_.offset +
                                       (item->layer() + layer_hierarchical_offset) * physical_parameters_.
     cell_distance + dz);
                geometry = new MultiPoint(molecules.value(item->name()).getNumberOfCharges(), center);
                try {
                    // done reading parameters: create the Element
                    QVector<double> initial_value = molecules.value(item->name()).getChargesValue(
     kInitialInputVoltage, kInitialClockVoltage);
                    Value value;
                    for(int i = 0; i < initial_value.size(); i++) {
                       value.AddValue(QString("Q%1").arg((char)(i + 'a')), initial_value[i]);
                    }
                    MolecularElement* element = new MolecularElement(geometry, item->name(), value);
                    element->setNumberOfCharges(molecules.value(item->name()).getNumberOfCharges());
                    element->defineChargePosition();

                    // generate second portion of element id
                    // according to its x, y, layer coordinates
                    // in the format x.y.layer
                    QString element_id = QString::number(item->pos().x()) + "." + QString::number(item->pos().y()) + "." + QString::number(item->layer());

                    // generate complete element id as the concatenation
                    // of any component of which the element is part of and the element id itself
                    QString full_element_id = component_id + "_" + element_id;

                    // append a letter to the id in order to identify
                    // the single molecule in the cell.
                    element->setId(full_element_id + (char)('a' + i));

                    element->setIsInput(false);
                    element->rotate(item->getProperty(QString("angle_%1").arg((char)(i + 'a')).toStdString().c_str()).toInt());
                    system_.addElement(element);
                    for(auto clk : system_.clocks()) {
                        if(clk->getName() == QString("clock%1").arg(item->getProperty("phase").toInt()+1))
                            clk->addInfluenced(center);
                    }
                }
                catch (const std::runtime_error& e) {
                    throw std::runtime_error("Error while creating an Element");
                }
            }
        }
    }
}
```

C.4 Consistent visualization in the FCNviewer

The section provides with the code to make the FCNviewer consistent with SCERPA's viewer.

Listing C.8. Source code of `getChargePositions()`. method to set molecule coordinate.

```
QVector3D Molecule::getChargePosition(molecule_t molType, int index, int angle){
    QVector3D pos;
    switch(molType){
    case molecule_t::BISFERROCENE:
        switch (index) {
            case 0: pos.setX(-9.4f); pos.setY(-506.2f); pos.setZ(-362.2f); break;
            case 1: pos.setX(-9.4f); pos.setY(+508.3f); pos.setZ(-358.8f); break;
            case 2: pos.setX(-75.5f); pos.setY(-1.1731f); pos.setZ(+313.3515f); break;
            case 3: pos.setX(+40.9f); pos.setY(-5.3777f); pos.setZ(+1177.6298f); break;
        default:
            qDebug() << "ERROR: Molecule::getChargePosition(molecule_t molType, int index, int angle): wrong index";
            return QVector3D();
        }
        break;
    default:
        qDebug() << "ERROR: Molecule::getChargePosition(molecule_t molType, int index, int angle): unsupported molecule type";
        return QVector3D();
    }
    return rotateZ(pos, angle);
}
```

Listing C.9. Source code of `resetRotation()` function to have a top view of the layout.

```
void GLWidget::resetRotation() {
    xRot = 180.0f; // to have a clearer above view of the two encoding DOTs
    yRot = zRot = 0.0f;
}
```