



ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR



Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering
(32.nd cycle)

Enabling Field-Coupled Nanocomputing Multi-Technological Circuits: Design, Simulation and Validation

From ToPoliNano and MagCAD to FCNS

Umberto Garlando

* * * * *

Supervisor

Prof. Maurizio Zamboni

Doctoral Examination Committee:

Prof. M. Becherer, Referee, Technische Universität München - TUM

Prof. G. Csaba, Referee, Pazmany Peter Catholic University

Prof. M. Ottavi, Referee, Università degli Studi di Roma "Tor Vergata"

Prof. G. Piccinini, Referee, Politecnico di Torino

Prof. R. Wille, Referee, Johannes Kepler University Linz

Politecnico di Torino
March 16, 2020

This thesis is licensed under a Creative Commons License, Attribution - Noncommercial-NoDerivative Works 4.0 International: see www.creativecommons.org. The text may be reproduced for non-commercial purposes, provided that credit is given to the original author.

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

.....

Umberto Garlando
Turin, March 16, 2020

Summary

The Moore's law defined the trend for digital circuits over the last 50 years. The number of transistors inside a single chip has doubled every year and a half thanks to dimensions scaling. However, this trend is reaching an end due to physical limitations. Dimensions and power density are the main issues. Transistors gate length below 5 nm will be probably the last technology node of the CMOS (complementary metal oxide semiconductor). Furthermore, the power density inside the chip, increasing with the performance of the circuit, is a big limit for the current technology. For these reasons new technologies are being studied in order to find possible alternatives. In particular, the QCA (quantum-dot cellular automata) technology is one of the most promising. Different implementation of this technology are currently studied by the research community. Molecular and magnetic QCA are addressed in this work. These technologies seems promising thanks to a lower power consumption, compatibility with CMOS fabrication process and their non-volatile nature. In fact, the concept of logic-in-memory can be explored using these technologies. However, their performance is not yet comparable with the CMOS technology.

Technologists focus on the design of the single device and its characteristics. However, even at the early stage of development, a circuit level exploration is necessary. This kind of analysis can be used to verify the behavior of new technologies inside digital circuits. Furthermore, parametric analysis are mandatory to understand the effect of modification of properties of the devices. Another key aspect is the research of architectural optimization specifically designed to take advantages from the technological properties. These kind of explorations need software tools, like CAD (Computer Aided Design) and EDA (Electronic Design Automation), in order to speed up the process.

No commercial tools are available capable of handling these kind of devices. To solve this issue the ToPoliNano framework was developed at the Politecnico di Torino. This framework, comprised of different tools, enables the analysis of emerging technologies giving almost the same flow available for CMOS. In particular two tools are part of the framework. In this work an overview of the entire framework

is presented, and a new simulator is introduced. Currently the framework is composed by ToPoliNano and MagCAD.

ToPoliNano is a tool that starts from a HDL (Hardware description Language) description of a circuit and performs the physical place & route based on the technology constraints. Different optimization algorithms are available in the tool and the user can also modify some properties of the technology. It supports hierarchical layout and it is possible to select different approaches in order to handle the hierarchy of the circuits. The tool is not focused on logic synthesis, however VHDL and Verilog are supported. After performing the layout of the circuit it is possible to run simulations to verify the layout thanks to a behavioral simulation engine. At the moment of writing it supports iNML technology.

MagCAD on the contrary is a custom layout editor. Firstly, it is possible to design a new circuit simply placing the technology building blocks inside a drawing area. It is possible also to open a ToPoliNano layout file and modify the circuit. This tool supports hierarchical layouts too. Furthermore, 3D circuits can be designed. After the design phase it is possible to extract a VHDL netlist based on the technological elements. Specifically designed model are embedded in the generated file and are solved during the simulation, which can be performed using commercial HDL simulators. The HDL netlists extraction is performed using the FunCoDe (Function and Connection Detection) algorithm. This tool fully supports two implementations of magnetic QCA (iNML and pNML) and MolQCA. Furthermore, the entire framework is based on general tools and technological plugin. In this way new technology could be easily added.

The main contribution described in this work is FCNS (Field coupled nanotechnologies simulator). This tool, that will become part of ToPoliNano, is a new simulator capable of handling different FCN technologies. A tech-independent core was developed following the idea of the framework. Furthermore, three simulation engines were added to demonstrate the flexibility of the system. MolQCA and iNML can be simulated using FCN. Another strength of FCN is the possibility to define different simulation engines for the same technology: iNML can be handled with both behavioral or physical engines. In order to verify the simulator capabilities, several circuits based on the different technologies were simulated and the results compared to state of the art simulators. FCNS performance was compared with the other simulators: FCNS is in general faster than the reference simulators giving compliant results.

The ToPoliNano framework can be used to perform circuit level exploration of digital circuits. The results obtained with this kind of analysis can be used to define new trends for the development of those technologies.

Acknowledgements

Vorrei ringraziare tutte le persone che mi sono state vicine in questi tre anni. Tutti i membri del VLSI, che sono stati come una seconda famiglia. Tra di loro un ringraziamento particolare a Giulia, un'amica preziosa con cui ho condiviso molto. Maurizio, il mio vicino di scrivania per i primi 2 anni. Luca, che mi ha aiutato a capire qualcosa dei magnetini, e Yuri, senza il quale non avrei mai potuto sviluppare la parte relativa alle molecole. Gianni, Maurizio e Stefano, che dopo avermi temuto come esercitatore sono diventati colleghi, ma soprattutto ottimi amici. Un ringraziamento infinito a Fabrizio, per le ore in cui abbiamo lavorato insieme e per le & dimenticate che cambiano tutto. Grazie a Giovanna e Marco, sempre disponibili e di grande aiuto. Grazie a Mariagrazia e Maurizio, non solo tutor e capi, ma soprattutto esempi da seguire.

Ringrazio anche tutti gli amici non addetti ai lavori. Marco, Chiara, Della, Simone e Tita, rappresentate quelle amicizie che non finiranno mai. Ovviamente grazie alla mia famiglia, che c'è sempre stata. Infine, un grazie particolare a Denise, che mi ha catturato all'inizio di questa avventura e che ora è una parte imprescindibile della mia vita.

A chi...

*... crede in ciò che fa nonostante tutto,
perchè solo in questo modo si arriva
all'obiettivo.*

*... crede negli altri, perchè in fondo da
soli non si è nessuno.*

*... ha creduto in me, perchè dove sono
oggi è anche grazie a voi*

Umberto

Contents

List of Tables	X
List of Figures	XI
1 Introduction	1
2 Background	3
2.1 Molecular QCA	6
2.2 Magnetic QCA	7
2.2.1 iNML	7
2.2.2 pNML	9
I ToPoliNano Framework	13
3 ToPoliNano	17
3.1 HDL Parsing	18
3.2 Layout Phase	19
3.2.1 fiction Integration	23
3.3 Simulation	25
4 MagCAD	29
4.1 FUNCTION AND CONNECTION DETECTION Algorithm	31
4.1.1 Connection Detection	34
4.1.2 Function Detection	38
4.1.3 Handling iNML Technological Issues	39
4.1.4 Handling pNML Technological Issues	41
4.1.5 Performance and verification	41
4.2 Technological properties change	49
4.3 Molecular plugin	50

II	Field-Coupled Nanocomputing Simulator	55
5	Physical simulators	59
5.1	Micro-magnetic Simulators	59
5.2	Molecular Simulators	60
5.3	QCADesigner	61
6	FCNS	63
6.1	General Flow	64
6.1.1	Simulation Loop	67
6.2	Class organization	70
6.3	Interface Classes	72
7	Technology specific implementations	79
7.1	iNML	79
7.1.1	Behavioral Algorithm	81
7.1.2	Physical Algorithm	87
7.2	MolQCA	92
8	Results and Performance Analysis	101
8.1	iNML Behavioral	103
8.2	iNML Physical	108
8.3	MolQCA	125
9	Conclusion and Future Works	131

List of Tables

2.1	Majority voter truth table.	5
4.1	Verification example of circuit c432. Inputs are applied to both the model and the FUNCODE generated netlist, outputs are collected and analyzed.	44
4.2	Generation algorithm timing report using different iNML architectures with an increasing number of elements.	46
4.3	Generation algorithm timing report using different pNML architectures with an increasing number of elements.	47
7.1	iNML behavioral values.	84
8.1	Horizontal wire results.	112
8.2	Vertical wire two stacked magnets results.	114
8.3	Vertical wire three stacked magnets results.	115
8.4	“L” shaped wire with two vertical stacked magnets results.	117
8.5	“L” shaped wire with three vertical stacked magnets results.	118
8.6	“Coupler” results.	120
8.7	MV results.	122
8.8	Simulation time, in seconds, comparison among FCNS and Mumax3.	122

List of Figures

2.1	Two QCA cell encoding binary information.	3
2.2	Basic structures in QCA circuits. (A) shows a wire. (B) is an inverter. (C) shows a majority voter.	4
2.3	QCA wire divided in three clock zones. Each clock zone is reset independently and the information is propagated.	5
2.4	Bis-ferrocene molecule. (A) structural representation. (B) Ball and stick representation.	6
2.5	3D view of a molecular wire. In the picture, it is possible to see the gold substrate where the molecules are anchored and the top electrode use to generate the clock field. The other molecules are responsible for the switching field.	7
2.6	A) iNML basic cells; B) Chain of nanomagnets antiferromagnetically coupled; C) Vertically aligned nanomagnets ferromagnetically coupled; D) iNML three-phase clock system;	8
2.7	NML: (A) magnetic binary representation; (B) Notch; (C)(D) pNML wire; (E)(F) pNML majority voter; (G) pNML clock mechanism. (H) Clock coil representation.	10
2.8	Schematic representation of the ToPoliNano framework.	15
3.1	Schematic representation of the ToPoliNano tool flow.	17
3.2	Graph representation in memory of a hierarchical 4-bits ripple carry adder. The top-level is the root of the tree. 2-bits RCA instantiated in the circuit are the trunks of the tree with the four instances of full adders. Finally, the leaves are composed by the logic gates: AND, OR, Inverter and MV.	19
3.3	Internal representation of the 2-to-1 multiplexer. Each circle in the picture is a node of the ToPoliNano data structure. Edges show inputs and outputs of each node.	20
3.4	The multiplexer structure after path balance and coupler management. The number of nodes is increased, together with the number of levels. This structure will be used to perform the actual <i>Place&Route</i>	21

3.5	Final layout of the 2-to-1 multiplexer: input and output pins are the light blue elements with the name superimposed. Magnets of different colors belong to different clock zones. Logic gates are differently colored and it is possible to see the slanted edge of the central magnet.	22
3.6	ToPoliNano windows showing the parameters related to fiction calls. The user can check the fiction box and specify the maximum time available to complete the layout phase before going back to the ToPoliNano engine.	24
3.7	Schematic view of the toll structure. fiction is integrated inside ToPoliNano and it is called during the layout phase. The exact algorithm is used to manage components in the hierarchy.	25
3.8	topo windows showing the parameters related to the simulation. The user can select the clock waveform specification and the simulation engine. Furthermore, the simulation resolution can be modified. . .	26
4.1	Technology selection window. On the right the settings for iNML technology modifiable by the user.	29
4.2	Mainwindow of MagCAD	30
4.3	Design methodology in which the FUNCODE algorithm was introduced	31
4.4	FUNCODE flow chart	33
4.5	A) iNML elements and corresponding available connections. Inputs are blue, while outputs are the red arrows. B) pNML elements and relative connections. Blue crosses show input coming from another layer, while red circles are output interconnections towards adjacent planes.	34
4.6	Example of how to circuit is translated in memory. A) Example of iNML coupler; B) Example of translation using two matrices; C) Example of translation using two vectors.	36
4.7	Elements connectivity examples. A) shows a portion of a pNML circuit with two elements and one input pin. B) highlights the connection available for the elements and C) shows the correct connection detection. D) represents the same circuit with extra input. E shows the connection available and F) is an example of output-output error. G) shows a portion of the iNML circuit where an inverter has two neighboring pins. H) highlights the error due to multiple input connections to a single input element. I) is a pNML corner with a pin placed where there are no available connections, generating the error in L).	37
4.8	Methodology adopted to verify the correct behavior of the proposed FUNCODE algorithm.	42

4.9	A) Example of iNML layout; B) Excerpt of the generated VHDL that show the automatically detected functional blocks; C) Simulation of the sample circuit that take into account the latency introduced by the target technology.	43
4.10	A) iNML FUNCODE execution time as a function of the total number of elements; B) pNML FUNCODE execution time as a function of the total number of elements.	45
4.11	Technology parameters selection. (A) iNML and (B) pNML.	51
4.12	MolQCA property widget.	52
4.13	Highlight of the distance among the molecules. Each dot, in the same molecule or of two different molecules is distant at least 1 nm from its neighbors.	53
6.1	Schematic overview of FCNS	63
6.2	Detailed flow of FCNS. Each step is executed in sequence, the last one is the simulation loop reported on the right.	65
6.3	Interaction radius example. Each value results in a different amount of interacting elements, reported with the number near each circle.	66
6.4	Example of circuit and the corresponding graph. The circuit is a short wire composed of four magnets and two pins. In the graph, obtained with $radius = \infty$ each node is an element and the arrows show the interactions. Input and output pins do not have incoming edges: they can not be influenced.	67
6.5	Class diagram of the interface classes. Here the class used to perform the operations are shown.	71
6.6	Class diagram of the data classes. These elements are used to store the data during the simulation.	72
6.7	Example of two objects of class <i>Value</i> . Each one stores four different quantities, named Q1, Q2, Q3, and Q4. The difference is performed and the resulting value has the same number of quantities.	74
6.8	Direction and distance example with the formula used to compute them.	76
7.1	Class diagram of the rectangular prism and the base class <i>Geometry</i>	80
7.2	Geometrical representation of the rectangular prism. Segments and points stored in the correspondent class are highlighted.	80
7.3	3D and top view of an example where three elements are placed in a region with two clock zones. In this case, element 1 and element 2 are influenced by Clock 1 while element 3 is influenced by clock 2.	81

7.4	(A) wire and an inverter with the vertical grid; the vertical lines are associated with the middle among the integer coordinates. Each element is translated into the corresponding object and placed in its position. This configuration results in a nonworking inverter: an even number of elements is not inverting the information. In the other case, an extra element (the red one) is added and the position of two elements is changed. In this way, each element is influenced only by one neighbor as highlighted in the graph shown in (B). . . .	83
7.5	Example of three clock zones spanning over four vertical lines of the grid. Each zone is four positions wide but an offset of 0.5 is applied. Zone 1 starts from -0.5 and ends in 3.5 where the new zone begins.	85
7.6	Examples of interaction evaluation in the behavioral simulation. (A) shows a simple wire. The central magnet is evaluated: the one on the left is stable, the one on the right is in reset. Preferred magnetization is zero and antiferromagnetic coupling is present, resulting in total interaction equal to -2 . Since the number of neighbors is equal to two, the new value will be directly -2 . (B) shows an OR gate. The central slanted magnet is evaluated: ferromagnetic coupling and preferred magnetization sums up to -2 . Given three neighbors the value is normalized to -1	86
7.7	Examples of interaction evaluation in the behavioral simulation for MV layout. (A) shows an example where the sum of the neighbors is 0. Therefore the new value will be equal to the previous one. (D) is similar to the previous configuration but here the neighbors' influence sums up to -2 . In this case, this value is normalized due to the number of interacting elements.	87
7.8	Example of evolution of an MV in iNML behavioral simulation. The elements in the circuit start from a reset condition(all zeroes). The input values are placed on the left of the elements. The first step the magnets near the inputs (red values) switch with antiferromagnetic coupling. Since the number of neighbors is 2 they assume immediately a stable value. In step number two, top right and bottom right elements reach a new stable value, while the central one will assume value equal to 1 ("weak 1") since its number of neighbors is greater than 2. In the next step other two magnets, in red, get the new stable value. The central one is in the same condition. In the final step, the central magnet is the only one that is evaluated, and therefore it is promoted to its stable version.	88

7.9	Schematic view of the multithreading approach present in the simulator. The <code>QtCOncurrent::Map</code> functions launches a thread, depending on the number available on the system, and assign to each thread the <i>Compute New Value</i> operation of a single element. Each thread evaluates the H_{eff} and calls the stepper for the ODE resolution. The <code>QFuture</code> waits that all the threads finish their operation and then the simulator moves to the next step.	91
7.10	Representation of the data structure used to store the molecules trans.characteristics. In this example, the molecule has four dots. For each dot a set of curves, one for each clock value, is available. The curves give the charge over the input voltage. In order to store the curves, the same input voltage is used to sample all the characteristics and define a vector: the size is equal to the number of charges and each element is a charge value. All the vectors resulting from this sampling are named with the corresponding input voltage and stored in an hash-map. The keys of the map are the clock voltages. Each key is associated with a vector where all the input voltages grouped charges are stored.	94
7.11	Flow chart of the molecular simulation engine. The main simulation loop is composed of a nested loop that is used to evaluate the stability. A new set of inputs and clock values is loaded only after reaching the stability.. . . .	98
7.12	Example of interaction of two molecules. The one on the left is evaluated considering the effects of the one on the right. Each dot of the influencing molecule generates a potential on each of the logic dots of the influenced one. The difference among these potentials is used as the input voltage of the molecule.	100
8.1	Waveforms obtained with the behavioral engine. Each waveform is the logic value of an element in the different time steps.	101
8.2	Each graph represents one of the components of the magnetization vector. Each magnet is associated with the same color in all the graphs. The legend is used to identify the magnets. The number of steps is present on the X-axis.	102
8.3	Each graph is associated with the charge of an aggregate charge. The charges are expressed in atomic units. on the fourth graph also the clock sources, square waves ranging from -2 to 2 volts, are showed.	103
8.4	Horizontal wire composed of four magnets. Each element is associated with an index that will be used in the simulation graph.	104
8.5	Waveforms obtained simulating the horizontal wire.	104
8.6	Layouts of the iNML logic gates obtained with the slanted edge magnets. (A) OR gate. (B) AND layout.	105
8.7	Waveforms obtained with the behavioral simulation of the AND gate.	105

8.8	Waveforms obtained with the behavioral simulation of the OR gate.	106
8.9	Layout of the majority voter.	107
8.10	Waveforms obtained with the behavioral simulation of the majority voter gate.	107
8.11	Layout of a wire followed by an inverter. The background of the inverter is different since it belongs to another clock zone.	108
8.12	Waveforms resulting from the simulation of the wire followed by the inverter. The magnets resulting from the inverter expansion and the extra one inserted are visible.	108
8.13	Final layout of the 2-to-1 multiplexer: input and output pins are the light blue elements with the name superimposed. Magnets of different colors belong to different clock zones. Logic gates are differently colored and it is possible to see the slanted edge of the central magnet.	109
8.14	Waveforms of the mux simulation. Inputs and clock sources are shown, together with the output magnet. The circles are used to highlight the latency due to the clock mechanism.	110
8.15	Shape of the clock signal applied to all the circuits here described. The duration of the different portions is reported in the figure. . . .	110
8.16	Graph of the physical simulation of the horizontal wire.	112
8.17	(A) Vertical wire with two magnets. (B) vertical wire with three magnets.	113
8.18	Graph of the simulation of the two magnets vertical wire. Both the elements assume the opposite value with respect to the input. . . .	113
8.19	Simulation results of a non working configuration of the three elements vertical wire. Two out of the three elements have wrong direction.	114
8.20	(A) “L” shaped wire. Two vertical stacked magnets and other three in order to get length equal four. (B) “L” shaped wire. Three vertical stacked magnets and the remaining to complete the horizontal wire.	116
8.21	Working simulation of the two magnets “L” shaped wire.	116
8.22	Wrong simulation of the three magnets “L” shaped wire.	117
8.23	Coupler structure used to double the fan-out.	118
8.24	Simulation waveforms of the coupler. All the magnets are oriented correctly.	119
8.25	Majority voter structure with magnet dimension changed in order to implement the virtual clock principle.	120
8.26	Simulation waveforms of the MV. Inputs are equal to “010”.	121
8.27	Example of circuit used to test the performance of the simulator. . .	123

8.28	Performance graph comparing FCNS and Mumax3. Different mesh grid dimensions are used in Mumax3. The dashed line shows the performance of FCNS fixing the interaction radius at $2e - 6$ m. On the X-axis, the number of elements is reported. It can be computed multiplying the dimensions of the circuits: $4*4$ results in 16 elements, while $16*16$ is associated with 256 elements. The “x” marks on the chart refer to the measured circuit. On the Y-axis the elapsed time in seconds.	124
8.29	Molecular wire structure. Twelve molecules are placed in the same clock zone.	125
8.30	Simulation results of the horizontal wire.	126
8.31	Simulation results of the horizontal wire. Only the stable step are used in the plots.	127
8.32	MagCAD layout of the inverter based on MolQCA technology. . . .	128
8.33	Simulation results of the molecular inverter. Both input values are showed. Solid and dashed lines are used to show the couples of molecules. In each couple, the molecules have the same logic value but the charges are slightly different. This is due to a non symmetric behavior of the molecules.	128
8.34	Simulation results of the molecular MV. One molecule per input wire is plotted. Two molecules, solid and dashed, are used to show the final output.	129
8.35	MagCAD layout of the MV based on MolQCA technology.	130

Chapter 1

Introduction

The scaling of CMOS technology is reaching its limit. The trend described by Moore's law will be no longer valid in the future. Transistor dimension and power dissipation are the main physical limitation arising. New technologies are being studied in order to find possible alternatives to CMOS. Emerging technologies, also called beyond CMOS technologies [26], are based on completely new paradigms and physical properties. A vast variety of technologies can be identified in this category: quantum computing [29], carbon nanotubes [4], quantum-dot cellular automata (QCA) [30] and many others. This work will focus on QCA and in particular on field-coupled nanocomputing (FCN) technologies. In FCN there is no current flowing among the elements, as in CMOS, but the information propagation is due to field coupling among the elements. The main advantage of this technology is the reduction of power consumption. Furthermore, implementations based on nano-magnets or molecules are based on non-volatile building blocks. This last characteristic can be exploited to develop circuits with the logic in memory principle. However, there are still severe limitations in FCN, like the performance. The operating frequency of circuits based on these technologies is limited to few MHz. Furthermore, the fabrication process is at a preliminary stage. Considering these issues, FCN technologies are not yet ready to replace CMOS technology. Technologists are studying these technologies in order to find new materials or processes to improve performance. This kind of research could lead to useless results without a complementary circuit-level analysis of the technological elements. Furthermore, new technologies are based on different paradigms, and the well-known principle of CMOS, both regarding architecture and layout, are not applicable in this case. The feedback obtained by a circuit-level exploration could be a huge help for technologists, helping to define a roadmap for the research. Even at the early stage of the research, the design of relatively complex circuits and architectures is mandatory to prove the feasibility of an emerging technology. Increasing circuit complexity arises the need for CAD and EDA tools. However, this kind of tool is not commercially available. This is the main motivation for this work.

In this work, a CAD framework is presented. The ToPoliNano framework is a complete framework enabling the FCN circuits in terms of design, simulation, and validation. The ToPoliNano project started before the beginning of this work. In fact, different parts of the framework were not developed during this thesis project. However, a general description of the framework is given since the developed features are strictly related to the framework. In particular, this thesis covers the development of some functionality of ToPoliNano and MagCAD, the two tools composing the framework. However, the main contribution here presented is FCNS (field-coupled nanocomputing simulator). FCNS is a general simulator, capable of handling different technologies. Furthermore, it was designed in order to be easily extended to new technologies based on the field coupling principle. Currently, the only options available to simulate FCN circuits are physical simulators. These tools are based on complex models of the physical properties of the technological elements and on numerical methods. Their huge complexity results in a limitation of the number of elements composing the simulated circuits. FCNS is based on simplified models and approximations. In this way, the simulation time and computational power demand are reduced. As a trade-off, the accuracy is also reduced. Furthermore, the simulator is designed in a way so that it is possible to define different levels of approximation for the supported technologies. For example, it is possible to perform a very fast “fully digital” simulation to verify the logical correctness and later perform a “physical” to validate the circuit.

This approach has been used also with CMOS technology. A circuit is verified before with logical simulators, and only after a behavioral verification, it is then simulated with low-level simulators. In this way the simulation phase is simplified in the beginning, avoiding the need for long and complex physical simulation when the architecture is designed.

The structure of this thesis is organized as follows. Firstly, a technological background is presented. A general introduction to QCA is given, focusing then on molecular and magnetic implementations of the technology. After the technological background, two parts are present. In the former, part I, the ToPoliNano framework is introduced and described. The two tools, ToPoliNano and MagCAD are described in this part, in chapter 3 and chapter 4. Particular focus is given to the new features. Verilog parser and the integration of *fiction*, another CAD tool for FCN, are the feature related to ToPoliNano. The FunCoDe algorithm, the technology property change, and the molecular plug-in are the main features developed for MagCAD. In the latter part, part II, the new simulator is introduced and deeply analyzed. Firstly, an introduction to the available physical simulators for FCN is presented in chapter 5. Then, the technology-independent core of the simulator is described in chapter 6. The different technological implementations initially supported by FCNS are described in chapter 7. Finally, the results of the simulations and performance analysis are presented in chapter 8.

Chapter 2

Background

The scaling of transistor dimensions predicted by Moore's Law will be no longer valid quite soon. Physical limits are being reached and it will not be possible to reduce gate dimension in the future [27]. Furthermore, doubling the number of transistors inside a circuit leads to power problems [11]. New technologies, so-called "Beyond CMOS" technologies, are currently researched to find a possible solution for those problems. Among these technologies, QCA (Quantum-dot Cellular Automata) [30][14] seems to be the most promising. In QCA technology binary information is encoded inside bi-stable cells. Each cell has four quantum-dots and two free electrons: the electrons will occupy the dots to minimize the energy. Figure 2.1 shows the configuration of two cells. The idea behind QCA is to use field

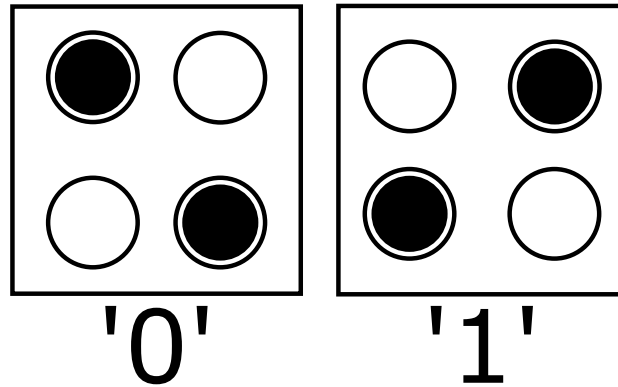


Figure 2.1: Two QCA cell encoding binary information.

coupling effects (both electric and magnetic) to propagate information. The basic cells are used to build circuits. Simply placing cell one near the other, a wire is formed as showed in Fig. 2.2.a. The Majority Voter (MV) is the basic logic gate of QCA. When a cell is surrounded by an odd number of other input cells the output will be the majority of the inputs. Fig. 2.2.b shows an example of three-inputs MV. Classical logic function, *AND* & *OR*, are realized fixing one of the inputs of

the MV. Table 2.1 shows the truth table of an MV. A logic *AND* is obtained if an input is set to '0'. On the contrary, a logic *OR* is obtained if '1' is used as fixed input. An inverter gate is shown in 2.2.c. By using wires, MVs and inverters it is possible to implement every boolean expression.

Since no current flows in the circuit, something is needed to define the direction

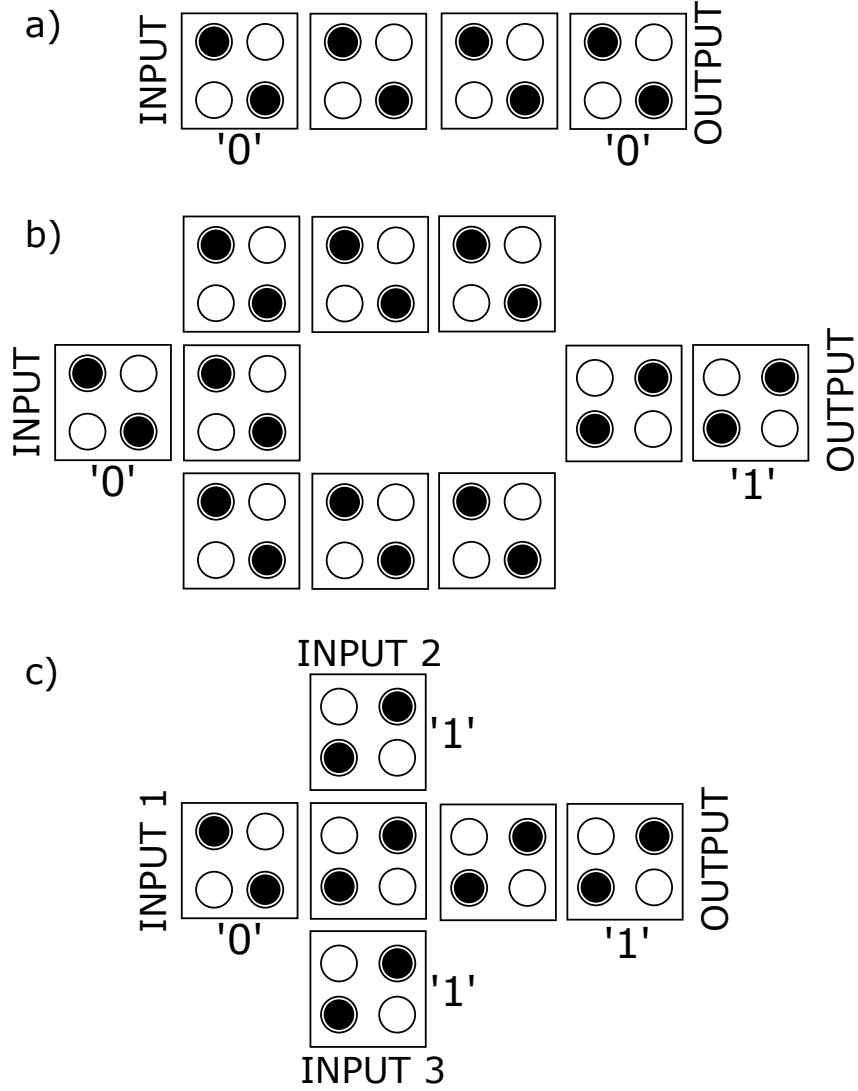


Figure 2.2: Basic structures in QCA circuits. (A) shows a wire. (B) is an inverter. (C) shows a majority voter.

of information propagation. A clocking mechanism is used for this purpose. The idea is to divide the circuit into clock zones, and then force the elements of a clock zone in an unstable state, called reset. By applying a multi-phase clock signal, it is possible to reset alternatively different clock zones. The information will be

therefore propagated when the clock signal is removed. The cells will reach a new stable state, that will depend on the value of the neighbors. Figure 2.3 shows an example of a circuit with a three-zone clock mechanism. Different clock schemes were proposed in literature [49][42][2][23]. The clock signal is needed also for the following reason. As mentioned before, the QCA cells have two stable configurations. It is impossible to change the state with the effect of surrounding elements. The clock signal will reduce the energy barrier of the cell and enable therefore the unstable state. The energy needed for the clock mechanism is the only one needed by the circuit. Differently from CMOS technology, no energy is consumed to keep the value or during state switching.

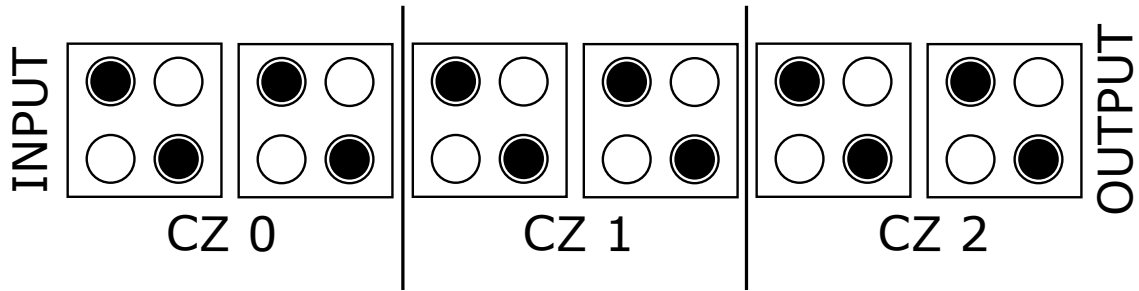


Figure 2.3: QCA wire divided in three clock zones. Each clock zone is reset independently and the information is propagated.

Table 2.1: Majority voter truth table.

A	B	C	Output
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

Different implementations of QCA are available in literature. This work will focus on a molecular and two different magnetic implementations. In this chapter, a technological background will be given.

2.1 Molecular QCA

A possible implementation of QCA is based on molecules. Molecular QCA (MQCA) is a promising implementation. The nano-scale dimension of its building elements and the possibility to work at room temperature are the key aspects.

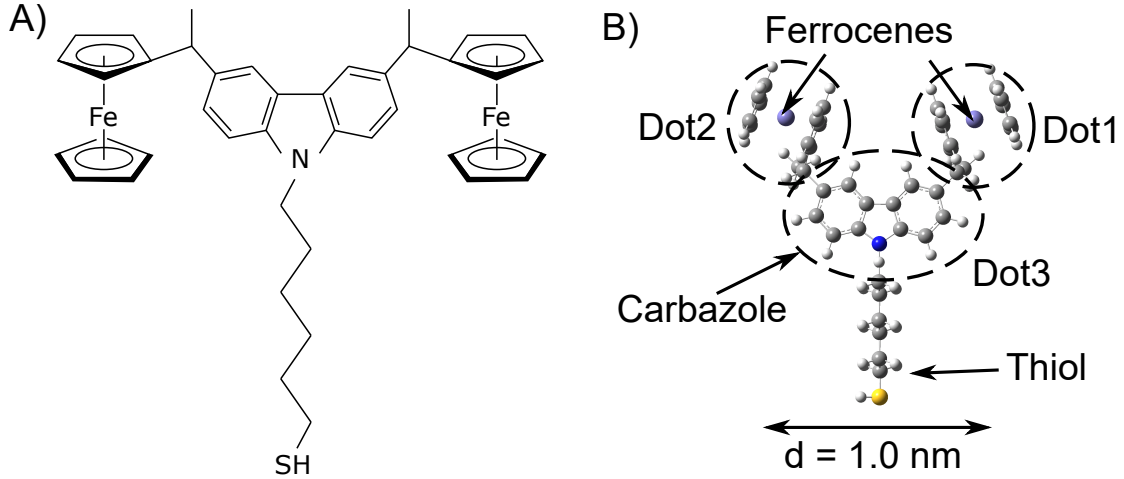


Figure 2.4: Bis-ferrocene molecule. (A) structural representation. (B) Ball and stick representation.

The Bis-ferrocene molecule, used in this work, will be presented. This molecule, Fig. 2.4, that has been synthesized ad-hoc to be used in MQCA, has two “dots”, called *Logic dots*, and another one called *Null dot*. These dots are redox sites and behave as charge containers. This is because a redox site can lose an electron (be oxidized) or gain an electron (be reduced) without breaking the chemical bonds of the molecule. This configuration results in a molecule with three spots where the charge could be located. The central dot is a third redox sites. Considering the previous introduction on QCA (2), four dots should be available. Even if molecules with four redox sites exist, two Bis-ferrocene molecules will form a QCA cell. In this way, the logic dots will form the discussed 4 dots structure, while the two null dots will be used to encode the reset, unstable, state. A possible clocking mechanism for Molecular QCA is the same showed in 2.3. A three or four phases clock could be used. Since electrons are involved in MQCA, the reset signal is an electric field. The molecules are anchored to the substrate through a fourth dot: a thiol group. This fourth dot is used during Self-Assembled-Monolayer (SAM) on a gold substrate. Figure 2.5 shows a schematic view of a MQCA wire. In order to reset the elements a vertical electric field is used. The charge will be pushed to the null dot and the molecule will be in reset state. An opposite electric field is used to force the charge out from the null dot. The other elements of the circuit

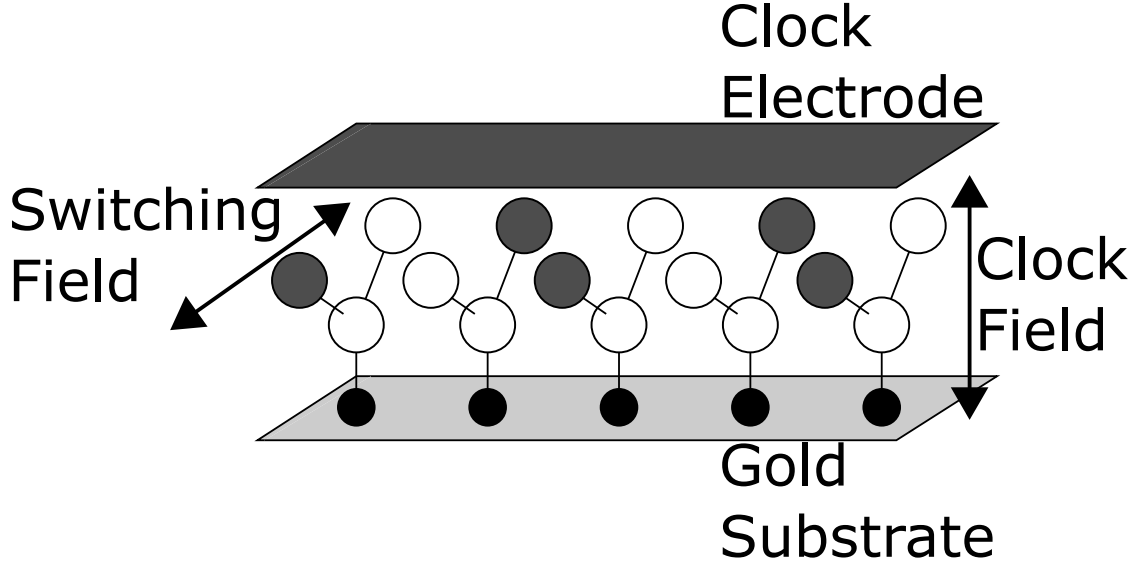


Figure 2.5: 3D view of a molecular wire. In the picture, it is possible to see the gold substrate where the molecules are anchored and the top electrode use to generate the clock field. The other molecules are responsible for the switching field.

will generate an interaction field. This field is mainly horizontal since the charges are almost equidistant from the substrate. The superposition of the vertical and horizontal field is enough to move the charge in one of the logic dots. In this way, the information will be propagated.

2.2 Magnetic QCA

Another possible implementation of QCA is based on magnetic elements. These technologies are normally referred to as Nano-Magnet Logic (NML). In NML technology, single-domain nanomagnets are used to represent binary information. The magnetization vector encodes the logic '1' and '0'. Two different versions of the NML can be identified: the in-plane NML (iNML) and the perpendicular NML (pNML). In iNML technology, the magnetization vector lies in the plane of the magnets [41][32]. On the contrary, the magnetization is perpendicular to the plane where the elements are placed in pNML.

2.2.1 iNML

In this technology, single-domain nanomagnets are used as a basic computational cell. The shape anisotropy is exploited to encode binary information. Rectangular shaped nanomagnets are usually preferred with a dimension of (50x100x20)nm or (60x90x20)nm [32]. They show only two stable configurations, which represent

the logic 0 and logic 1 (Fig.2.6.A). Thus, the magnetization vector lies parallel to the plane where the magnets are placed [25]. The magneto-dynamic interaction among neighboring devices makes it possible to propagate the digital information through the circuit. Magnets arranged in a row try to reach the minimum energy configuration, i.e. they are antiferromagnetic (AF) coupled and align themselves in an antiparallel way (Fig. 2.6.B). On the other hand, nanomagnets aligned vertically are coupled ferromagnetically (F) (Fig. 2.6.C). As described in [45], the coupling

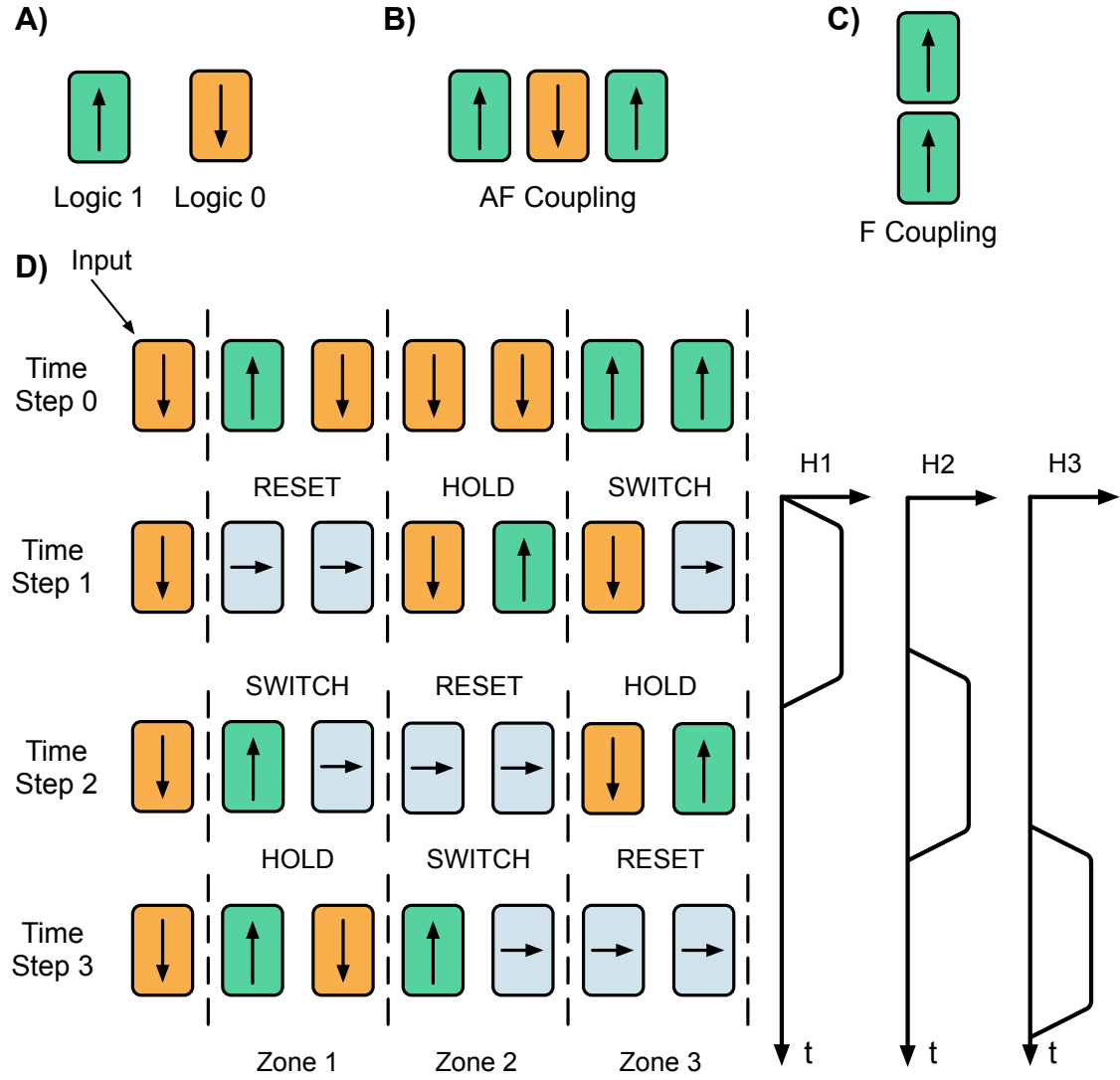


Figure 2.6: A) iNML basic cells; B) Chain of nanomagnets antiferromagnetically coupled; C) Vertically aligned nanomagnets ferromagnetically coupled; D) iNML three-phase clock system;

among neighboring cells is not enough to obtain the switching of the nanomagnets. The energy barrier introduced by the shape anisotropy is too high to be overcome by the dipole-dipole interaction. Hence, magnets need to be forced into a metastable state to propagate the information correctly. An external field generated by a current wire buried inside the substrate is usually used to force the magnets into an unstable state, which is called the reset state. This external agent is called the clock and rotate the magnetization vector along the short axis by 90° . Once the external field is released, the magnets re-align themselves according to the dipole-dipole interaction. Unfortunately, due to the thermal noise, the number of magnets that can be cascaded is limited to five or six [32]. In the literature, several clocking mechanisms have been proposed to overcome this limitation. The most common is a three-phase clock scheme, where three partially overlapped clock signals are alternatively applied as depicted in Fig. 2.6.D. The picture summarizes the topology of common iNML circuits that in this particular case is a magnetic wire. The layout is divided into slices, named clock zones, and by applying in sequence the three clock signals the digital information propagates towards the output. Time step 0 shows the initial configuration of the magnets. At time step 1, the clock signal H1 is applied to zone 1, forcing the magnets belonging to that zone into the metastable state (RESET). At time step 2, the clock signal is released from zone 1, consequently, the magnets from that zone go into the SWITCH state, while zone 2 is forced into the reset state. Similarly, at time step 3, zone 1 retains its magnetization (HOLD state), zone 2 switches, while zone 3 moves into the reset state.

Logic operations are achieved by iNML gates that are obtained by properly arranging the nanomagnets in the layout or changing their geometry. The basic gates are majority voters and inverters: the former is achieved by surrounding a central element with three magnets (Fig. 4.5.A). The latter is obtained by chaining an odd number of magnets in the clock zone. Finally *AND* and *OR* gates are obtained with magnets with a slanted edge [33], in this way a preferred direction is set, and the logic function is defined.

2.2.2 pNML

In pNML technology, single domain nanomagnets with perpendicular magnetic anisotropy (PMA) are used to represent the binary information. The magnetization vector encodes the logic '1' and '0' [12] [10] as depicted in Fig. 2.7.A. In pNML technology, information propagation is accomplished differently from CMOS; here, each magnet is a non-volatile memory element. The PMA is obtained with a multi-layer stack of Co/Pt. The number of layers and their thickness define the magnetic properties of the device [8]. By placing magnets one near the other, it is possible to obtain a wire (Fig. 2.7.(C)). Two neighboring magnets try to reach the minimum energy configuration; thus, opposite magnetization direction. However, the magnetic

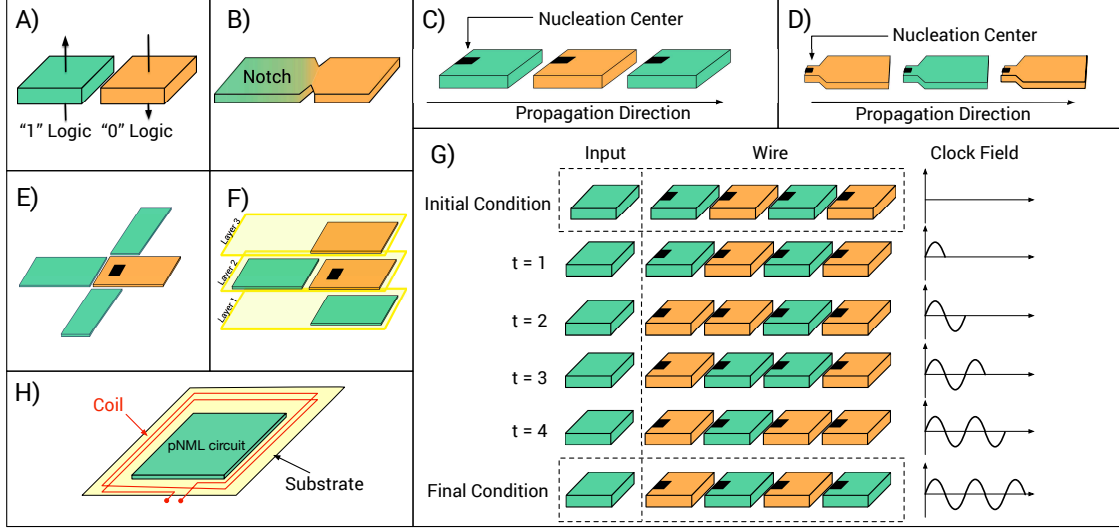


Figure 2.7: NML: (A) magnetic binary representation; (B) Notch; (C)(D) pNML wire; (E)(F) pNML majority voter; (G) pNML clock mechanism. (H) Clock coil representation.

interaction among neighboring magnets is not enough to enable the magnetization reversal. Two additional elements are introduced to guarantee a correct information propagation: the clock mechanism [41] and the artificial nucleation center (ANC) [39]. The former is needed to help the magnetization reversal when the superposition of the clock field and the coupling field, coming from neighboring cells, are present. The latter, instead, defines the information propagation direction.

An example of the clock mechanism is depicted in Fig. 2.7.(G). In pNML technology, the clock field is a sinusoidal magnetic field, perpendicular to the magnets plane, applied to the whole circuit. In order to generate this magnetic field, coils are placed beneath the magnets. Figure 2.7.(H) shows how the coil can be 3D integrated in pNML technology. The clocking apparatus corresponds to the “power supply” in CMOS circuits [32]. To understand its behavior, consider that a new input is placed near a pNML wire in stable conditions (Fig. 2.7.(G)). The input magnet and the first element of the wire have the same magnetization direction. The anti-ferromagnetic interaction tries to force an opposite value on the first magnet, but the coupling field is not enough. When the clock field has the same direction as the “desired” value, the superposition of the two forces leads to the magnetization reversal. This situation is verified at $t=1$ in the example. At time $t=2$ the first two elements of the wire have the same magnetization, and the clock field has inverted its direction. The same conditions as before are verified and also the second magnet can switch to the new value. The process is repeated for the successive time instants until the input is transferred to the output. The minimum energy configuration is preserved even if the clock field continues oscillating.

The problem of information direction is not solved by the clock field. The process described before works only if it is possible to ensure that the magnet on the right will be the switching one. The ANC is used to force this behavior. By changing the magnetic properties on a specific spot of each magnet, it is possible to define a region more sensitive to magnetic field changes. The ANC is the area of the magnet where the nucleation of the domain wall takes place. In this area, the magnetic field needed to force a new value is less with respect to the rest of the magnet. Considering Fig. 2.7.(C), the darker part of each element identifies the ANCs. In this example, the left magnet is able to influence the one to its right, while the contrary is not possible. Thus, the only allowed propagation direction is from left to right. Normally the ANCs are obtained through a Ga^+ Focused Ion Beam (FIB) irradiation. The FIB changes the magnetic properties of the irradiated part, lowering the magnetic anisotropy. A different way for creating the ANC has been presented in [28]: a modification in the magnet geometry and the fabrication process leads to the same result as FIB irradiation. An example can be seen in Fig. 2.7.(D). In pNML technology, the main logic gate is the Majority Voter (MV), Fig. 2.7.(E). By placing three different magnets near an ANC it is possible to define an MV. The output of this logic gate is the opposite of the majority of the inputs, due to the anti-ferromagnetic interaction. Note that an MV with a fixed ‘0’ input behaves as an AND gate, while a ‘1’ defines an OR gate. Furthermore, 3D circuits are intrinsically enabled by the technology [5]. Fig. 2.7.(F) shows a compact version of the MV. In that case, two inputs lay on different planes with respect to the gate. Considering that MV can work with an odd number of inputs [7], it is easy to create very compact layouts.

pNML technology provides also a synchronization element. Fig. 2.7.(B) shows a notch [22]. This element, thanks to its particular shape, pins the incoming information [38] [37]. An additional magnetic field, parallel to the magnet plane, is needed to lower the energy barrier restoring the information propagation in the notch. This in-plane field is called the de-pinning clock. The de-pinning clock is generated by a wire passing under the notch.

Part I

ToPoliNano Framework

Even at the early stage of a technology, it is important to explore its possibility and limitation. While the technologists focus on the device, also a circuit exploration is important. These kinds of studies could give important feedback to the scientific community. Increasing the number of elements inside a circuit leads to the need for CAD and EDA tools to perform design and simulation. No commercial tools exist capable of handling such technologies. Also at the research level the available tools are limited. QCADesigner [47] is widely used by the scientific community. However, it is based on the general QCA principle. The cell available don't have a physical counterpart, resulting in an approximate analysis of the technological implementation. QCADesigner is not capable of automatic place and route QCA cells. Other tools are available in literature for this purpose, like fiction [46] and Ropper [19]. The former is an open-source project that enables minimum layout dimensions fixing the constraints. It also provide heuristic approach. During the development of this work, the collaboration with the developers of fiction lead to an integration of the two tools. The latter is very similar to fiction and enable the automatic place and route of verilog netlists. None of the other tool in literature is based on physical implementation of QCA technologies. For these reasons the ToPoliNano was designed. The ToPoliNano framework (Torino Politecnico Nanotechnologies), is a complete framework for the design and simulation of digital circuits based on QCA technologies. The idea is to have a flow similar to the one available for CMOS technology. The ToPoliNano framework is composed by two different tools: ToPoliNano [36] and MagCAD [35]. It supports different implementation of FCN technologies and it was developed with a modular structure. In this way it is possible to add new technology and increase the framework possibilities. The tools are developed in C++ and can be downloaded for free from the ToPoliNano website (<https://topolinano.polito.it>). Furthermore, they are cross-platform and available for Windows, MacOS and CentOS. Figure 2.8 shows the framework design flow.

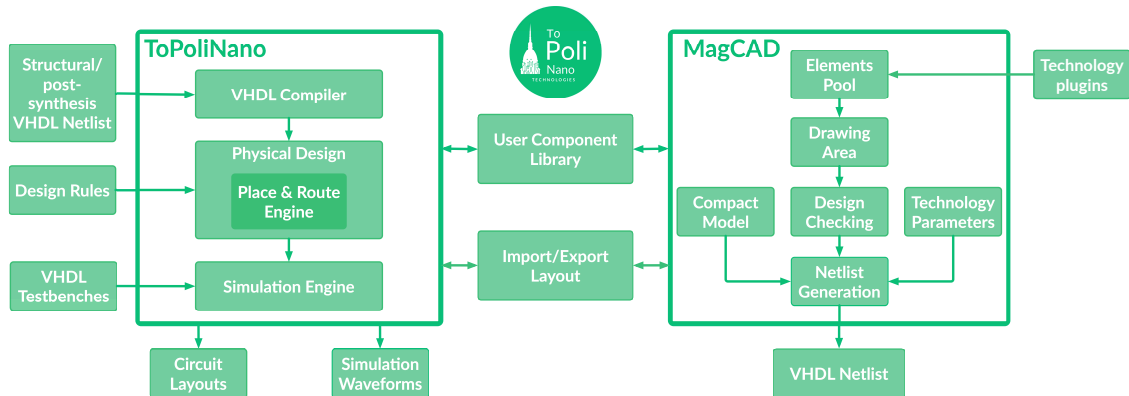


Figure 2.8: Schematic representation of the ToPoliNano framework.

ToPoliNano is an EDA tool that embeds a place&route and simulation engine. It can perform physical layout automatically, starting from a structural HDL description. Furthermore, it can compile a VHDL testbench and perform the simulation of the generated layouts. Layouts performed by ToPoliNano can be exported and later re-opened or can be manually modified using the other tool, MagCAD.

MagCAD is a custom layout editor. Here, the user has a fast prototyping flow: design a circuit, and then extract a VHDL netlist to simulate it. The automatically generated netlist embeds a compact model of the target technology. The extracted layouts can also be opened in ToPoliNano or used as cells within the user's component library during the physical design. An important aspect of our framework is the support for different technologies. The tools are based on technology plugins: specifying the technology building blocks, the associated compact models for each element is enough to define a new technology that could be loaded and used during the design phase. With the tools, new files type have been defined: QCA Layouts (qll) and QCA Components (qcc). Those extensions define particular type of *XML* files. Both are based on "tags" as standard *XML* but the main difference is in the fact that components do not have information on the technological settings. On the contrary layout files need to have all the information related to the elements and the technology. In the following section, detailed descriptions of the tools are presented, with a specific focus on the features developed during my Ph.D.

Chapter 3

ToPoliNano

ToPoliNano is an EDA tool that enables a top-down design flow similar to the one available for CMOS. Starting from post-synthesis HDL description it is possible to perform Place&Route and simulation. A detailed flow of the tool is available in Fig. 3.1. In this way, it is possible to design complex architecture using the

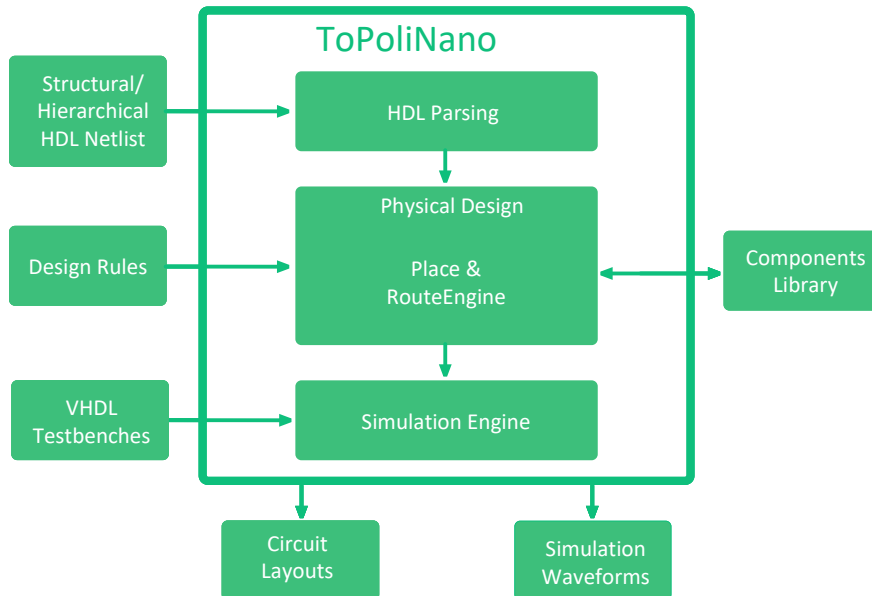


Figure 3.1: Schematic representation of the ToPoliNano tool flow.

selected technology. Different architectures and circuit topology can be described in HDL, and then the tool is used to perform the Place&Route. Furthermore, it is possible to tune the technological parameter and also select among different optimization algorithm. ToPoliNano can handle circuit with increasing complexity and number of gates up to tens of thousands. Behavioral descriptions can be synthesized (using Synopsys design compiler or ABC or any desired tool) in order

to provide a structural netlist. ToPoliNano supports also majority gate synthesis, where logic gates are achieved through MVs. Currently, it supports a three-phase clock mechanism. In the following sections, the main part of the tool are analyzed.

3.1 HDL Parsing

The tool supports structural, hierarchical and combinational circuit descriptions. Loops are not handled by the tool, indeed it is able to detect loops in the netlist and generate an error. Input files could be both in VHDL and Verilog. The VHDL parser was already present in the tool, while the Verilog one has been developed as part of this work. The Verilog parser is based on an open-source project developed at the EPFL, called “Lorina” [16]. The adopted parser is structured in order to call a specific function, called “callbacks”, for every tag in the Verilog file (Module Declaration, Inputs, Outputs, And Gate, etc.). This approach is very flexible, the “callbacks” are part of the API of the parser. Indeed, the parser is divided into two different classes: the actual parser, called *VerilogParser* and the *VerilogReader*. The former is used to tokenize, parse and interpret the Verilog source file. The latter, instead, is an interface, providing all empty implementation of the “callbacks”. This class can be subclassed in a custom class where the “callbacks” are used to build the desired data structure. Thus, the “callbacks” have been reimplemented in order to translate the parsed Verilog in the HDL Graph used inside ToPoliNano. Figure 3.2 shows an example of the HDL Graph resulting after parsing a 4-bits ripple carry adder (RCA). The internal data structure of ToPoliNano for HDL netlist is a tree: the root is the top-level entity/module, the other components/modules are the trunks, while logic gates are in the leaves. The nodes of the tree, called *HDLNodes*, have the name of the corresponding element in the HDL description. Furthermore, each node holds the pointer to its father and a list of connections. Each connection is stored as a pair of string, where each string is one of the two net to be connected. Nodes can be of two types: *Composite Node* or *Leaf Node*. The former is used for all the components/modules of the hierarchy. It has also a vector of pointers where the children nodes are saved. The name used for *Composite Node* is the one of the instance, while member *Component Type* is used to save the name of the component. As an example, in the RCA4 of Fig. 3.2 the two RCA2 nodes will have different names but the same type. The latter is used for the logic gates of the circuit and has no children nodes. Also in this case, *Component Type* is used to define the element associated with the node. *Leaf Nodes* can be AND, OR, Inverter or majority voter. With this structure, it is possible to visit all the elements of the graph knowing only the head. This is used during the layout phase. The “callbacks” of the different logic gates were defined to allocate a new *Leaf Node* and define the interconnections. In the case of a module instance, a new parser object is defined recursively. If the module description is available,

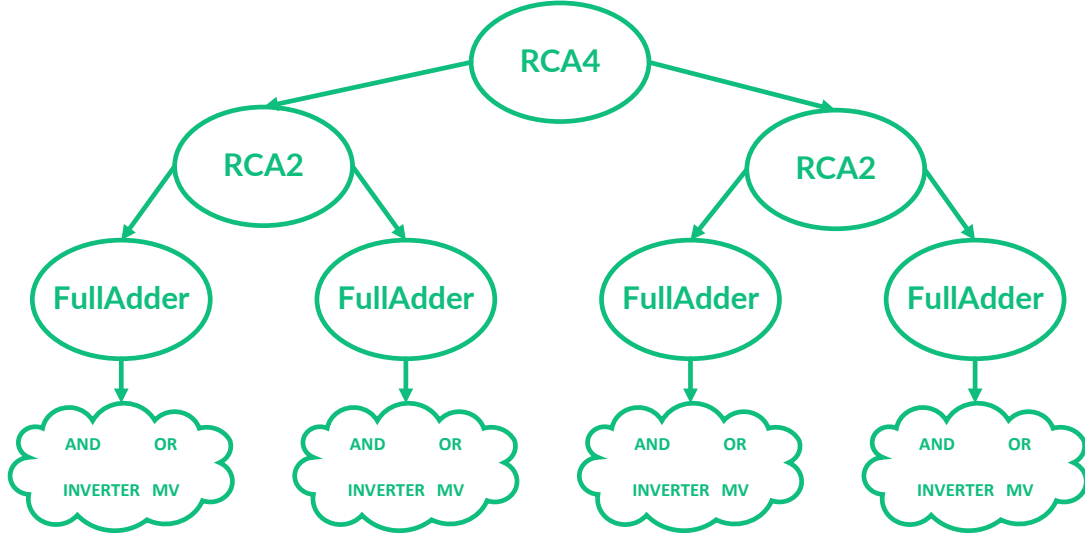


Figure 3.2: Graph representation in memory of a hierarchical 4-bits ripple carry adder. The top-level is the root of the tree. 2-bits RCA instantiated in the circuit are the trunks of the tree with the four instances of full adders. Finally, the leaves are composed by the logic gates: AND, OR, Inverter and MV.

it is parsed and an HDL Graph associated with the module is defined. The head of the new graph will be used as a node of the original graph and also added to the component list. In this way, multiple instances of the same components will be parsed only once. Unfortunately, hierarchical Verilog parsing was a new feature of the “Lorina” parser. During the development of the “callbacks”, several issues were fixed also on the parser side. For example, wire buses assigned in module instantiations resulted in parser errors. To solve those problems the source file of the “Lorina” parser was modified and the support for hierarchical designs was added. The final version of the parser is capable of handling Synopsys post-synthesis Verilog descriptions, ABC output files and Verilog produced with the “Lorina” parser itself.

3.2 Layout Phase

After parsing the source files the internal data structure is ready for the layout phase. During this phase, specific algorithms are used to perform the placement and routing of the elements. The layout is composed of different steps. Firstly the HDL Graph is translated into another data structure. Since ToPoliNano has been developed to handle different technologies, the HDL data structure needs to

be translated according to the selected technology. Currently, only iNML is supported for the layout process. An adjacency list is used during the iNML layout. The translated data structure is then processed by specifically designed algorithms, that take into account the technology constraints. As an example, the maximum number of elements or the limit of cascaded logic gates in the same clock zone are some of the constraints. Figure 3.3 shows the internal graph of a 2-to-1 multi-

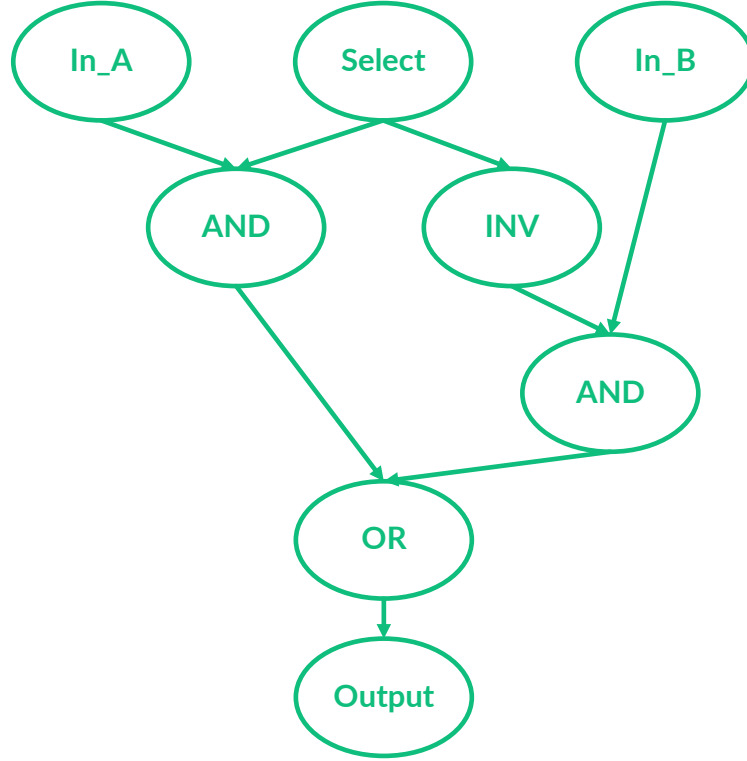


Figure 3.3: Internal representation of the 2-to-1 multiplexer. Each circle in the picture is a node of the ToPoliNano data structure. Edges show inputs and outputs of each node.

plexer, after the HDL translation. Fan-outs are added as the first step: a *coupler* is inserted when a signal drives more than 1 gate or output. Later, the circuit goes through a function that balances the paths among the logic gates. iNML circuits are like big pipelines, and the inputs of logic gates must be synchronized. Figure 3.4 shows the mux graph after fan-out management and path balance. The *coupler* inserted for input “Select” was balanced with two wires node for the other inputs. In this way, both the input of the logic gates have to travel the same number of clock zones. At this point the main issue of iNML technology is addressed: cross wires reduction. Since iNML is a planar technology, a special element is used to implement crossing in the signal paths. The introduction of a cross wire needs to

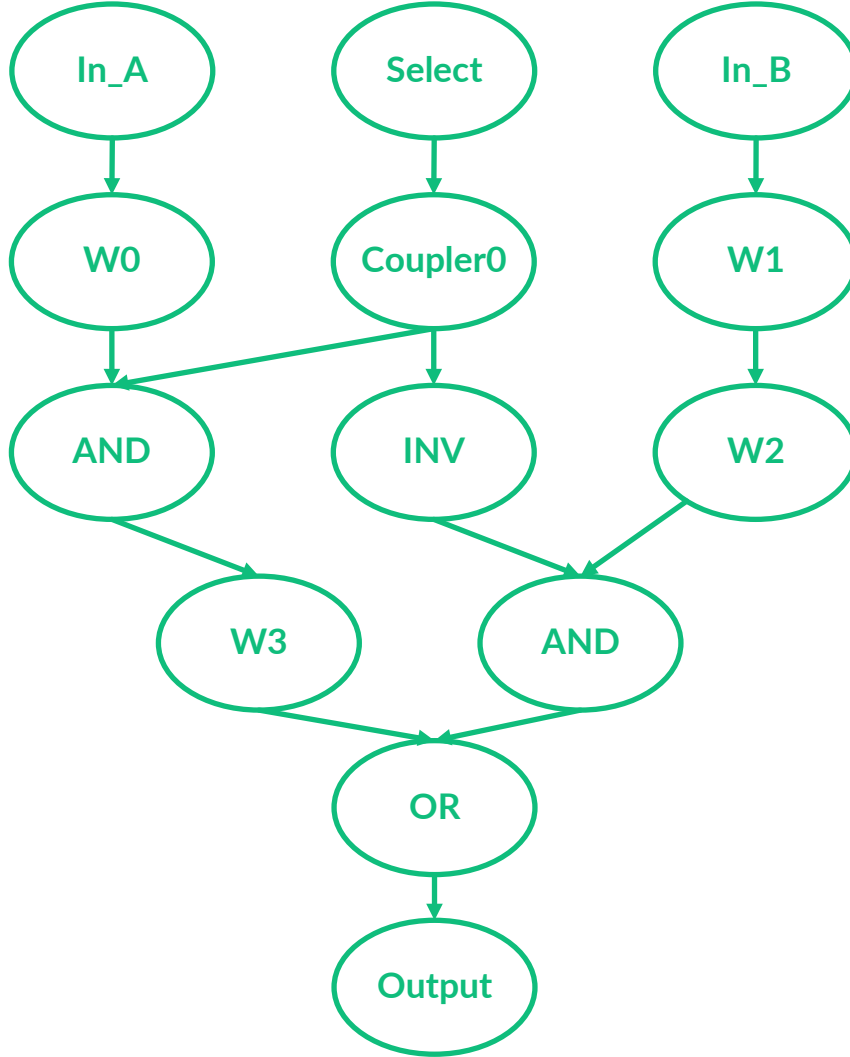


Figure 3.4: The multiplexer structure after path balance and coupler management. The number of nodes is increased, together with the number of levels. This structure will be used to perform the actual *Place&Route*

add a clock zone to the circuit: wires have to be added to balance again the paths. Therefore, the number of crosswires is critical and a specifically adapted algorithm is used to tackle this issue [36]. After this step, the graph is fixed and the physical placement is started. Each element is translated into the corresponding physical implementation. Each level of the graph will be a clock zone of the physical layout. After placing the gates and every element resulting from the previous management of the graph the connections among the clock zones are routed. The final layout is then optimized and useless columns, composed by only straight wires, are removed

in order to improve the circuit performance. Figure 8.13 shows the final layout of the discussed mux.

ToPoliNano supports different layout approaches: Fully Hierarchical, Partially

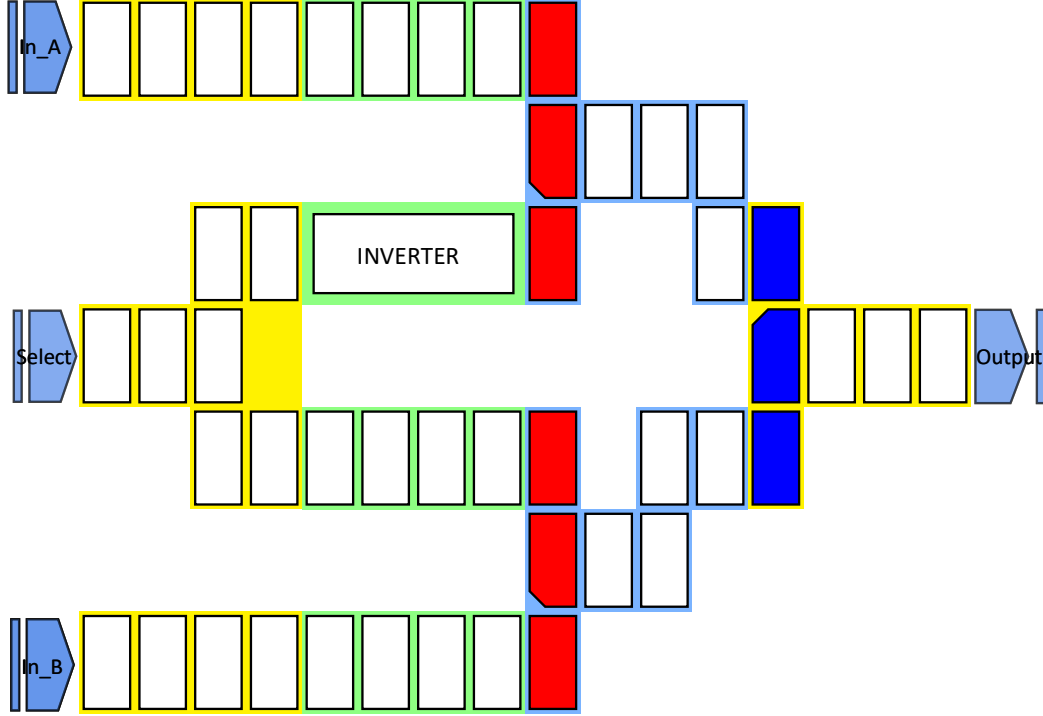


Figure 3.5: Final layout of the 2-to-1 multiplexer: input and output pins are the light blue elements with the name superimposed. Magnets of different colors belong to different clock zones. Logic gates are differently colored and it is possible to see the slanted edge of the central magnet.

Hierarchical and Flat. Depending on the selected approach only some nodes of the HDL Graph are used to build the adjacency list. For example, with the Flat approach, only the leaves of the HDL graph are used. Thus, only the basic logic gates will be translated and used to perform the layout. Differently, with hierarchical approaches, the nodes identified as components will be used, and all the children of those nodes will be ignored. Considering again Fig. 3.2, if a fully hierarchical approach is selected, the 2-bits RCA will be marked as components. As a result, in the final layout, only those elements and the magnet used to interconnect them will be present. A hierarchical layout could be used also to reduce computational time in case of big circuits. Indeed, it is possible to load components previously defined during the layout phase. If components are not present in the library a new one is generated and saved on the disk. To handle components the layout

phase is divided into two parts: component management and the final layout. The second part consists of the same steps performed during the flat layout. Instead, during component management, all the components node are considered. If the user selected to load the component from the library, the tool will try to load it from the disk. If the file is not present a new component is generated. Otherwise, component information is extracted from the file, and the component is added to the layout. This process is recursive: if a component includes a smaller block, the operations are repeated until the leaves of the graph are reached. For example, in the 4-bits RCA, the 2-bits RCA is a component. When the component needs to be generated, a new layout process is launched, and considering the 2-bits RCA as the top-level circuit. Recursively, the FA is generated, and then each block is used to build the higher circuit in the hierarchy. This description is valid for the fully hierarchical approach. In case of partially hierarchical selection, the 2-bits RCA are realized flat.

Those differences in the hierarchical approaches defined the need for a particular management of the component on the disk. Thus, two different folder are present in the ToPoliNano workspace for components: *Components Created* and *Components Hierarchical*. In this way it is possible to separate the matryoshka like circuits from the flat ones. During the layout phase ToPoliNano is looking in the folder corresponding to the selected method. The final layout could be inspected through the ToPoliNano GUI. Furthermore it is possible export an image or save it in the ToPoliNano file format.

3.2.1 fiction Integration

ToPoliNano place&route engine is based on heuristic algorithms. An exact approach was not available in the framework. However, *fiction* [46] have been developed at Bremen University. *fiction* is a command-line tool capable of performing an exact placement of a circuit given a set of cells and technology parameters. Unfortunately, this kind of approach is limited to small circuits, less than twenty gates. Anyway, the resulting circuits are the best in terms of area occupation, and therefore performance, obtainable with the given constraints. Given the hierarchical design capability of ToPoliNano the exact approach could have been used for the building blocks, using the heuristic algorithm during the top-level interconnections routing. This idea was used during a joint research project with the Bremen University. The scope of the project was to implement an automatic procedure integrating *fiction* and ToPoliNano. Firstly, the ToPoliNano clock scheme and technological parameters have been inserted inside *fiction*. Furthermore, the possibility of writing a “.qcc” file was introduced inside the *fiction* code. Simultaneously, ToPoliNano was modified in order to add the *fiction* call during the layout phase. A checkbox and a spin-box were added to the layout parameter selection window, Fig. 3.6.

Figure 3.6: ToPoliNano windows showing the parameters related to fiction calls. The user can check the fiction box and specify the maximum time available to complete the layout phase before going back to the ToPoliNano engine.

In this way the user can select to use *fiction* and specify a timeout. *fiction* executions could last also days if a high number of logic gates is present in the circuit. The user can set a maximum iteration time, if the exact placement is not available after that time, the heuristic process is called. Inside the ToPoliNano code, the “QProcess” class was adopted to implement the *fiction* calls. In particular, the string needed to perform the layout, including the technological parameters, is composed by combining all the elements. Then a “QProcess” is called with the given string. This class executes a command in a temporary shell, and collect the output of the execution. When the *fiction* call ends, either with positive output or terminated by the time out, ToPoliNano check if the component description was generated. A new folder, *Component fiction* was defined inside the ToPoliNano folder structure. The components generated by *fiction* are placed there. Furthermore, during the layout phase this folder is checked before the other ones: if a component is present, it is loaded In this way the exact components are preferred

to the heuristic ones. The Resulting block diagram after the integration is shown

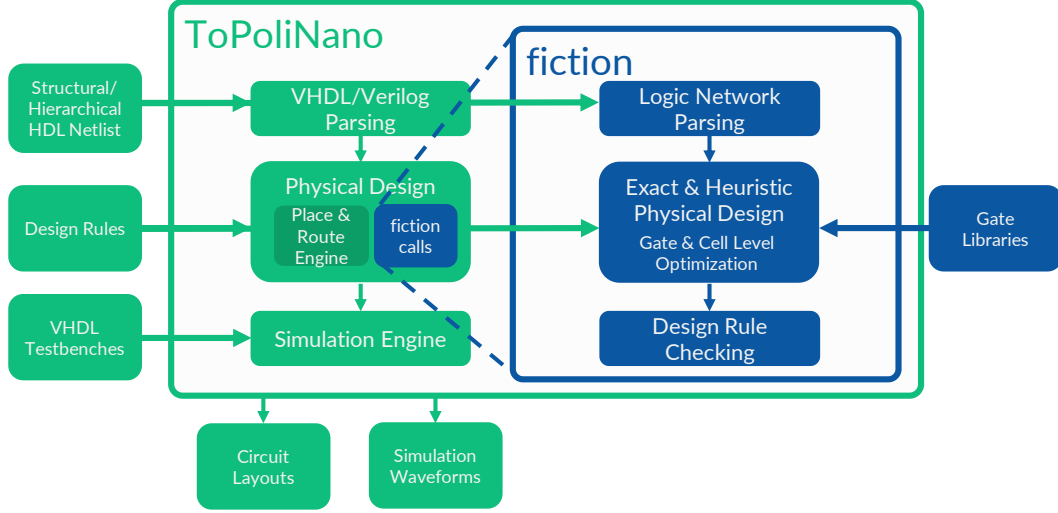


Figure 3.7: Schematic view of the toll structure. *fiction* is integrated inside ToPoliNano and it is called during the layout phase. The exact algorithm is used to manage components in the hierarchy.

in Fig. 3.7.

3.3 Simulation

After performing the layout it is possible to simulate the resulting circuits. It is also possible to load a previously saved circuit and perform directly the simulation. A VHDL testbench is needed to perform a simulation: ToPoliNano can parse the testbench and apply the correct stimuli to the circuit. The parsed VHDL testbench is used to create an internal data structure composed of *Levels*. Each *Level* has a boolean value and two time information, start and stop time. This structure is used during simulation to retrieve the values to be applied to the input of the circuit. Furthermore, the clock signals are defined according to the selection made by the user through the GUI, as shown in Fig. 3.8. In the same picture, it is possible to notice that two algorithms are available for simulation: Behavioral and Single Domain Equation. As expected, the former does not take into account the physical properties of the elements. It is based on the ideal ferromagnetic and anti-ferromagnetic coupling among the elements, that are considered boolean variables. If the clock is active for an element, it will be in the reset state. On the contrary, if the clock is not active the surrounding elements are used to evaluate the new state. Since the clock mechanism is fixed, it is possible to visit the circuit knowing

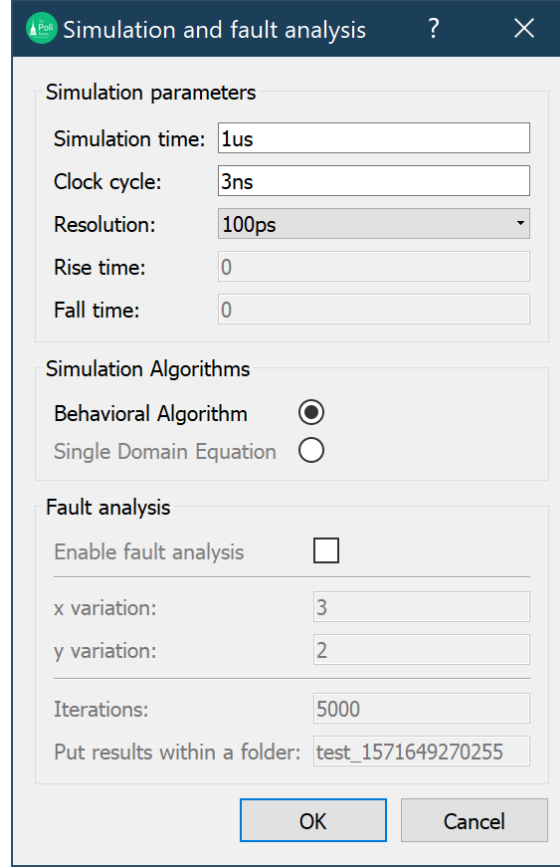


Figure 3.8: topo windows showing the parameters related to the simulation. The user can select the clock waveform specification and the simulation engine. Furthermore, the simulation resolution can be modified.

the direction of propagation. This kind of visit, described in [36], distinguishes the case where the number of neighbors is greater than one (MVs and *Couplers*) from the others (Wires, Inverters, etc.).

Pseudocode 1 is used in the former situation. In the case of *Couplers* some of the elements will be in the reset state, therefore only one of the surrounding element will be used to determine the value of the magnet. The other possible configuration is solved using pseudocode 2.

The other simulation engine is based on the LLG equation [llg]. As in the previous case, the visit of the circuit is not general and assumes a specific clocking mechanism. Indeed, changing the visiting order would lead to completely different results. Furthermore, it is designed only for iNML technology. For these reasons, a new simulation engine has been developed. The new simulator is the main contribution described in this work and it will be presented in part II.

Algorithm 1 Behavioral simulation algorithm for more than one surrounding magnets

Precondition: *magnet* is the magnet to be evaluated, *neighbors* is the list of the elements surrounding the evaluated magnet.

```
1: ones  $\leftarrow$  0
2: zeros  $\leftarrow$  0
3: for each element  $\in$  neighbours do
4:   if element.x() = magnet.x() then
5:     if element.value = '1' then
6:       zeros  $\leftarrow$  zeros + 1
7:     else
8:       ones  $\leftarrow$  ones + 1
9:     end if
10:  else
11:    if element.value = '1' then
12:      ones  $\leftarrow$  ones + 1
13:    else
14:      zeros  $\leftarrow$  zeros + 1
15:    end if
16:  end if
17: end for
18: if zeros > ones then
19:   magnet  $\leftarrow$  '0'
20: else
21:   magnet  $\leftarrow$  '1'
22: end if
```

Algorithm 2 Behavioral simulation algorithm for single surrounding magnet

Precondition: *magnet* is the magnet to be evaluated, *neighbour* is the coupled magnet.

```
1: if neighbour.x() = magnet.x() then
2:   if neighbour.value = '1' then
3:     magnet  $\leftarrow$  '0'
4:   else
5:     magnet  $\leftarrow$  '1'
6:   end if
7: else
8:   if neighbour.value = '1' then
9:     magnet  $\leftarrow$  '1'
10:  else
11:    magnet  $\leftarrow$  '0'
12:  end if
13: end if
```

Chapter 4

MagCAD

The other tool part of the ToPoliNano framework is MagCAD. This tool is a custom layout editor: the user can design custom circuits starting from the technology building elements and previously designed components. The tool has a simple and user-friendly GUI. At the start-up a new layout is created: the user can select the target technology among the ones supported by the tool. The technologies are plug-in that the tool can load. A new technology could be added simply by defining the correspondent plugin. A technology is identified by its building blocks and a set of general properties. Fig. 4.1 shows the technology selection windows and in

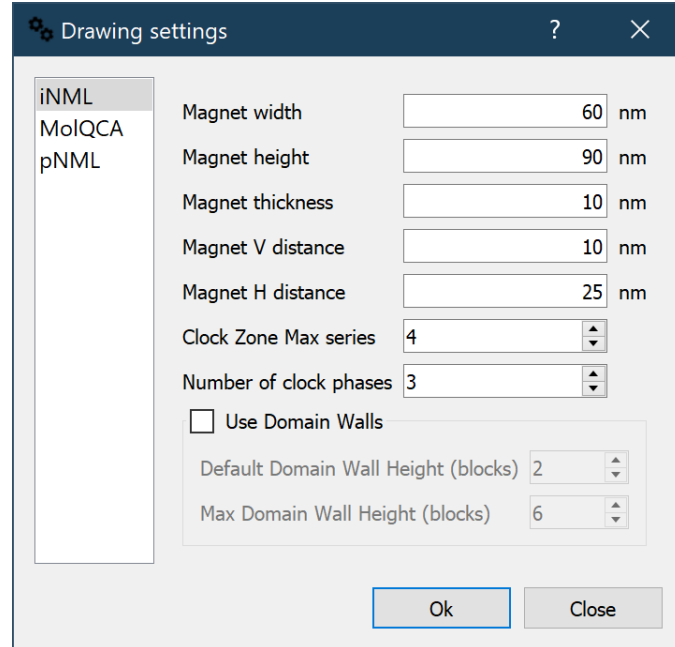


Figure 4.1: Technology selection window. On the right the settings for iNML technology modifiable by the user.

particular the technology settings for iNML technology. In the same figure it is possible to notice that three technology are supported in MagCAD: iNML, pNML, and MolQCA. After technology selection, the main GUI is shown to the user, Fig. 4.2. It is composed of four main parts. The central portion is the drawing area. On the

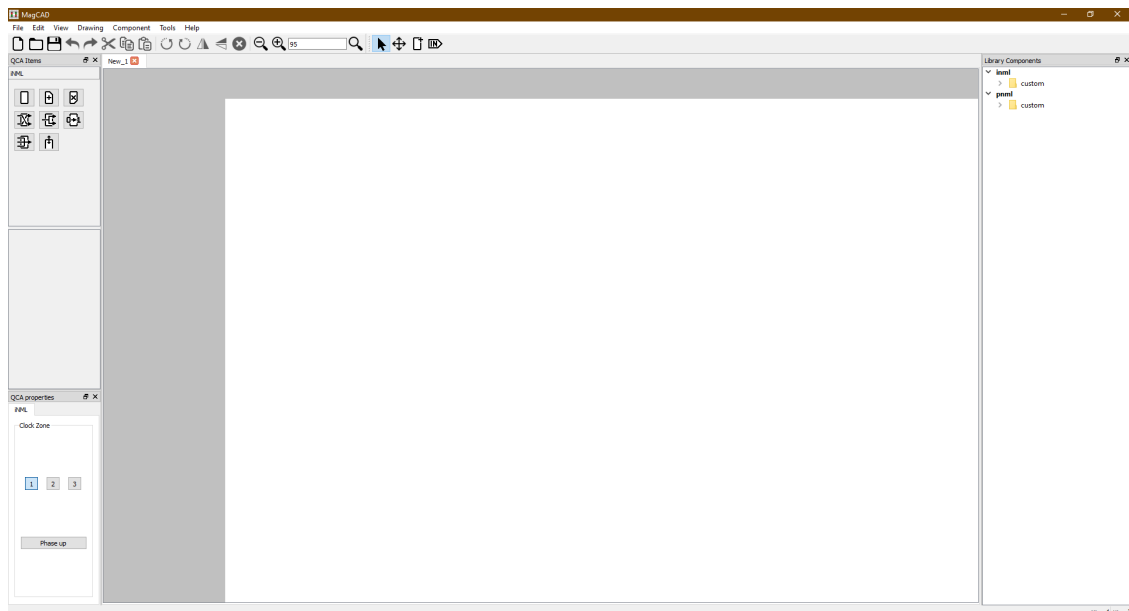


Figure 4.2: Mainwindow of MagCAD

left, there are the technological elements and the widgets used to set the technological properties of the elements. On the right, a tree-view is used to browse all the previously designed circuit. Finally, in the top part, all the menus are present. The design phase consists of placing the element from the left part in the drawing area. Furthermore, the tool has different features that the designer could use during the layout: cut&paste, collision detection, and zoom are some of them. It is possible to insert also pin elements: these objects are used to define the connection when the circuit is exported as a layout “.qll” or as a component “.qcc”. Components will load default technology settings when opened for inspection. Furthermore, it is possible to insert in the layout also circuit from the component library: in this case, the settings of the layout will be used to define the elements. Indeed, two possible type of insertion are available: *Insert as component* or *Flat insert*. With the former, the circuit is inserted as a black-box and the pins are shown to properly connect the signals. In order to verify the designed circuit, it is possible to extract a VHDL description of the layout. This description is based on technological libraries that embed models of the target technologies [40]. The extracted netlist can be simulated using standard HDL simulators, like Modelsim from Mentor Graphics [31]. Another possibility is to open the circuit with ToPoliNano and perform the simulation with the internal simulation engine. Different functions of MagCAD have

been developed as part of this work. In the following, three main contributions will be described: the FunCoDe (Function and CONnection DETection) algorithm, the molecular Plugin and the physical parameter management.

4.1 FUNction and CONnection DETection Algorithm

MagCAD enables the design of custom circuits. After the design phase it is possible to extract a VHDL netlist of the circuits. In the netlist each cell is associated to the corresponding behavioral model. However, the interconnections among the cells and also their functionality depend on the circuit layout. The whole algorithm has been designed in a general way so that it may be easily extended to new devices. The algorithm determines the function implemented by groups of building blocks. Indeed, according to the surrounding elements and the signal direction, specific logic functions can be implemented. First, the connectivity of the custom layout is verified according to technology-dependent rules. Fig. 4.5 reports the connections allowed for every building block of both iNML and pNML technology. For example, in Fig. 4.5.A the central magnet of the *Coupler* and the *Majority Voter* implement a different function according to the neighboring elements. Moreover, many build-

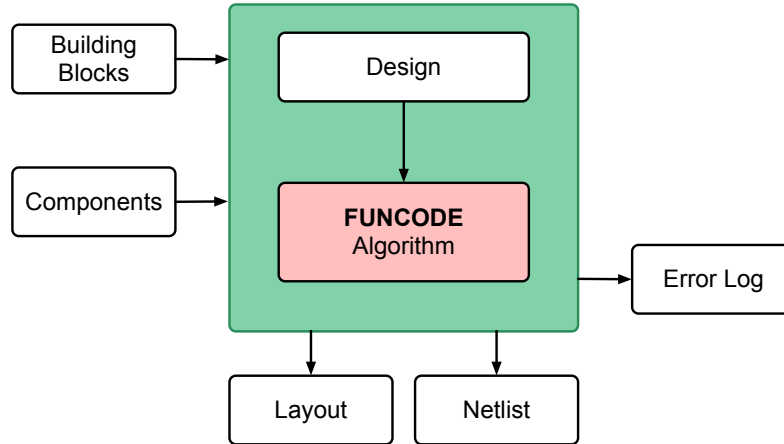


Figure 4.3: Design methodology in which the FUNCODE algorithm was introduced

ing blocks do not have a predefined connection. Thus, the final connectivity can be determined with an overview of the whole layout by looking at how the digital information moves from input to output.

As an example, in the iNML wire described in Fig. 2.6.D, the clock zone sequence can be helpful for determining the signal propagation. A similar approach can be

applied to pNML circuits. The signal flows from ANC to the opposite side of the magnetic nanowire.

After defining the connectivity of the building blocks, the basic gates are identified before writing the final VHDL netlist. Fig. 4.3 summarizes the design flow in which the proposed solution has been adopted. It is possible to use previous designed circuits, stored in a user library, as building blocks. In this case too, the connections are defined by the algorithm and a hierarchical netlist is provided to the user. The generated netlist could be simulated using a standard VHDL simulator, like ModelSim [31]. It is therefore possible to use commercial simulator to verify FCN circuits. A model of the technological elements is needed. However, the definition of a behavioral model at the early stage of a technology can speedup the research phase. In the following, a detailed description of the main steps is presented.

The proposed algorithm, FUNCODE (FUNction & CONnection DETection), can be used for circuits based on iNML and pNML technology presented in chapter 2. The general flow of the algorithm is summarized in Fig. 4.4. It starts from the inputs of the design, correctly connects each element of the circuit, determines the function implemented by an element based on the neighboring cells and then writes the final VHDL netlist.

Firstly, all the elements of the layout are translated into their corresponding *HDL-Element*. This data structure and the list of circuit inputs are taken as input by the algorithm. The *HDL-Element* stores the information about the available connections. The connectable cells are set according to the source element. As an example, the *Simple Magnet* in iNML technology has a total of eight possible connections. Each side of the magnet can be both input or output (Fig. 4.5.A). On the contrary, the pNML Magnet has four connections available, which means that only two sides can be connected (Fig. 4.5.B). However, in both cases, it is not possible to determine in advance which is the input and therefore the output. Fig. 4.5 shows examples of connections for the elements of both iNML and pNML technology. From this figure it is possible to observe that some elements have pre-defined connectivity.

Each time an element in the layout is translated, an *HDL-Element* is allocated in the memory. As the circuit gets bigger, the matrices become incredibly large and sparse. Therefore, the position of the *HDL-Element* is inserted into two special table structures, which henceforth are called sparse matrices, (Fig. 4.6.C). In the case of multi-layer (3D) layouts, a vector of these sparse matrices is used, where the layer number is used as the index in the vector. The sparse matrices are implemented as vectors and a binary search is used to insert and read the data. This approach has been preferred in order to limit the required memory to just the actual number of elements in the layout. Fig. 4.6.B and Fig. 4.6.C show the *HDL-Elements* translation, comparing the matrix and sparse matrix representations respectively. Even if the circuit reported in Fig. 4.6.A is quite simple, it is possible to observe

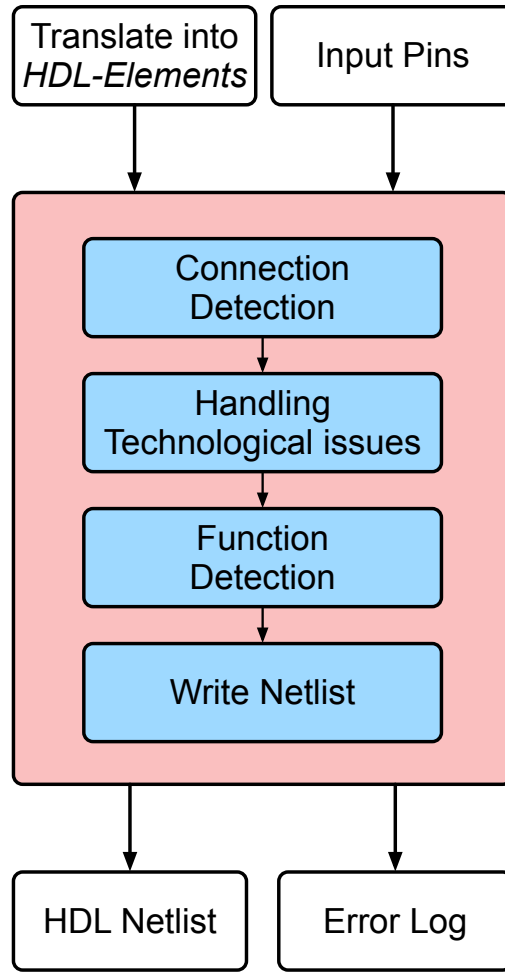


Figure 4.4: FUNCODE flow chart

that for large circuits a lot of memory can be saved. Usually, the number of building blocks enclosed within the bounding box of the circuit is much smaller than the total number of cells in the grid. Furthermore, some elements are not in both the input/output data structures. As an example, input pins are only available in the output sparse matrix, while output ones only in the input (Fig. 4.6.B). This distinction is necessary since, from the netlist point of view, inputs can provide an output connection, while outputs can receive an input connection from a neighboring cell. The vector representations are used for fast look-up during the element connections: the structure can be accessed with the coordinates of the element in the layout and the output is available in logarithmic time. When this setup phase is completed, the main algorithm can start.

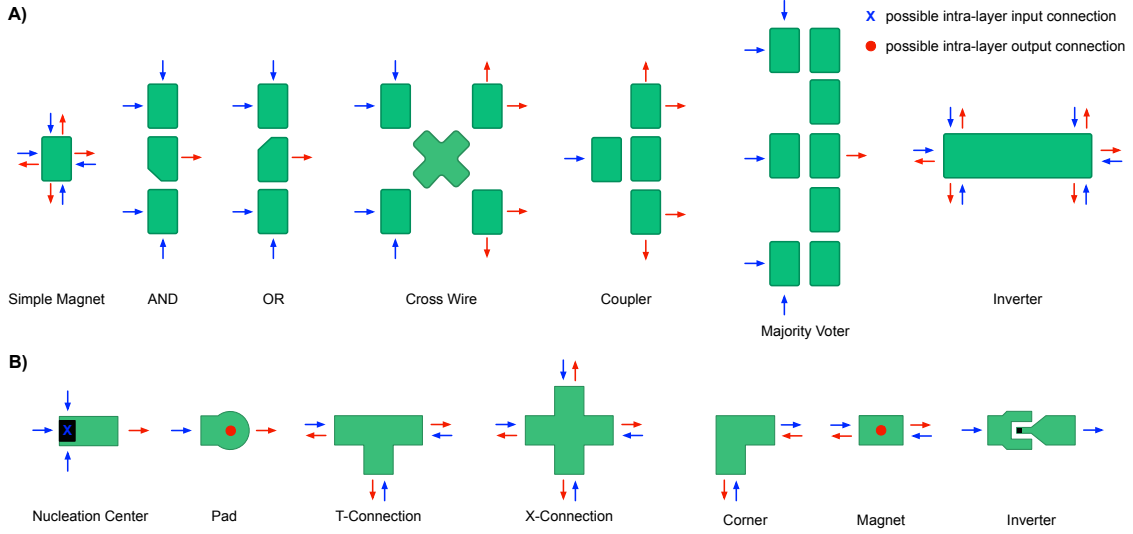


Figure 4.5: A) iNML elements and corresponding available connections. Inputs are blue, while outputs are the red arrows. B) pNML elements and relative connections. Blue crosses show input coming from another layer, while red circles are output interconnections towards adjacent planes.

4.1.1 Connection Detection

The first step of the algorithm is responsible for the detecting the real connections among elements and therefore how the information propagates inside the circuit. The pseudocode is summarized in algorithm 3. Only some elements have their inputs and outputs pre-determined, while the others have their direction defined as they are linked to their neighboring cells. Since each element has different connection, at line 2 the connectable cells are extracted. The list of connectable cells, with all the directions available for output interconnections, is used to access the input sparse matrix. Considering the layout in Fig. 4.6.A, pin “A” is the only element of *start*. This pin, according to its orientation, can be only be connected to its right. The input sparse matrix is accessed at (1,1) and “M1” is returned. The function at line 6 is used to determine if it is possible to create a connection between the current pin and the one extracted from the sparse matrix. The presence of the cell in the sparse matrix is not enough to determine if a connection is possible. The input connection of the target pin could already be occupied. Moreover, the orientation of the element in the layout provides information about its connectable sides. Indeed, the connection cannot be achieved with an element on a non-connectable side. When it is not possible to create a new connection, an error is generated. The error is used to provide a report message to the user. On the

Algorithm 3 Connection detection pseudocode

Precondition: *Inputs* and *Outputs* are the two sparse matrices, *start* is the list of the input pins of the circuit.

```
1: for each element  $\in$  start do
2:   connectable  $\leftarrow$  element.GetConnectableCells()
3:   for each cell  $\in$  connectable do
4:     if inputs.contains(cell) then
5:       input  $\leftarrow$  inputs[cell]
6:       if cell.canConnect(input) then
7:         CreateConnection()
8:         input.SetEffectiveDirection()
9:         start.insert(outputs[cell])
10:      else
11:        return ConnectionError
12:      end if
13:    else
14:      return ConnectionMissing
15:    end if
16:  end for
17: end for
```

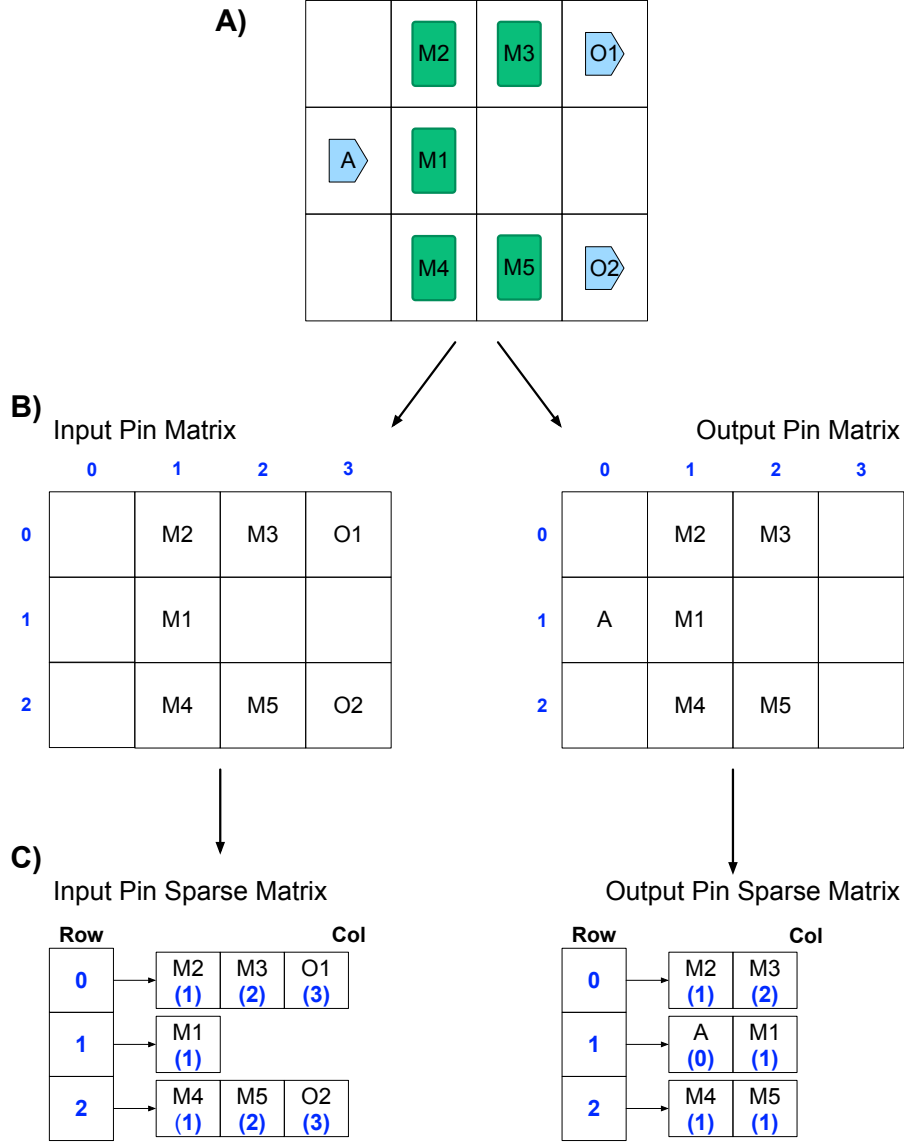


Figure 4.6: Example of how to circuit is translated in memory. A) Example of iNML coupler; B) Example of translation using two matrices; C) Example of translation using two vectors.

contrary, if the connection is available, it is formed. Each *HDL-Element* has the name of its output signal. To form a connection, the output name is inserted in the vector of input of the target cell. This information is useful when writing the VHDL netlist. In particular, the signal name is used in the *port map* of the corresponding cell. Once the new connection is defined, an element-specific function is called to define the propagation direction (line 8). These operations are repeated for all the

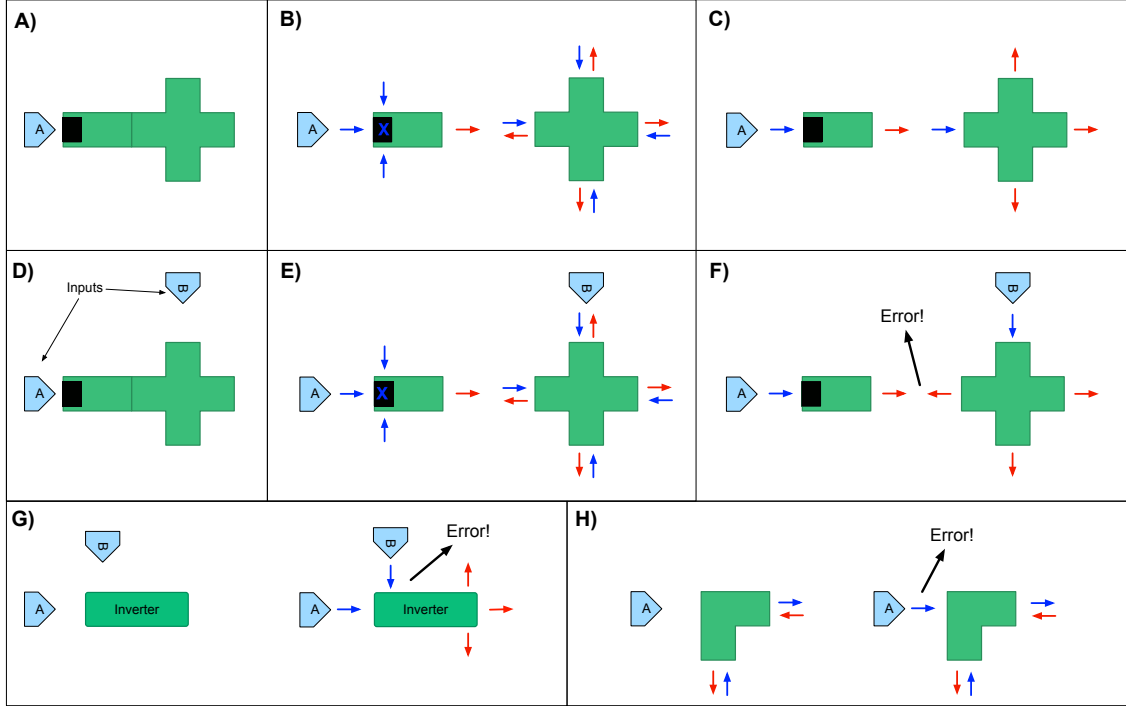


Figure 4.7: Elements connectivity examples. A) shows a portion of a pNML circuit with two elements and one input pin. B) highlights the connection available for the elements and C) shows the correct connection detection. D) represents the same circuit with extra input. E) shows the connection available and F) is an example of output-output error. G) shows a portion of the iNML circuit where an inverter has two neighboring pins. H) highlights the error due to multiple input connections to a single input element. I) is a pNML corner with a pin placed where there are no available connections, generating the error in L).

possible outputs of the current cell, and each new connected element is added to the *start* list. When an output pin is reached, nothing is added to the *start* list, thus nothing is present in the corresponding position in the output sparse matrix. Therefore, the loop ends when all the elements have been visited.

As an example, consider Fig. 4.7.A. The *X-Connection* has connectable pins in every direction. The number of pins is doubled, i.e. each side could be both input or output. During the execution of the algorithm, the left input comes from an input pin of the circuit. When that connection is defined (Fig. 4.7.B), the redundant pins need to be removed. The one connected is the only available input and the others are set as outputs, as shown in Fig. 4.7.C.

Fig. 4.7.D shows an example of an incorrect layout. Two input pins are placed close

to the *X-Connection* element. The first connected input sets the actual direction of the *Nucleation Center* cell. Subsequently, input “B” is connected defining the actual direction of the *X-Connection*. At this point, when it tries to connect the two elements, an output/output error occurs (Fig. 4.7.F). Another possible error is shown in Fig. 4.7.G: the first pin will be connected and since the inverter has only one possible input, the second will trigger an error. Finally, an input mismatch is shown in Fig. 4.7.H.

This part of the algorithm is common to both the technologies. Some elements need particular attention: the *Simple Magnet* in iNML and *Nucleation Center* in pNML. They are handled by the *Handling Technological Issues* blocks, which are described in sections 4.1.3 and 4.1.4. Before entering into the details of this step, the function detection, still common to both the technologies, is described.

4.1.2 Function Detection

Once the connections have been defined for all the elements and no errors have been generated, no more entries are available in the *start* list. The execution moves to the next step, the function detection, whose implementation is summarized by pseudocode 4. The VHDL netlist is based on a library of technological components.

Algorithm 4 Function detection pseudocode

Precondition: *Elements* is the list of all the HDL-Element of the layout.

```

1: for each element  $\in$  Elements do
2:   if element.outputs  $\neq$  connected then
3:     return OutputDisconnected
4:   end if
5:   if element.inputs  $\neq$  connected then
6:     return InputDisconnected
7:   end if
8:   EvaluateFunctionality(element.connections)
9:   WriteVHDL()
10: end for
```

The library embeds the description of all the gates and building blocks supported. During the final phase, each layout element is associated to one of the library elements, depending on its function. Some elements have a one-to-one mapping in the library: an *AND* element in the layout is mapped into an *AND* gate in VHDL. A *Simple Magnet* in iNML could either be a *Majority Voter*, a *Coupler* or a *Simple Magnet* based on its interconnections. Each element is therefore analyzed. The first check is made on the output interconnections: if an output pin is not connected to any other element an error is generated. The same test is done on the

input pins. If both tests fail, the function at line 8 is executed. This function has specific implementation for each *HDL-Element*. The number of input and output connections is used to determine the function. Once detected, the correct library component is associated with the cell. For example, if a *Simple Magnet*, in iNML, has three inputs and one output, a *Majority Voter* is associated with the element and the corresponding VHDL netlist is written. On the contrary, if the same element has one input and two or three outputs, a *Coupler* is instantiated. Similarly, for the *Nucleation Center* in pNML, three inputs identify a *Majority Voter*, while one input identifies an inverter. Another example is the pNML *Pad*, but outputs are considered in this case. The components *Pad*, *T-Connection* or *X-Connection* are associated to one, two or three outputs respectively. Finally, during the VHDL writing step, the coupling among cells needs to be evaluated. The relative position between the interconnected elements is used and the inverters are inferred in the netlist in order to model the correct coupling.

The previous part of this section described the general algorithm adopted for connection and function detection. In the following, two sub-sections will describe the technology-dependent aspects of the proposed solution.

4.1.3 Handling iNML Technological Issues

In iNML technology, the *Simple Magnet* is the most critical element during the process of detecting the connection. When it is first connected, the number of surrounding elements is evaluated. If only one or two neighboring cells are available, the proper number of connections is created and the process continues to the next element. If a *Simple Magnet* has all the connectable positions occupied by other elements, it is marked as “ambiguous” and added to a specific list called *AmbiguousItems*. No connections are defined and the *start* list is not updated. After completing the analysis of the non-ambiguous elements, the *AmbiguousItems* list is checked with the pseudocode presented in 5. In order to understand the connection of an ambiguous element, all the neighboring elements are analyzed. Each neighbor can be ambiguous, thus this connection is marked as undetermined, or non-ambiguous. In the case of non-ambiguous elements, a connection with a new element could be evaluated. If the new element has a fixed direction for input/output (it is an *AND* gate for example) or belongs to a different clock zone, the direction of the connection can be determined (line 12). Otherwise, this element becomes the new neighbor until the direction is fixed (line 9). When all the directions are completed, the if-then-else statement at line 19 is used to extract the actual number of connections. Since the resolution of an ambiguous pin could solve other connection uncertainty, the loop is interrupted as soon as a new function is identified. The resolved element is removed from the *AmbiguousItems* list and added to *start*. Thus, the normal connection algorithm 3 is executed with the updated *start* list. On the contrary, if it is impossible to determine the function of

Algorithm 5 Ambiguous element resolution

Precondition: *AmbiguousItems* is the list of all the ambiguous elements found.
start is the list of the non-ambiguous elements.

```
1: Iterations  $\leftarrow$  0
2: while iterations  $\leq$  2 do
3:   for each element  $\in$  AmbiguousItems do
4:     for each neighbor  $\in$  element.getConnectable() do
5:       while neighbor  $\neq$  fixedDirection do
6:         if neighbor.outputs  $>$  1 then
7:           neighbor  $\leftarrow$  Undetermined
8:         else
9:           neighbor  $\leftarrow$  neighbor.output
10:        end if
11:      end while
12:      element.defineNeighborDirection()
13:    end for
14:
15:    inputs  $\leftarrow$  element.inputs
16:    outputs  $\leftarrow$  element.outputs
17:    undetermined  $\leftarrow$  element.undetermined
18:
19:    if inputs = 3 | (inputs = 2 & undetermined = 1) then
20:      element  $\leftarrow$  MajorityVoter
21:      start.append(element)
22:      AmbiguousItems.pop(element)
23:      break
24:    else if (inputs = 1 & undetermined  $<$  2) | (outputs  $>$  1
25:      & undetermined = 1) then
26:      element  $\leftarrow$  Coupler
27:      start.append(element)
28:      AmbiguousItems.pop(element)
29:      break
30:    else if (inputs + undetermined) = 3 & outputs = 1 &
31:      iteration  $\neq$  0 then
32:      element  $\leftarrow$  MajorityVoter
33:      start.append(element)
34:      AmbiguousItems.pop(element)
35:      break
36:    end if
37:  end for
38:  Iterations  $\leftarrow$  Iteration + 1
39: end while
```

an element, the next one in the *AmbiguousItems* list is analyzed. If no ambiguous elements are resolved during the loop on the list, the number of iterations is increased and therefore the condition at line 29 becomes true. This condition, lines 29-32, is used to “guess” the function of an element. The *Majority Voter* function is assigned to the first ambiguous element with exactly one output and a total of three connections, considering inputs and undetermined neighbors. Usually, this situation is very uncommon and correct circuits are solved avoiding the need for two subsequent analyses of the *AmbiguousItems* list. If a guess is needed, there is probably an error in the layout. Indeed, this solution can detect errors. After guessing the function, the algorithm tries to connect all the neighboring cells as outputs, and conflicts appear in the circuit. Exceptions exist but are very uncommon: a symmetric circuit belonging to a single clock zone is correctly solved by the algorithm, but the function assignment is unpredictable. As mentioned, these kinds of circuits have been used as a corner case during the validation of the algorithm and without any practical function.

4.1.4 Handling pNML Technological Issues

In pNML technology, the critical element is the *Nucleation Center*. Thanks to the fixed direction of all the pNML elements, in this technology the solution is simpler compared to iNML. During the connection generation, it is important to know the number of inputs for all the *Nucleation Centers* in the circuit. In the case of an even number of inputs, an error is present in the layout. To handle this issue, a *Set* data structure storing the ambiguous *Nucleation Center* has been introduced. When a *Nucleation Center* is connected, if the number of connections is even, its position is added to the *Set*. On the contrary, it is removed from the *Set* if an odd number of connections is present. When the *start* list is empty, all the positions stored inside the *Set* are marked as error and added to the output log file.

4.1.5 Performance and verification

The presented algorithm was tested with several circuits. Fig. 4.8 shows the flow adopted to test the algorithm. Different layouts were used, designed by hand with MagCAD [35] or generated with ToPoliNano [36] starting from a behavioral description. The FUNCODE algorithm was used to extract the VHDL netlists from the circuit layouts. The generated netlists were verified through VHDL simulations by using ModelSim from Methor Graphics [31].

Fig. 4.9.A shows an example of a circuit in iNML technology. It has three inputs (A, B and C) and two outputs (O1 and O2). It is composed of several *Simple Magnet* items and one inverter. Furthermore, two groups of magnets (the blue squares in the figure) implement the *Coupler* and the *Majority Voter* gates. Once the layout is processed by FUNCODE, it produces the VHDL netlist, of

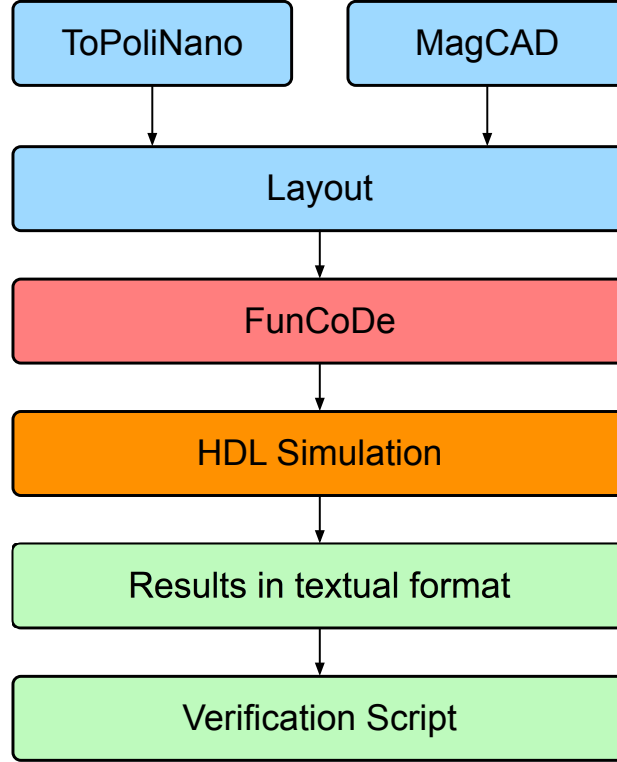


Figure 4.8: Methodology adopted to verify the correct behavior of the proposed FUNCODE algorithm.

which an excerpt is shown in Fig. 4.9.B. The highlighted cells report automatically detected functions. Each cell label is named with its position, and a different library element is associated to it. For example *cell5_2* is associated to a *NML_cell* and the number of interconnections is specified using the *generic map* directive of VHDL. In the *port map*, the corresponding connections are made. Two issues can be noticed by observing the connections of the *Majority Voter* by name associations. First, a negated function is inserted to model the AF coupling of the input on the left. Second, the signal naming format. The signal *ls4_2o1* inverted is assigned to *CELL_IN(0)*. For each cell connection, one signal is defined in VHDL. The name is constructed with the format *ls + cellposition + _o + outputnumber*. The interconnections to the *Majority Voter* come from the cell on the left, above and below. For the *Coupler* the same type of cell is used but different interconnections are specified in the *generic map*. A different cell is instead used for the inverter. The simulation of the generated VHDL is shown in Fig. 4.9.C. The majority between A, B, C determines the O1 output, whereas the output O2 is the inverted value of input C. It is possible to observe that the output signals take into account the

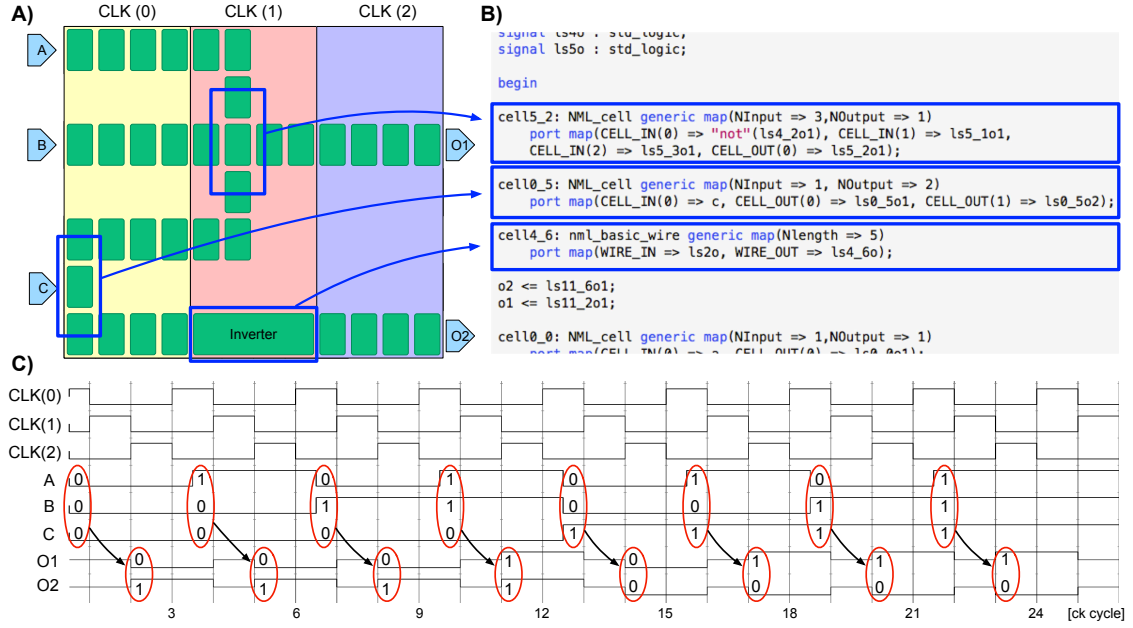


Figure 4.9: A) Example of iNML layout; B) Excerpt of the generated VHDL that show the automatically detected functional blocks; C) Simulation of the sample circuit that take into account the latency introduced by the target technology.

signal delays introduced by the clock scheme. The circles highlight the input-output correlation. The latency is due to the clock mechanism. Inputs have to travel through the circuit before reaching the outputs. Therefore, the generated VHDL takes into account the physical behavior in terms of timing introduced by the target technology.

The same approach was used for all circuits discussed in this section. Before starting the analysis of the results, the selected circuits are presented. The layouts for the iNML technology have been generated using ToPoliNano. We adopted two synthesis approaches to generate the circuits. The first is based on standard *AND*, *OR* and inverters gates. The second approach is based on majority gate synthesis. This approach makes larger circuits possible. The full adder, the Ripple Carry Adders (RCA) with different data parallelism and the array multiplier belong to the first category. Furthermore, the c17, c432, c880 and c499 circuits from the IS-CAS85 benchmark suit [9] were generated using the same strategy. The remaining circuits are based on the second approach, performed with SIS and ABC [6], two tools developed at the University of California, Berkeley. The circuits based on the majority gate synthesis are part of the design of an Advanced Encryption Standard (AES) Substitution-BOX (S-BOX). The S-BOX is composed of the following sub-circuits: a 4-bit xor (xor4), a matrix multiplication circuit (affine_transformation),

Table 4.1: Verification example of circuit c432. Inputs are applied to both the model and the FUNCODE generated netlist, outputs are collected and analyzed.

Inputs Vectors				Model outputs	FUNCODE netlist outputs
E[8;0]	A[8;0]	B[8;0]	C[8;0]	PA PB PC	Chan[3;0]
11111111	01111111	01111111	01111111	1 1 1 0000	1 1 1 0000
11111111	10111111	10111111	10111111	1 1 1 1111	1 1 1 1111
11111111	11011111	11011111	11011111	1 1 1 1110	1 1 1 1110
11111111	11101111	11101111	11101111	1 1 1 1101	1 1 1 1101
11111111	11110111	11110111	11110111	1 1 1 1100	1 1 1 1100
11111111	11111011	11111011	11111011	1 1 1 1011	1 1 1 1011
11111111	11111101	11111101	11111101	1 1 1 1010	1 1 1 1010
11111111	11111110	11111110	11111110	1 1 1 1001	1 1 1 1001
11111111	11111111	01001111	10111111	1 1 1 1000	1 1 1 1000
11111111	11111111	10111111	01000000	0 1 0 0110	0 1 0 0110
11111111	11111111	11011010	11111111	0 1 0 1111	0 1 0 1111
11111111	11111111	11101011	11111111	0 1 0 1110	0 1 0 1110
11111111	11111111	11110011	11111111	0 1 0 1101	0 1 0 1101
11111111	11111111	11111011	11111111	0 1 0 1100	0 1 0 1100
11111111	11111111	11111101	11111111	0 1 0 1011	0 1 0 1011
11111111	11111111	11111110	11111111	0 1 0 1010	0 1 0 1010
11111111	11111111	11111111	01111111	0 1 0 1001	0 1 0 1001
11111111	11111111	11111111	10111111	0 1 0 1000	0 1 0 1000
11111111	11111111	11111111	11011111	0 0 1 0000	0 0 1 0000
11111111	11111111	11111111	11101111	0 0 1 1111	0 0 1 1111
11111111	11111111	11111111	11110111	0 0 1 1110	0 0 1 1110
11111111	11111111	11111111	11111011	0 0 1 1101	0 0 1 1101
11111111	11111111	11111111	11111101	0 0 1 1100	0 0 1 1100
11111111	11111111	11111111	11111111	0 0 1 1011	0 0 1 1011
11111111	11111111	11111111	11111111	0 0 1 1010	0 0 1 1010
100000010	100000010	100000010	100000000	0 0 1 1001	0 0 1 1001
011111101	011111101	011111101	011111100	0 0 1 1000	0 0 1 1000
111111111	111011011	000100100	000100100	1 0 0 1101	1 0 0 1101
111111111	111101001	111111111	111111111	1 0 0 1100	1 0 0 1100
111111111	011111111	100000000	100000000	1 0 0 1101	1 0 0 1101
111111111	001111111	111111111	001111111	1 0 0 0000	1 0 0 0000
000000000	000000000	000000000	000000000	1 0 1 0111	1 0 1 0111
000000000	000000000	000000000	000000000	0 0 0 0000	0 0 0 0000

a Galois Field Isomorphic Mapping circuit (GFMap), a 4-bit Galois Field multiplication (gfmul4), a Galois Field Inversion circuit (GFInv), a circuit performing

square exponential computation plus constant multiplication (SquareEX) and finally the inversion of the Galois Field Isomorphic Mapping (GFmapinv). The pNML circuits were manually designed using MagCAD. In pNML technology, the *Majority Voter* and the inverter are used to perform all the logic functions. Therefore, the number of gates in the same circuit is different with respect to the iNML version. The ripple carry adders with different input parallelism were used. The 32-bit version of the RCA has been presented in [21]. Furthermore, a decoder

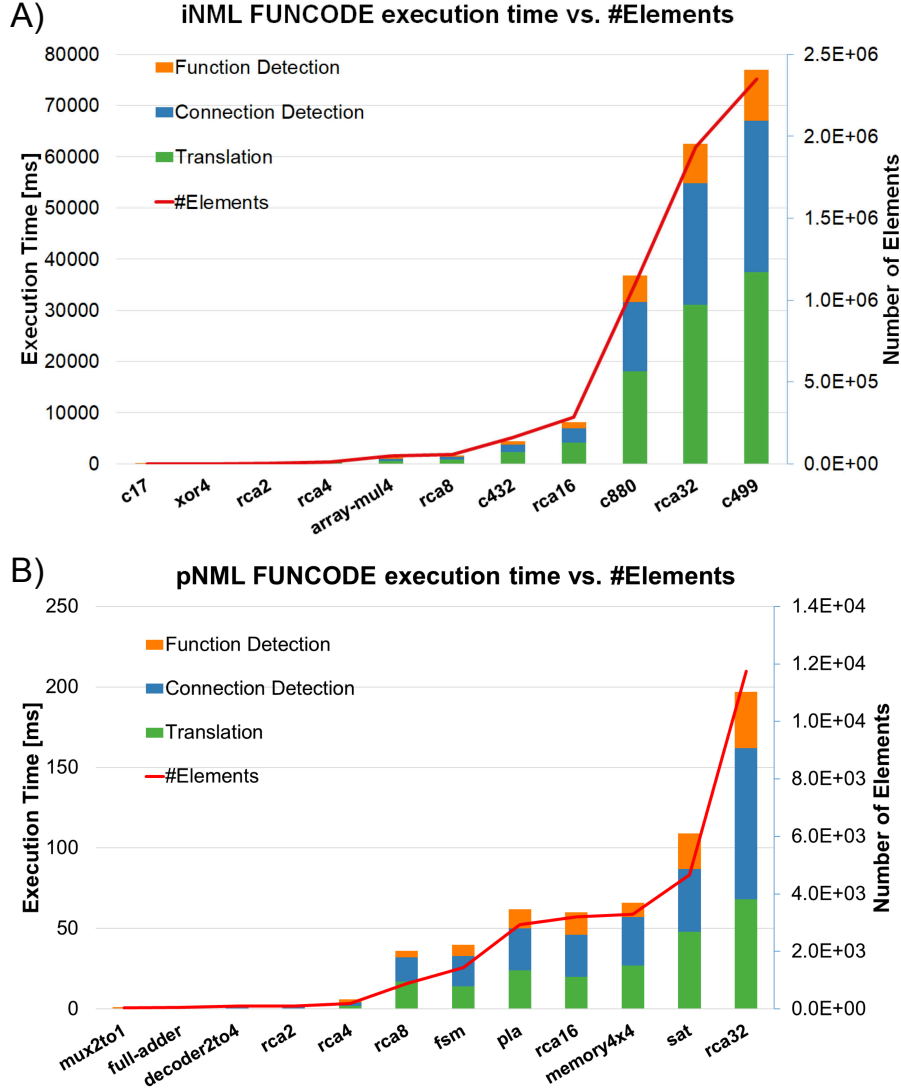


Figure 4.10: A) iNML FUNCODE execution time as a function of the total number of elements; B) pNML FUNCODE execution time as a function of the total number of elements.

Table 4.2: Generation algorithm timing report using different iNML architectures with an increasing number of elements.

Circuit Name	#Gates	#Inputs/ #Outputs	#Elements	Translation Elements [ms]	Connection Detection [ms]	Function Detection [ms]	Total Time [ms]
c17	6	5/2	233	2	2	1	5
xor4	20	8/4	260	5	3	1	9
squareXE	14	4/4	1688	59	22	19	100
rca2	26	5/3	2687	39	26	23	88
rca4	52	9/5	11987	198	128	92	418
GFInv	34	4/4	16645	291	195	129	615
GFmapinv	55	8/8	37145	578	369	228	1175
array mul4	148	8/8	47972	594	404	201	1199
GFmap	56	8/8	55864	782	487	235	1504
rca8	104	17/9	57502	815	525	324	1664
affine_transform	70	8/8	85206	1341	883	496	2720
c432	160	36/7	161114	2236	1500	634	4370
rca16	208	33/17	285874	4209	2727	1202	8138
gfmolt4	199	8/4	379258	5753	4035	1528	11334
c880	383	60/26	1093246	18031	13574	5242	37027
rca32	416	65/33	1935112	31135	23641	7747	62523
c499	202	41/32	2348735	37431	29635	9921	76987

Table 4.3: Generation algorithm timing report using different pNML architectures with an increasing number of elements.

Circuit Name	#Gates	#Inputs/ #Outputs	#Elements	Translation Elements [ms]	Connection Detection [ms]	Function Detection [ms]	Total Time [ms]
mux2to1	13	2/1	36	0	0	1	1
full-adder	8	3/2	55	0	1	0	1
decoder2to4	21	2/4	94	0	1	1	2
rca2	21	5/4	94	0	1	1	2
mux2to1x3	49	7/3	162	0	1	0	1
rca4	43	9/5	194	2	2	2	6
carry select	149	9/5	838	2	2	1	5
rca8	156	17/9	886	17	15	4	36
fsm	116	5/12	1438	14	19	7	40
pla	513	51/14	2939	24	26	12	62
rca16	461	33/17	3199	20	26	14	60
memory4x4	972	5/14	3294	27	30	9	66
sat	1050	10/19	4662	48	39	22	109
rca32	1391	65/33	11745	68	94	35	197

with two inputs and four outputs and a memory array 4x4, presented in [18], were used. Other circuits adopted are a finite state machine (FSM) with four states

and a programmable logic array (PLA) with three inputs, four minterms, and four maxterms, both presented in [21]. A circuit implementing the summed area table [34] (SAT) algorithm was analyzed. Finally, a hierarchical version of a 4-bit carry select adder (CSA) circuit was designed: the building blocks used are 2to1 multiplexers and 2-bit RCA. We performed VHDL simulations to verify the correct logic behavior of the generated netlists. Given the complexity of the benchmark circuits, the process was automated via a script. A behavioral code, or model in case of custom circuits, was simulated and the output was saved on a file. The extracted FUNCODE netlists were fed with the same input and all the produced outputs were written into a file. The latency introduced in the new structures led to a larger file, with many more output lines. A script was used to extract the useful data in the simulation output and to compare these with the data obtained with the model. Two data are needed in order to run the script: circuit latency and number of clock cycles without changing the input vectors. The circuit latency is used to remove the firsts outputs, where no data is present since inputs are still loading the *pipeline*. The number of clock cycles is used to skip clock cycles where the outputs would have been equal.

Table 4.1 shows an example of the verification process for the c432 circuit. Circuit c432 is a 27-channel interrupt controller. The input channels are grouped into three 9-bit buses (named A, B, and C), where the bit position within each bus determines the interrupt request priority. A fourth 9-bit input bus, named E, enables and disables interrupt requests within the respective bit positions. The seven outputs PA, PB, PC and Chan[3:0] specify which channels have acknowledged interrupt requests. Only the channel of highest priority in the requesting bus of highest priority is acknowledged. The input vectors are listed in the first column of Table 4.1. Since the circuit is combinational the outputs of the reference description are immediately available without any latency. They are reported in the second column. The output file of the FUNCODE netlist was elaborated through the script and the resulting values are reported in the third column.

All the analyses reported in this section were performed on an Intel Core i-7 7700, equipped with 16GB of RAM, running CentOS 7 operating system.

The tests show that the algorithm complexity is linear with the number of elements in the circuit. Fig. 4.10 shows an analysis of the execution time for the different parts of the algorithm for the two technologies. The performance of the iNML version is reported in Fig. 4.10.A. The different circuits are listed on the x-axis. The line shows the number of elements of the circuit and is plotted on the secondary y-axis. The stacked bars show the time needed to execute the algorithm. The graph shows that the total execution time increases linearly with the number of elements. Furthermore, the different parts of the algorithm are associated with different portions of the stacked bars. The translation to the *HDL-Elements* is in green, while the connection detection is in blue. The connection time also includes the handling of technological issues. The function detection and the VHDL netlist

writing are represented by the orange part of the bars. The bars reveal that the translation time is the longest, and is almost half of the total time. In fact, during the translation, different data structures are allocated and different operations are performed on each element. Furthermore, the connection detection execution impacts the total time for approximately 37%. Finally, the function detection is the shortest. Once all the connections have been defined, the function is assigned according to the connected elements. A similar analysis was performed for pNML technology. Fig. 4.10.B shows the trend of the execution time over the number of elements. As expected, also for the pNML case, a linear complexity is achieved since the core of the algorithm is shared by both the technologies. Differently from iNML technology, the most time-consuming portion of the algorithm is the connection detection, which is almost 45% on average. The translation time is about 38% of the total time and the function detection only impacts 18% of the time. The differences between the two technologies derive from the fact that the *Simple Magnet* in iNML technology, the most common item in the circuits, has high connectivity. In fact, during translation, a huge number of redundant connections are generated, and this impacts on the final performance. By contrast, in pNML technology, the global clock mechanism increases the complexity of the interconnection detection: without the constraint of sequentiality given by the clock zones an additional degree of freedom is present.

Tables 4.2 and 4.3 show the results for iNML and pNML technology respectively. The tables are organized as follows. The first column is the circuit name. Circuit details, such as the number of gates, inputs, outputs, and elements are shown in the subsequent columns. In the fifth column the approach, flat or hierarchical, is presented. The last columns are used to show the time performance of the algorithm. As in Fig. 4.10, the times are divided in translation, connection and function detection. The last column is the total time needed by the algorithm.

4.2 Technological properties change

One of the most important aspects while studying a new technology is the possibility to analyze the effect of different parameters on the device performance. Even in circuit exploration, this kind of analysis is very important. The netlists extracted by the FunCoDe algorithm are not enough for this purpose. Indeed, the technological library developed to perform the simulations is based on equations that model the devices of the selected technology. Those libraries have been therefore split into two different files: the library itself and a definitions file. The former contains the descriptions of the technological elements and the model equations. It is the same for every circuit designed with the selected technology. On the contrary, the latter embeds all the physical parameters that could be changed for the selected technology. In this way, by changing the variables inside the definition files it is

possible to analyze the effect of those changes on the circuit. Given that MagCAD has been developed in order to simplify the design and therefore a circuit-level exploration of emerging technologies, manual modification of the definition files was not enough. Furthermore, a potential user of the tool could be un-expert of VHDL syntax. Therefore, a graphical interface was developed for parameter variation. During netlist extraction, if no error were generated, a window, Fig. 4.11, is showed to the user depending on the select technology. The changes in the window will be transferred to the definitions file. Furthermore, physical limits for the available parameters are specified: when the user confirms its selection, the out-of-range elements will be highlighted. The process is iterated until all the inserted parameters are valid.

4.3 Molecular plugin

QCADesigner [47] enables the design of circuit based on MolQCA technology. However, it is based on the general idea of the technology, but no physical implementation is really used in the tool. In the ToPoliNano framework, the physical implementation is very important. MolQCA technology support was added to the framework, starting from MagCAD. The technological plugin for MolQCA has been developed. MagCAD is able to load new technology at start-up as plugins. iNML and pNML plugins have been developed in the first versions of the tool. In order to define a new plugin, the technological basic blocks were selected. As described in section 2.1, the bis-ferrocene molecule has been used. The molecule is represented by three circles. The two bigger ones are the logic dots of the molecule, while the small central circle is the null dot. Indeed, a couple of molecules, therefore a QCA cell, defines the basic element. This choice results in a uniform squared grid. In this way, elements in the grid can be rotated without the risk of collisions and superposition. On the other hand, using the single molecule as building block could give more flexibility to the user, enabling the possibility to change properties of the single molecule. The main aim of MagCAD is a fast design of circuits based on different technology. Indeed, the exploration of different technological solutions or the modeling of process variation is a key aspect of the tool. For this reason, a more sophisticated technological widget has been developed, Fig. 4.12. The widget is divided into two different parts: each portion is associated with one of the two molecules defining the cell. In this way, it is possible to set different properties for each molecule. Rotation angle and 3D displacement could be changed by the user. Furthermore, it is also possible to remove one of the two molecules, simulating an error in the fabrication process. During the layout phase, rotation of the cell or molecule properties could lead to a difficult understanding of which molecule is associated with the different parts of the widget. For this reason, different colors are used to draw the two molecules. Furthermore, a smaller circle is inserted in one of

A)

Vhdl physical parameters choice: prova_inml

Clock Pulse Constants

Clock period [s]: 3.0e-8

High level current [A]: 2.0e-3

Low level current [A]: 1.0e-3

Duration of transition 0 -> I1 [s]: 1.0e-10

Start of transition I1 -> I2 [s]: 5.0e-11

End of transition I1 -> I2 [s]: 1.0e-10

start of transition I2 -> 0 [s]: 1.0e-10

Base Nanomagnet Constants

Magnet width [nm]: 60.0

Magnet height [nm]: 90.0

Magnet thickness [nm]: 10.0

Horizontal space [nm]: 25.0

Vertical space [nm]: 10.0

Input-output cell delay [ns]: 0.0

Multiplicative factor for energy related to the switch of a magnet: 30.0

Multiplicative factor for energy related to the switch of a magnet: 30.0

Temperature [K]: 293.0

Restore Defaults OK Discard Changes

B)

Vhdl physical parameters choice: New_2

Fields Constants Parameters

Field pulse amplitude [Oe]: 560.0

Clocking field amplitude [Oe]: 560.0

Intrinsic Pinning Field [Oe]: 190.0

Critical nucleation probability: 0.50

Critical nucleation prevent probability: 0.50

Exchange stiffness [J/m]: 1.3e-11

Apex angle [degree]: 51.5

Notch width [m]: 5.4e-8

Activation volume [m³]: 1.26e-23

Depinning probability: 0.98

Coupling field for the via [Oe]: 75.0

Base Components Constants

Grid size [m]: 3.0e-7

Width of the domainwall [m]: 2.2e-7

Interlayer space [m]: 7.0e-8

Numerical prefactor in depinning regime [m/s]: 1.14e+6

Pinning energy barrier [J]: 12.1

Numerical prefactor in flow regime [m/s]: 31.9

Domain wall mobility [Oe*m/s]: 0.042

Inverter coupling field [Oe]: 153.0

MV coupling field [Oe]: 48.0

MV5 coupling field [Oe]: 29.0

Attempt frequency [Hz]: 2.0e+9

Saturation magnetization [A/m]: 1.4e+6

Thickness [m]: 3.2e-9

Stack thickness [m]: 6.2e-9

Effective anisotropy [J/m²]: 2.0e+5

Fib irradiation factor: 0.1416

Volume of the ANC [m³]: 1.68e-23

Temperature [K]: 293.0

Restore Defaults OK Discard Changes

Figure 4.11: Technology parameters selection. (A) iNML and (B) pNML.

the two logic dots. The clock zone property, represented by the background color, can be modified for the entire cell. The technological property that is common to all the layout is the inter-molecular distance. During the technology selection, it is possible to set this value, and it will be used to define the grid dimension. As mentioned before, a square grid is used. In order to have the same distance among

The image shows a software window titled "Molecule properties". It is divided into two columns, "Molecule 1" and "Molecule 2". Each column has a checked checkbox labeled "Enabled". Below the checkboxes are four rows of input fields: "Angle", "X Shift [pm]", "Y Shift [pm]", and "Z Shift [pm]". Each input field contains the number "0". At the bottom of each column are two buttons: "Set properties" and "Reset properties".

	Molecule 1	Molecule 2
Enabled	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Angle	<input type="text" value="0"/>	<input type="text" value="0"/>
X Shift [pm]	<input type="text" value="0"/>	<input type="text" value="0"/>
Y Shift [pm]	<input type="text" value="0"/>	<input type="text" value="0"/>
Z Shift [pm]	<input type="text" value="0"/>	<input type="text" value="0"/>
Buttons	<input type="button" value="Set properties"/> <input type="button" value="Reset properties"/>	<input type="button" value="Set properties"/> <input type="button" value="Reset properties"/>

Figure 4.12: MolQCA property widget.

all the molecules, also considering adjacent cells, the double of the inter-molecular distance is used as the grid unit. Figure 4.13 shows the resulting dimension in the default settings: inter-molecular distance equal to 1 nm.

Even if the bis-ferrocene is the only molecule considered at the moment for the analysis described in part II, this may not be in the future developments. Therefore different molecules could be defined in the plug-in. For test purpose, the butane molecule has been defined. Even if the molecular structure is not compatible with the QCA paradigms, it is possible to insert those molecules in the layouts. Since the grid dimensions are fixed with the inter-molecular distance the molecule dimensions are used to scale the representation. For example, the butane molecule defined in the tool has a 600 nm distance among the dots, while the bis-ferrocene has 1000 nm. When the two cells are placed in the same layout the butane molecules will be smaller compared with the bis-ferrocene.

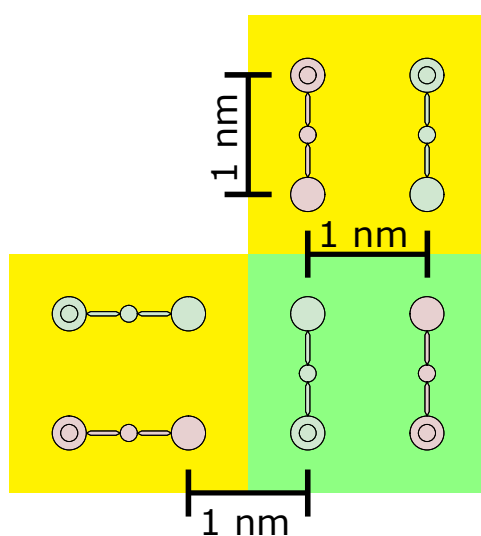


Figure 4.13: Highlight of the distance among the molecules. Each dot, in the same molecule or of two different molecules is distant at least 1 nm from its neighbors.

Part II

Field-Coupled Nanocomputing Simulator

The ToPoliNano framework has been designed in order to extend the classic CMOS flow also to emerging technologies. Place & route and custom layouts have been already described in the previous part with the overview of the two tools. MagCAD and the FunCoDe algorithm can be used to simulate the layout produced with the framework but it relies on the usage of external tools. Furthermore, even if the library of technological elements is based on models of the physical behavior, standard HDL simulators are not designed to perform those kinds of simulations. Indeed, physical simulators are available and are used to perform the device level analysis. Unfortunately, increasing the number of elements it is no more possible to perform physical simulations in a reasonable amount of time. The high accuracy achievable with that approach costs in time and computation power demand. For those reasons, a new simulator was developed, and inserted in the framework: Field-Coupled Nanocomputing Simulator (FCNS). In the following chapters the simulator will be described and its performance analyzed.

Chapter 5

Physical simulators

Before entering in the details of the proposed simulator, an introduction about the available simulation tools is presented. Speaking of magnetic technologies micro-magnetic simulators are the reference ones. In particular, OOMMF [15] and Mumax3 [44] are widely used by the scientific community. For what concern MQCA technology instead a different approach is present. Physical simulators for molecules, like *Gaussian* [17] and ORCA [24], are limited to ab-initio simulation, where the characteristic of a single molecule can be extracted. It is possible to simulate also a few molecules together but with some limitations that will be later discussed. For this reason, a simulation tool has been developed in the VlsiLab of Politecnico di Torino: SCERPA [48] [3]. SCERPA was used as a reference simulator during the development of the proposed software. Finally, QCADesigner can be used to design and simulate circuits based on QCA technology. In the following sections, the state-of-the-art simulators are described.

5.1 Micro-magnetic Simulators

In order to simulate the evolution of the element of a circuit a time-dependent motion of the magnetization has to be simulated. The transition among two stable states based on external magnetic fields is described by the Landau-Lifshitz-Gilbert (LLG) equation [13]. The formula describes the magnetization dynamic considering an applied effective magnetic field:

$$\frac{\partial \mathbf{M}}{\partial t} = -\gamma \mu_0 (\mathbf{M} \times \mathbf{H}) + \frac{\alpha \gamma}{M_s} \left(\mathbf{M} \times \frac{\partial \mathbf{M}}{\partial t} \right) \quad (5.1)$$

In the formula, γ is the gyromagnetic ratio while α is the damping constant. This differential equation is used to describe the orientation of the magnetization vector of an element. The equation is usually solved numerically in order to evaluate the system evolution. Both the micromagnetic simulators cited before are based

on the finite differences method (FDM) and this equation. The region of space that has to be simulated is described defining geometries of different materials, like permalloy and also the empty spaces. Then, the region is divided into smaller parts using a grid mesh: in this way the equation is solved for every element of the mesh. The interactions among all the mesh elements and an optional external field are grouped in the Heff term of the equation. Since the equation describes the dynamic behavior of the magnetic elements, the result of the simulation is the evolution during time of the tested region. The simulation time is divided into very small time steps. The changes in the magnetization direction are considered to be infinitesimal. The computation could be done in parallel: at each step, all the elements can be evaluated. Successively the values are updated and simulation moves to the next step. The main difference between OOMMF and mumax3 is that the former runs on CPUs, while the latter is developed to run on GPUs. Parallel evaluation of a time step could lead to impressive improvements with a high number of elements. Indeed, Mumax3 outperforms OOMMF even on a low-end GPU. Even with parallel execution, these kinds of simulations are extremely slow. Increasing the complexity of the simulated region, meaning the number of elements to be simulated, causes an exponential growth of the mesh dimension. For example, a single magnet, 60x90x20 nm, with a grid size of 5 nm results in 864 grid elements. This number increases to 1872 by adding another aligned magnet, with the same dimensions, at 10 nm distance. Therefore, it is not feasible to simulate entire circuits with this approach.

5.2 Molecular Simulators

In order to study the behavior of electrons inside a molecule, quantum chemistry should be considered. This approach could be used solving the Schrodinger Equation for the target molecule, resulting in a probability of finding an electron in a particular position. Unfortunately, that equation can be solved numerically only for the hydrogen atom. Therefore, some approximation are used to handle more complex molecules. Among the different approaches available, ab-initio simulations will be here described. This kind of approximations are based on quantum mechanical laws and physical constants. Indeed, ab-initio methods are accurate and computational intense. Using *Gaussian* it is possible to perform ab-initio simulation on the desired molecules. The main drawback of this approach is that computation cost becomes unsustainable with a very small number of elements: this kind of simulation on a circuit based on four molecules is require a huge amount of time. Furthermore, it is not possible to define several zones where different electric fields should be applied, as in the clock zone mechanism normally used for MolQCA. For these reasons, a different approach was introduced. The ab-initio simulation are used to extract the molecule characteristics, that are then used to model the

behavior of complex systems. SCERPA (Self Consistent Electrostatic Potential Algorithm) has been developed in order to perform simulation of complex molecular architectures. It consists in Matlab scripts capable of loading a circuit description and perform simulation taking into account different clock zones and input values. Differently from micro-magnetic simulators the model used inside SCERPA does not consider molecular dynamic evolution during time. Instead, a equilibrium simulation engine is used. During simulation, molecule are evaluated until the equilibrium is reached. Then new clock and input values are applied and the process iterates again until a new stable state is reached.

5.3 QCADesigner

Among the available simulator for QCA technologies QCADesigner [47] has to be listed. It is a tool that enables the design and simulation of circuits based on QCA technology. Well known to the scientific community, it gives all the functionality useful to a designer. While it is very similar to MagCAD the main difference lays in the fact that QCADesigner is not based on a particular physical implementation of QCA technologies. This means that the basic cells are the ones described in the beginning of chapter 2. Squared cells with four dots are the only available elements in the tool. Therefore it is almost impossible to explore also other QCA technologies, like the magnetic ones. By the way, QCADesigner can perform simulations of the described layouts. Two different simulation engines are present: Coherence Vector and Bistable engine. The former, that can model dissipative effects as well as perform a time-dependent simulation of the design. The simulation engine evaluates the equation of motion (a partial differential equation) using an explicit time marching algorithm. The latter does not include any timing information. The simulation engine calculates the state of each cell with respect to other cells within a preset effective radius. This calculation is iterated until the entire system converges within a predetermined tolerance. Once the circuit has converged, the output is recorded and new input values are set. This approximation is sufficient to verify the logical functionality of a design it cannot be extended to include valid dynamic simulation. The bistable simulation engine can simulate a large number of cells very rapidly, and therefore provides a good in-process check of the design.

In the next chapters a detailed description of the proposed simulator will be presented.

Chapter 6

FCNS

Before entering in the details of the simulator, an overview of the approach will be given. The idea is to have a multi-technological system, capable of handling different implementation of FCN. Fig. 6.1 shows the general principle of the simulator. Even if different implementations rely on different physical phenomena and interac-

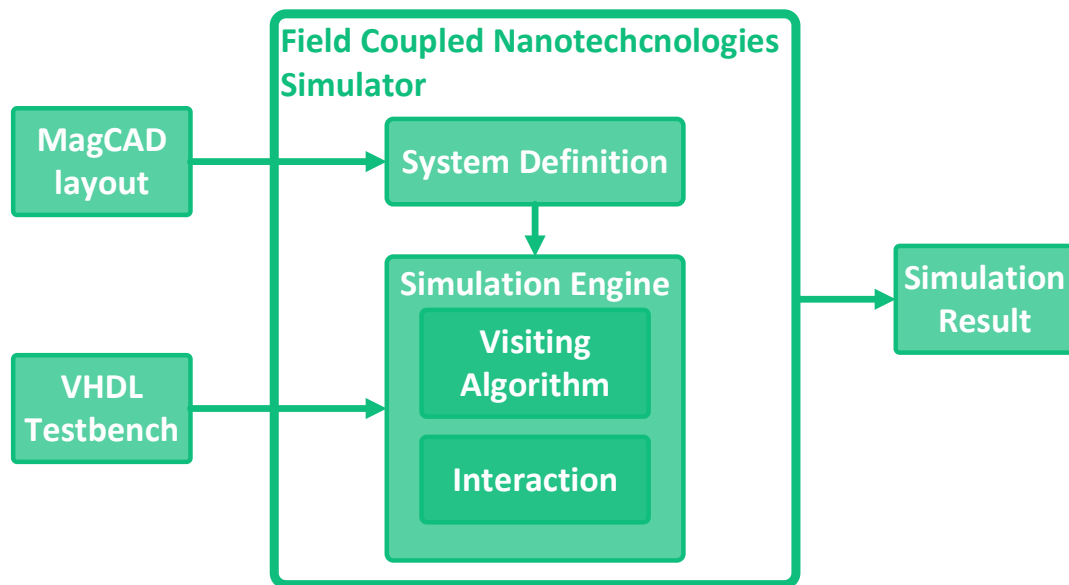


Figure 6.1: Schematic overview of FCNS

tions, common characteristics are present. In FCN the information is moved thanks to the interactions among the elements. The magnetic or electrical field generated

by one of the elements will influence the other ones, and information will be propagated. In particular, each element is influencing every other element of the system. This statement is true for every FCN. Therefore, the visiting algorithm has been developed in order to be technology independent. Another aspect common to the different implementations is the division in evaluation steps. During simulations, new values are evaluated freezing the state of the system. After the evaluation of a new value for every element, an update function is called: during the update the new value is set and the process is repeated for the entire simulation. Above the general core of the system, technology-specific modules are used. Each technology defines a set of functions, like interaction models and stability criteria, that will be called during execution. The flexibility of the simulator enables the insertion of specific properties of the technological elements. In this way it is possible to check how process variations impact the technology. The simulator is developed in C++ and Boost libraries [20] have been used. Furthermore, it is multiplatform thanks to the Qt framework. The following sections will enter the details of the tool.

6.1 General Flow

Before starting the analysis of the simulation process, it is useful to describe how the inputs, both circuit and stimuli, are loaded in the simulator. Currently, the simulator has been developed as a standalone command-line application, but it is going to be integrated inside the ToPoliNano tool. Therefore the circuit description is a layout in the ToPoliNano format (“.qll”). ToPoliNano can parse the “.qll” files and show to the user the layout. When the user selects to perform a simulation an instance of *Simulation Controller* is created. This object accepts as input the list of the elements and the technological settings relative to the circuit, both already loaded in the internal data structures. In the current version of the simulator, the *Simulation Controller* is instantiated inside the main of the program. Therefore the parser of the “.qll” files is part of the simulator. Since no GUI is present, the layout is directly used to perform simulation, without the creation of a graphical representation. A similar approach is used for the input stimuli. In order to simplify the process of integration of FCNS inside ToPoliNano the following approach has been used. The data structure resulting after parsing a testbench is written to a file using a temporary debug output of ToPoliNano. The resulting file is read in the FCNS main and translated in the original data structure that is then passed to the simulation controller. In this way, the controller is ready to be integrated inside ToPoliNano. The other parameters for the simulation process are now retrieved as command-line arguments.

The *Simulation Controller* is the main class of the simulator. It receives the list of elements and the technological settings, Fig. 6.2 shows a detailed flow chart of the simulator structure. The first step of the simulation process is the creation

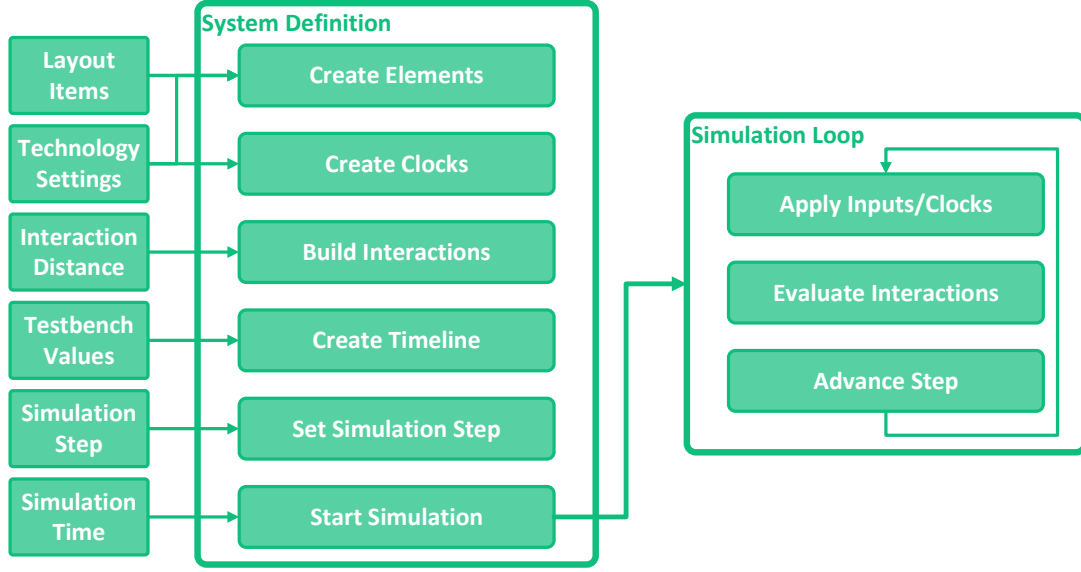


Figure 6.2: Detailed flow of FCNS. Each step is executed in sequence, the last one is the simulation loop reported on the right.

of the internal graph used to represent the circuit. Depending on the technology selected each element in the list of items read from the “.qll” is translated in the corresponding simulation element and added as a vertex to the *System* graph. In the case of input pins of the layout, the corresponding boolean flag is set to “true”. As shown in Fig. 6.6, a geometry is associated with every element. The geometry is sub-classed depending on the technology, but the base class holds the position of the element in space. This information is used to compute the distance among the elements in the subsequent step: *Build Interactions*. This function iterates on all the vertex of the *System* and defines the edges of the graph. Here the interaction distance is used to build the graph. This key parameter, that is passed to the constructor of the controller, is used to define a range of interaction for each element. Since the field coupling effects decay rapidly increasing the distance, it is useless to compute the interaction of very distant elements. However, the user could be interested in evaluating all the possible interactions in the designed circuit. Therefore the interaction radius is one of the parameters modifiable by the user. Fig. 6.3 shows a schematic representation of the described idea. In the *Build Interactions* function the distance, intended as geometrical distance among the element geometries, is compared with the interaction radius. If it is larger, the edge is not created. On the contrary, if the distance is smaller than the radius, the correct edge is instantiated and added to the graph. In the graph the presence of

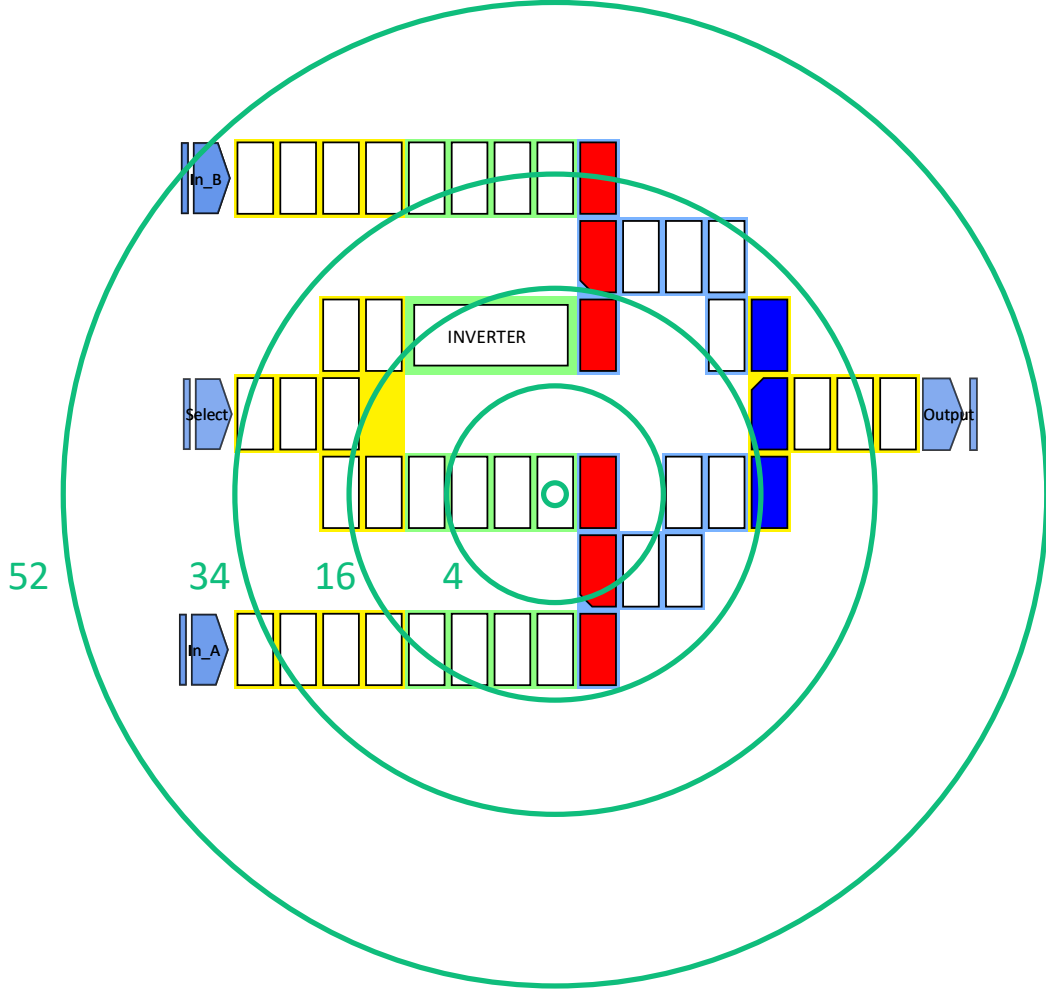


Figure 6.3: Interaction radius example. Each value results in a different amount of interacting elements, reported with the number near each circle.

an edge from node A to node B means that node A is influencing the target node B. If the node is related to an input, the incoming edges are not created during the process: input are particular nodes capable of influencing the other elements but not influenced by any other element of the circuit. Fig. 6.4 shows an example circuit and the resulting graph. The last preparatory step is the *Clocks Creation*. Depending on the technological settings the clock signals are defined. Clock sources are represented as particular elements, with different properties.

At this point the circuit data structure is complete. The simulation process could

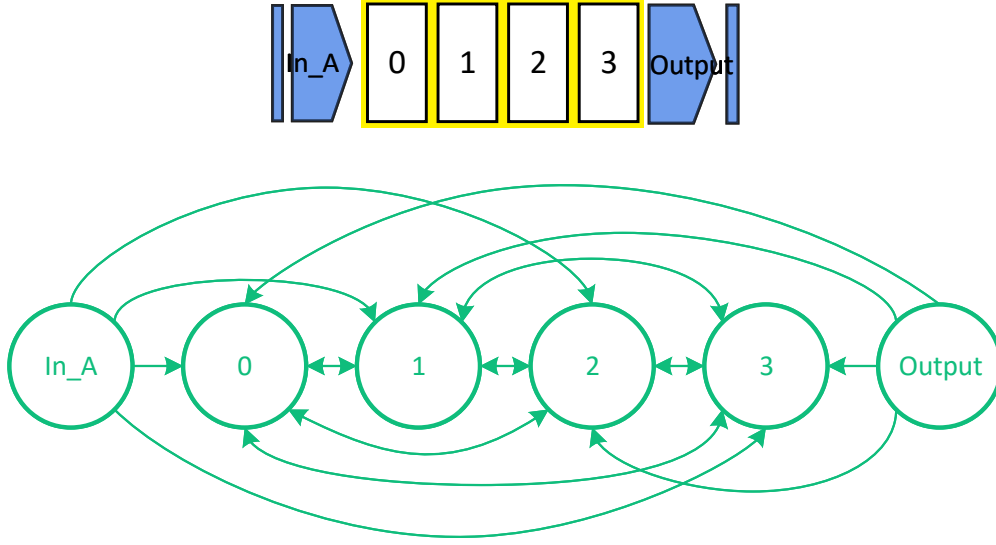


Figure 6.4: Example of circuit and the corresponding graph. The circuit is a short wire composed of four magnets and two pins. In the graph, obtained with $radius = \infty$ each node is an element and the arrows show the interactions. Input and output pins do not have incoming edges: they can not be influenced.

be repeated on the same circuit changing input stimuli and simulation step avoiding the need of loading the circuit again. Therefore, two functions are used to load the input stimuli and set the desired simulation step. The former is called *Timeline creation*. In this function, the testbench data structure is translated into an ordered list of signal-level pairs. Each pair holds the name of the signal and the same *Level* structure present in the ToPoliNano data structure. The list is ordered based on the starting time of the *Levels* to simplify the simulation process. The latter is *Set Simulation Step*. It sets the step value in the controller. Finally, the simulation process can begin by calling *Start Simulation* with the desired simulation time. When the function execution ends it is possible to load new stimuli or change the simulation step and launch a new simulation. In the following sub-section, the simulation loop will be described.

6.1.1 Simulation Loop

The simulation itself is a sequence of steps, repeated until the simulation time is reached. Fig. 6.2 shows the flow chart of the main simulation loop. Three main

macro step are present: *Manage inputs&clocks*, *Evaluate Interactions* and *Advance Step*. The first step is used to update the input and clock values depending on the *Timeline*. Since the elements in the list are time ordered, during simulation it is possible to check the head of the list: if its time is lower or equal to the current time, the value translation encoded in the *Level* is applied to the relative input signal. This process is repeated until a higher value is found as described in pseudocode 6.

Algorithm 6 Input stimuli application process.

Precondition: *timeline* is the list of pairs defining the input stimuli.

```

1: while timeline.head().level.start ≤ simulationTime do
2:   name ← timeline.head().name
3:   level ← timeline.head().level
4:   timeline.pop_front()
5:   if name.contains("clock") then
6:     SetClockValue(name, level.value)
7:   else
8:     SetInputValue(name, level.value)
9:   end if
10: end while

```

This pseudocode introduces two functions: *SetClockValue* and *SetInputValue*. The former is used to set a new value for a clock signal, while the latter is the only option to change an input element value. Both the functions iterate among the respective list of elements and, when the correct name is found, the new value is set. Values for clock and circuit elements might be represented by different quantities. Furthermore, different technologies rely on different physical quantities. In order to use the same class for the different possible values the *Value* class has been defined and it will be described in section 6.2. The second step is *Evaluate Interactions*: it is the central core of the simulator, where the visiting algorithm is implemented. Each element of the *System* graph is visited and the function *Compute New Value* is called for every node. This last function evaluates the new value for the current element considering the influence of the clock signals and all the incoming edges. The pseudocode of this step is presented in 7

From the pseudocode, it is possible to understand that the value of each element is not changed during this step. In this way the order used to visit the element is not important: the same results are obtained independently from the visiting order, they depend only on the values of the elements. The *Solver* class is used to evaluate the actual resulting values of the elements. How the value is computed depends on the interaction defined for the technology. Once all the elements are evaluated, it is possible to update the values and move to the next step. The last function is *Advance Step*. The operations performed by this function are mainly two. Firstly,

Algorithm 7 Interaction evaluation step.

Precondition: *system* is the graph of the circuit, while *clocks* is the list of all the clock signals defined for the simulation. *Solver* is the class managing the interactions.

```
1: for each node  $\in$  system do
2:   computeNewValue(node)
3: end for
4:
5:
6: function computeNewValue(node)
7:   if node.isInput() then
8:     return
9:   end if
10:  ckValue  $\leftarrow$  0
11:  for each clock  $\in$  clocks do
12:    ckValue  $\leftarrow$  ckValue + clock.getValue(node)
13:  end for
14:  neighborValue  $\leftarrow$  0
15:  for each neighbor  $\in$  graph.inEdges(node) do
16:    neighborValue  $\leftarrow$  neighborValue + neighbor.getValue(node)
17:  end for
18:  result  $\leftarrow$  Solver.evaluate(node.value, ckValue, neighborValue)
19:  node.setNewValue(result)
20: end function
```

update the value of each element with the one computed before. This is done calling a specific function, *updateValue*, on every node of the graph. This function has been developed in order to perform multiple operations: Before updating the value a pair composed by the element position in space and the old value is instantiated. Furthermore, the difference among *Value* and *newValue* is computed. Finally, the update is performed and the two results previously described are returned to the caller. In this way, the pair is passed to the *Data Manager*, while the difference could be used to define the stability of the step. The difference could be unused by some technological implementation, but with this choice the *updateValue* function is common to every element implementation. The other element introduced before, the *Data Manager*, is used to handle the output result. Its functionality will be later described. Finally, the simulation time is incremented by the step value. The described loop will be repeated until the simulation time is lower than the stop time.

In the following section the class organization will be presented.

6.2 Class organization

Fig. 6.5 shows the general view on the FCNS structure. The organization presented here refers to base classes that are the interfaces of the tool. Each technology needs to extend the functionality of the classes, as it will be later described. The main class is called *SimulationController*. This class handles the entire simulation process. Starting with loading the circuit layout and parameters, launching the actual simulation and writing the outputs. Furthermore, it manages the *System* class. It is the actual circuit under test: it contains all the elements and the clocks. *System* is a template class. This choice was mandatory to have a technology-independent core. Another class present in the *Simulation Controller* is the *Data Manager*. As expected, this class handles the data produced during simulation, and in particular, the files produced. Finally, the *Solver* class is present. This last class is the actual interaction used during simulations.

The aforementioned classes define the controller part, while the ones present in Fig. 6.6 are used to represent the data. *Element* class is the general item of the circuit. It is always associated with another class, *Geometry*, that holds the position and shape of the element. Furthermore, two objects of the *Value* class are members of each element. The same figure shows also the derived class: *Clock Element*. This class extends an element with a particular function needed to evaluate the clock value during simulation. Similarly the *Edge* class has been defined. This is used as the base class for the edges in the graph. It is defined by two components: a real number indicating the geometrical distance among the elements and a three-element vector. The elements of the vector are the x, y and z components of the distance, referred to as direction in the following descriptions.

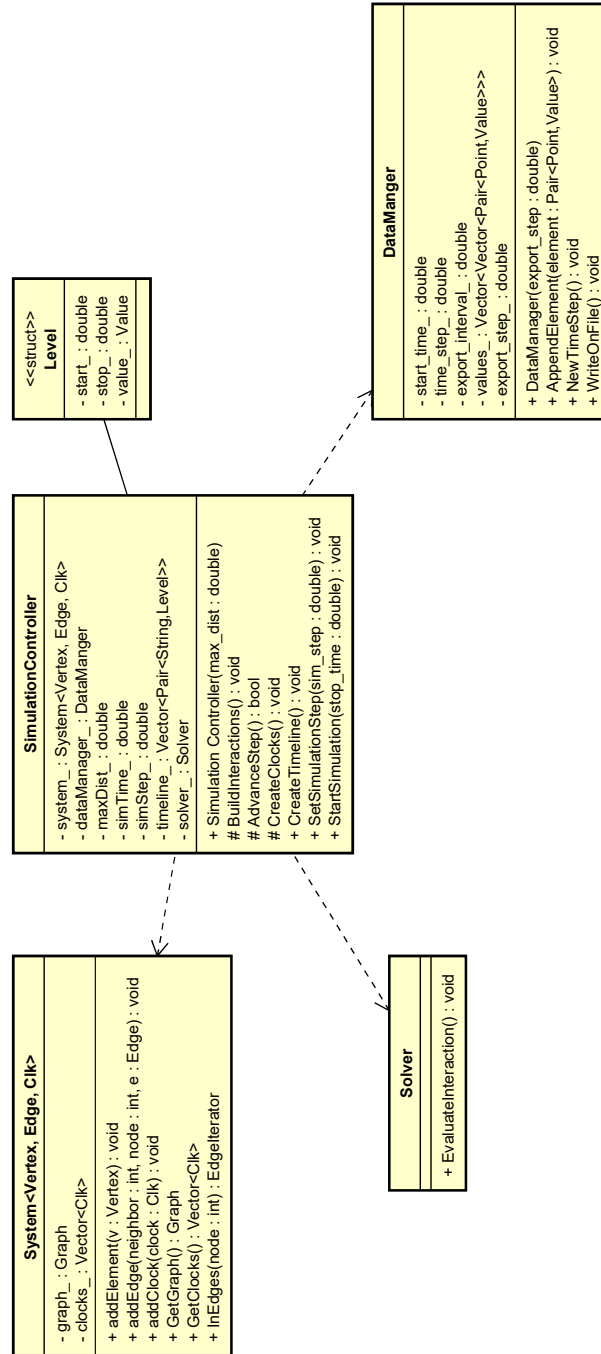


Figure 6.5: Class diagram of the interface classes. Here the class used to perform the operations are shown.

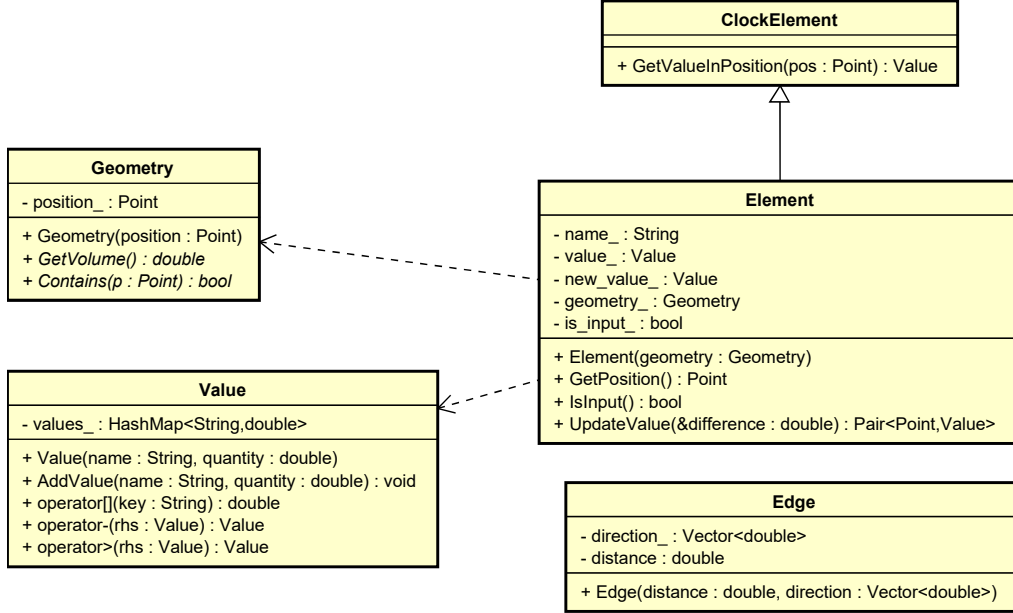


Figure 6.6: Class diagram of the data classes. These elements are used to store the data during the simulation.

6.3 Interface Classes

In this section each class presented before will be described and documented. The first class that will be presented is the *SimulationController*. Firstly the members will be described. Members stored in this class are the parameters used during the execution. They are:

Max Distance It stores the interaction radius used to build the interaction graph.

Simulation Time It is the actual time: it is increased each step and used to understand when new inputs have to be loaded.

Simulation Step This value is added to the *Simulation Time* every time a step is completed.

Timeline This vector holds an ordered sequence of new values for the inputs of the circuit.

Furthermore, three other members are available: *System*, *Data Manager* and *Solver*. These last three members are objects of the corresponding classes.

```
// Constructor.
SimulationController(double max_dist);

// Circuit preparation.
virtual void BuildInteractions() {};
virtual void CreateClocks() {};
virtual void CreateTimeline();

// Simulation loop.
virtual void StartSimulation(double stop_time, QString file_name
    = 0) {};
virtual bool advanceStep() {return true;}

// Level Structure
struct Level
{
    double start_;
    double stop_;
    Value value_;
};
```

Listing 6.1: Simulation controller methods and Level structure

Code in listing 6.1 shows the functions of the controller interface. The constructor receives as parameter the interaction distance. This parameter is the only one common to all the technological implementations. Then, three functions are present. These functions are the ones used during circuit preparation. The *Build Interaction* and *Create Clock* are just empty virtual functions. On the contrary, *Create Timeline* is defined in this interface since the *Level* struct, showed in 6.1, is based on the *Value* class. This class is a wrapper around an ordered map. In the map keys are strings, and refer to the physical quantity that is stored in that entries. Each key is associated with a real number. In this way it is possible to use the same class to store different physical properties: for example, it is possible to store a three-axis magnetization vector, a logic value or the value of four dots in a molecule. The *Value* class exposes methods to add and retrieve a quantity. Furthermore, mathematical operators have been re-implemented. In particular, during the update of the values, the difference is calculated for each key in the map, and a new *Value* object is defined, where each element is the result of the subtraction. Fig. 6.7 shows an example of *Values* and the result of a subtraction. Another important interface is the *System* class. As showed in Fig. 6.5, this has two member values: *Graph* and *ClockList*. Those two elements are the circuit data structure, used to store the elements and compute the interactions, and the list of the available clock signals respectively. The former is based on the Boost *Adjacency List*, one of the possible representations of a graph available in the selected libraries. In

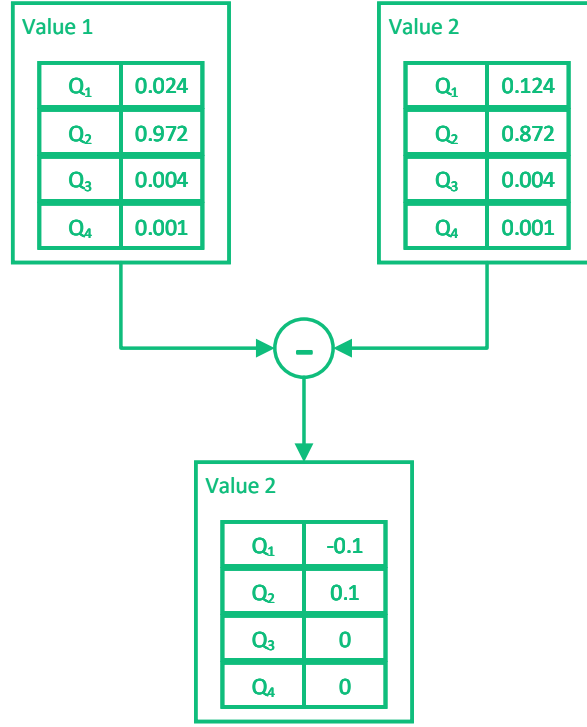


Figure 6.7: Example of two objects of class *Value*. Each one stores four different quantities, named Q₁, Q₂, Q₃, and Q₄. The difference is performed and the resulting value has the same number of quantities.

order to define an *Adjacency List*, two classes are needed. One is used as vertex elements, while the other defines the edges of the graph. Since the *Adjacency List* is used for each technology defined in the simulator, interfaces were developed for representing vertexes and edges. The base class for vertexes is the *Element*. This class, as showed in Fig. 6.6, has five different members:

Name A string used to identify the element.

Value It stores the current value of the element.

New Value It is the value that the element will assume in the next step.

Geometry It defines the physical properties of the element.

Is Input This boolean is used to distinguish normal elements from circuit inputs.

The most important is the *Geometry* class. Indeed, the constructor of an *Element* needs a geometry object: it is not possible to define a new element of the simulator

without providing a correct *Geometry*. The methods available for the *Element* class are *Getter&Setter* for the members, except for the *Update Value*. The function code is shown in listing 6.2. The difference value is passed as reference, while the pair storing the position of the element and its current value is returned to the caller. In this way, the pair is ready to be sent to the *Data Manager*, while the difference among all the quantities inside the *Value* is used by the simulation controller, as defined by the technology.

```
QPair<Geometry::Point, Value> Element::updateValue(Value&
    difference )
{
    QPair<Geometry::Point, Value> p(geometry_ ->GetPosition(),
        value_);
    if(!is_input_)
    {
        difference = new_value_ - value_;
        value_ = new_value_;
    }

    return p;
}
```

Listing 6.2: Update Value function code.

The next class that will be described is the *Geometry* one. As mentioned before, an object of this class is associated with each element. Different technological elements have different shapes and therefore the *Geometry* is sub-classed to describe the properties of the elements. Also this class is based on the Boost libraries: Cartesian points and segments from Boost *Geometry Model* are used. The base class defines that each geometry is associated with a point, saved as a member called position. This position is used to compute also the distance among two elements and it is normally the center of the shape in case of more complex geometries. Another important member is the *Volume*: the base class, intended as a single point has a volume equal to zero. An important method of the *Geometry* class is *Contains*: it returns true if the point received as parameter is “inside” the geometry shape. Its implementation depends on the different implementations. Finally, operators like subtraction and division have been re-defined to handle the three dimensions points. These operators are used to compute the direction among two elements. Direction is intended as the vector, or better its components, representing their distance, as shown in Fig. 6.8. The direction and its magnitude, referenced as distance, are the two fundamental members of the *Edge* class. These members have to be defined for each edge, but different technologies can extend the base class to add useful information. The idea behind this class is the possibility to store properties of a couple of interacting elements in the graph, in order to avoid further computations during simulation. Another class derived from *Element* is

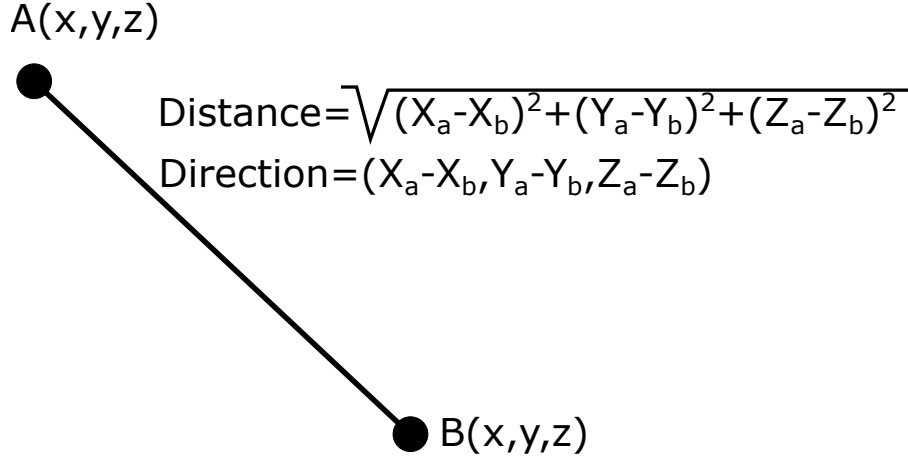


Figure 6.8: Direction and distance example with the formula used to compute them.

the *Clock Element*. This class was developed as an additional interface to represent the different clock sources defined by each technology. Indeed, each clock source is associated with a geometry, it is an *Element*. In this way, it is possible to define physical shapes and also properties for the clock sources. Indeed, the clock element could be a “logical signal” represented by a dimensionless point or a copper wire placed below the elements or a dipole placed in a specific position. The flexibility of this approach will be clear when the technological implementation is described. A specific method was defined: *Get Value In Position*: as expected, it receives a point as a parameter, and a *Value* is returned. In this way it is possible to specify also distance dependent equation for the clock value: if the clock nominal value is a magnetic field, its effect is different based on the point where the value is computed.

The *Solver* class is used to define the interaction among the elements. Each technology specifies this class adding the equations or models for the interaction. Finally, the *Data Manager* is presented. This class has been defined to handle output file management. At the moment of writing FCNS is not yet part of the ToPoliNano tool. Furthermore, the old simulator present in ToPoliNano was saving only output pin values. In FCNS all the elements’ values for each time step could be saved on disk. All the data produced by the simulation process are moved from the controller into the *Data Manager*. Here, a vector of vectors is used: each element of the vector is a time step. Each time step stores the values of each element of the circuit, counting also the clock sources of the circuit. The *Data Manager* class opens a new file each time a simulation is started. Then, each time a step is completed the controller calls the *New Time Step* function. This function is used to add an element to the vector of time steps. When the vector size multiplied by the time step is greater than the export interval, the data are written to the file.

Time step and *Export Interval* are members of the class, as well as the *File Name*. This last parameter is used to open the output file in append mode each time it is needed, to avoid to keep it opened during the entire simulation process. Through another member parameter, *Export Step*, the user can select to write on the file only a smaller portion of the computed time step. During the writing process data are extracted from the vector through a for loop: *Export Step* is used to increment the loop vector index as shown in listing 6.3. The values vector is cleared after the writing process is completed.

```
for (int i = 0; i < values_.size(); i+=export_step_)
{
    file << "Time: " << start_time_ + time_step_ * i << "\n";
    foreach (auto element, values_[i])
    {
        file << element.first << "\t";
        for(auto value : element.second.values())
        {
            file << value << " ";
        }
        file << "\n";
    }
}
```

Listing 6.3: Loop used to write simulation values on the output file.

This concludes the description of the technology independent core. In the next chapter, the iNML and MolQCA implementation of the simulator will be presented.

Chapter 7

Technology specific implementations

The simulator structure was proved with two different technologies: iNML and MolQCA. Among the presented ones, these technologies were selected because are based on different physical phenomena. Anyway, it is shown how the simulator could be easily adapted to handle those differences. Few methods have been re-implemented and they will be clearly described in the following sections. Furthermore, the model and the interaction defined for the simulation process are analyzed.

7.1 iNML

The first technology that will be discussed is iNML. Two different solver was developed for this technology: behavioral and physical. The two models are used to show the flexibility of the proposed simulator. The classes used in both approaches are very similar. For each approach the *Simulation controller*, the *Element* and *Edge* classes was developed. Furthermore, the same class was used to represent the clock sources for the physical and behavioral model. The different implementations are described in the following sub-sections, focusing on the model adopted to compute the new values and the specific methods developed. Here, a description of the *iNML clock* class is presented.

Each technology can use different models for clock sources. In iNML technology, the clock zones are rectangular shaped, with a fixed width and height equal to the circuit vertical dimension. Therefore a rectangular prism geometry was developed. Class *Geometry* was sub-classed to define the rectangular prism. The constructor of the new class, Fig. 7.1, receives as input six points. These points define the three segments (x, y, and z) of the rectangular prism as shown in Fig. 7.2. In the constructor, the points are used to compute and then store as members of the three axes. The interaction among these axes is used to define the center of the geometry.

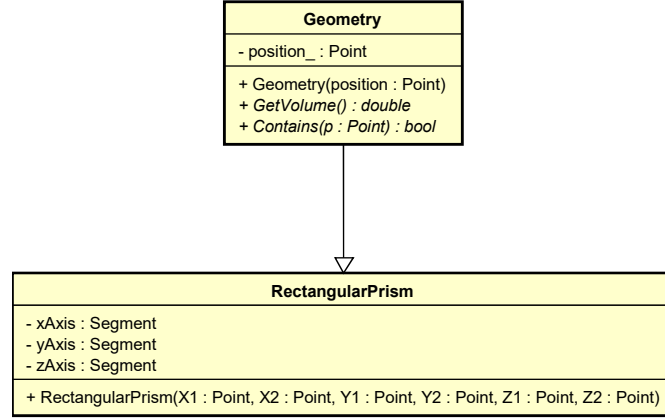


Figure 7.1: Class diagram of the rectangular prism and the base class *Geometry*.

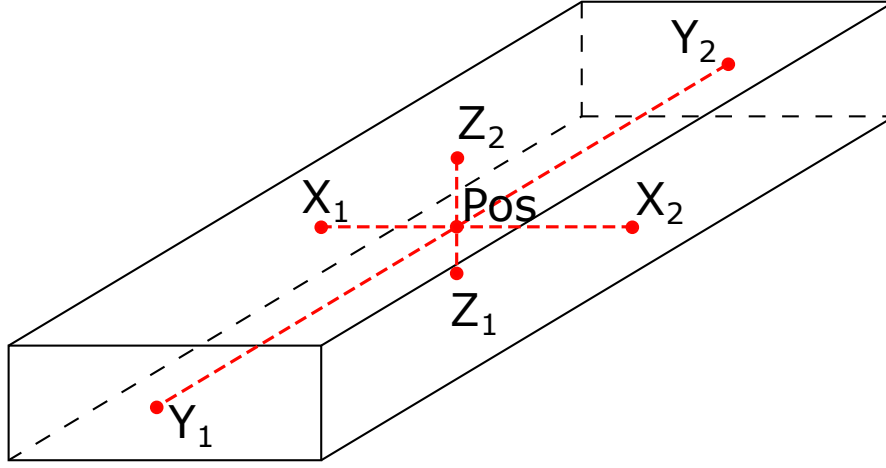


Figure 7.2: Geometrical representation of the rectangular prism. Segments and points stored in the correspondent class are highlighted.

Furthermore, the axes are used to compute the volume of the prism and also in the *contains* function. This function is used to evaluate the clock's influence on the different elements. As described before the clock source class needs a function called *GetValueInPos*. Currently, the iNML clock sources are influencing all the elements placed in the region of space delimited by the clock x and y dimension. Fig. 7.3 shows an example of elements that are influenced by the clock element. Therefore, the *contains* function returns true if the position received as parameter, projected on the x and y axis of the prism, belong to the axis itself. Inside the clock element, an if-then-else statement is used to establish the value in a given position. In fact, the clock is considered ideal. The actual value of the clock source

is returned if the element is “contained”, which means it is influenced by the specific clock. Otherwise, a default value, set to zero is returned. The iNML clock element is a magnetic vector, and the three components (x, y, and z) are present. However, the clock is active when the X component of the vector is different from 0. This is true also for the behavioral simulation: in this way the same testbench can be used for both the approaches.

In the following, the elements, edges, and controllers specific to the two approaches

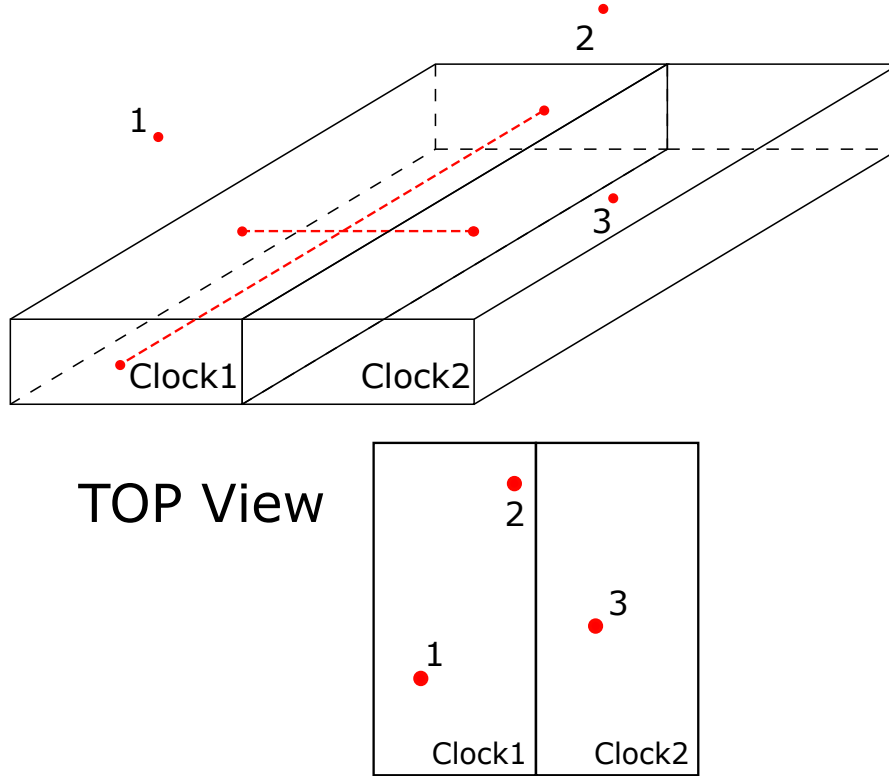


Figure 7.3: 3D and top view of an example where three elements are placed in a region with two clock zones. In this case, element 1 and element 2 are influenced by Clock 1 while element 3 is influenced by clock 2.

are described.

7.1.1 Behavioral Algorithm

The first simulation engine developed inside FCNS was the iNML behavioral algorithm. This approach is very fast, but it does not consider the physical quantities involved. Every interaction is simplified to the logical level. Nevertheless, complex circuits with a huge amount of elements are resolved in few seconds. The

simulation controller, in this case, receives a list of *QCAItem* and the technological settings. The *Max Distance* parameter is set to one for this solution. In this way, only the “touching” elements are considered during the evaluation of the interactions. A new structure is defined in the *iNML Behavioral Simulation Controller*, called *Physical Parameters*. The information included in the technological settings, that is a hash map, are extracted and used to populate the struct. They are:

X Distance Distance on the X axis among two neighboring elements in nanometers.

Y Distance Distance on the Y axis among two neighboring elements in nanometers.

Z Distance Distance on the Z axis among two neighboring elements in nanometers.

X Axis Width of the magnets.

Y Axis Height of the magnets.

Z Axis Thickness of the magnets.

Offset Quantity added to the position of the elements in order to avoid negative coordinates.

Clock zone number Clock mechanism expressed in number of different clock signals.

Magnets per clock zone Width of a clock zone in number of magnets.

Layout width Width of the layout expressed in number of elements.

Layout height Height of the layout expressed in number of elements.

The element dimensions, the spacing, and the offset are used to define the layout structure. In the behavioral algorithm, the dimensions of the elements are not used. Each magnet present in the layout is translated to an *iNML Behavioral Element*. The geometry used for behavioral elements is the basic interface already presented. Furthermore, the coordinates used are the ones read from the layout. Since coordinates are integer values, in the MagCAD layout file, the geometries of the behavioral elements will be on an integer grid. therefore, diagonal distances will be larger than the minimum interaction distance, leading to a maximum of four neighboring elements (up, down, left and right). Input pins of the layout are treated as simple magnets, and the input flag is set to true. In the case of layout elements that are associated with multiple magnets, specific functions are used.

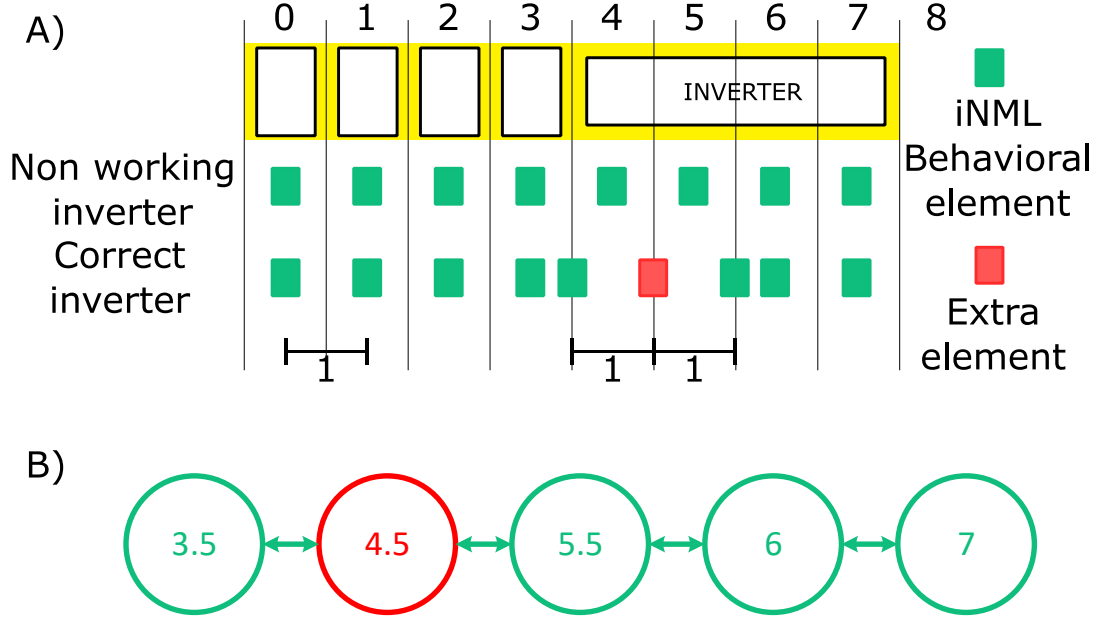


Figure 7.4: (A) wire and an inverter with the vertical grid; the vertical lines are associated with the middle among the integer coordinates. Each element is translated into the corresponding object and placed in its position. This configuration results in a nonworking inverter: an even number of elements is not inverting the information. In the other case, an extra element (the red one) is added and the position of two elements is changed. In this way, each element is influenced only by one neighbor as highlighted in the graph shown in (B).

The AND and OR gates are translated into three magnets. Furthermore, the central magnets is different from the normal ones. The *Preferred Magnetization* is set according to the logic function. The inverters are instead handled with another approach. In iNML, the inverters are achieved placing an odd number of elements in a clock zone. Inside MagCAD an inverter is a special element with an additional parameter called *length*. This parameter is used to add the element to the simulator *system*. Fig. 7.4.a shows the inverter as it is represented in MagCAD and the grid where the elements are placed, while Fig. 7.4.b shows the elements actually added to the graph with the resulting edges. The picture shows how the real coordinates available in the simulator are used to modify the structure and handle the inverter. Considering an inverter length of four magnets and placed in coordinate (4,0), the final element is in position (7,0). However, an odd number of elements has to be placed. Therefore, an additional magnet is inserted, and real coordinates are used

to avoid superposition. The first magnet is moved in coordinate (3.5,0): it will be influenced by the magnet in position (3,0) since their distance is less than one. The second is moved to position (4.5,0) in order to be logically separated from the previous one. The additional element is placed in the middle of the last two. The remaining elements are placed by a for loop until the length is reached. When all the elements are translated and the *Build Interaction* function is called. In the behavioral simulation the *Edges* are not sub classed. The only information needed in this algorithm is the distance, which has to be lower than one to create an edge and the direction. The direction is used to understand the relative position among two interacting elements: ferromagnetic or antiferromagnetic coupling is applied during interaction evaluation.

The clock elements are defined taking into account the technological properties. The number of clock sources is computed dividing the layout width by the clock zones width. The result of the division is rounded-up in order to cover all the layout. The width and height of the rectangular prism are set equal to the number of magnets per clock zone and the circuit height respectively. A for loop is used to define the clock sources. For each element, the six points needed to define the prism are evaluated. All the numbers are integer in this version of the simulator. Therefore, the offset is added to the prism border and the elements geometries will be included in the clock element. The offset value is set to 0.5. The names for the clock sources are assigned using the integer remainder of the loop index and the number of clock sources. Fig. 7.5 shows an example of clock zones. The testbench is then loaded. Inside the testbench logic values for the inputs and clocks should be defined. It is possible also to perform logic simulation providing a physical testbench. In this case, the physical quantities are translated to logic values. At this point, the simulation can start.

The simulation loop was already described in the previous chapter. In the behavioral simulation the possible values for the elements are represented in table 7.1.

Table 7.1: iNML behavioral values.

Logic value	Number value	Meaning
Reset	0	Magnet is in the unstable state
Weak 0	-1	Magnet is going to “logic 0” but it is not yet stable
Strong 0	-2	Magnet’s value is “logic 0”
Weak 1	1	Magnet is going to “logic 1” but it is not yet stable
Strong 1	2	Magnet’s value is “logic 1”

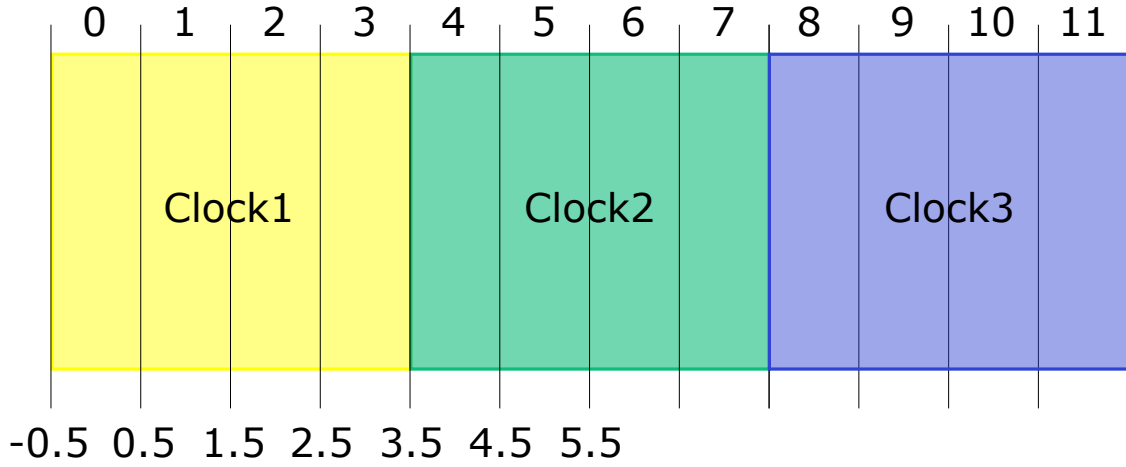


Figure 7.5: Example of three clock zones spanning over four vertical lines of the grid. Each zone is four positions wide but an offset of 0.5 is applied. Zone 1 starts from -0.5 and ends in 3.5 where the new zone begins.

This values are used in the *Compute New Value* function. In the behavioral version of this function, the clock influence is considered as dominant. The first check performed is on the clock value. For each clock source in the *System* the value in the position is retrieved: if it is different from zero the element *New Value* is set to “reset” and the function returns. If no clock source is influencing the element it is checked for stability. These properties were introduced for the behavioral elements: a magnet is considered stable if its value is “strong”. A stable magnet can not be influenced by surrounding elements: the behavioral simulation considers ideal elements. However, the “weak” version of the values is available. This idea was introduced because the visit algorithm is completely independent of the circuit layout. Therefore, information propagation direction is not known a priori. To overcome this problem, the value resulting from the interaction evaluation is always set to its “weak” version, and a specific algorithm is used to set the stable one. This algorithm is applied during the *Advance Step* function. Before entering in the details of this function, the actual new value evaluation is described. An integer value is used during interaction evaluation. It is set to the magnet preferred magnetization, and then all the incoming edges are evaluated. For each edge, the direction is considered to discriminate among ferromagnetic or antiferromagnetic coupling. Furthermore, only stable neighbors are used to compute the new value. The neighbors’ values are accumulated in the integer value defined before. In the case of antiferromagnetic coupling, the opposite of the neighbor state is used. When all the neighbors are evaluated and the resulting magnetization is different from zero, the total number of surrounding elements (stable and not stable) is used to

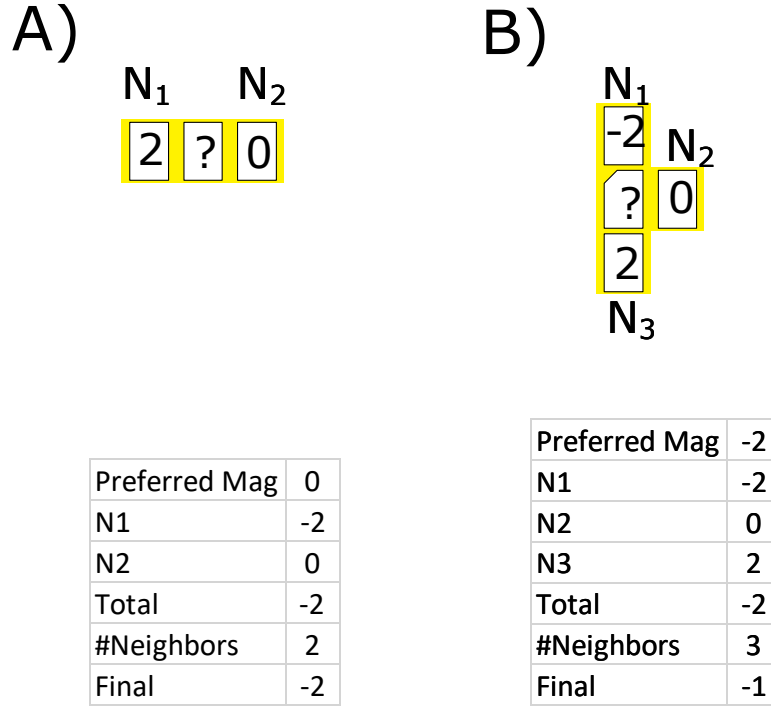


Figure 7.6: Examples of interaction evaluation in the behavioral simulation. (A) shows a simple wire. The central magnet is evaluated: the one on the left is stable, the one on the right is in reset. Preferred magnetization is zero and antiferromagnetic coupling is present, resulting in total interaction equal to -2 . Since the number of neighbors is equal to two, the new value will be directly -2 . (B) shows an OR gate. The central slanted magnet is evaluated: ferromagnetic coupling and preferred magnetization sums up to -2 . Given three neighbors the value is normalized to -1 .

distinguish two situations. If there are more than two neighbors, it is not a simple wire, the magnetization is set to the accumulated value divided by its absolute value. Otherwise, the normalization is not performed and the next value will be a stable one. Fig. 7.6 and 7.7 shows some examples of interaction evaluation. In the *Advance Step* function, the stability of each element is checked before and after performing the update of the values. If none of the elements of the circuit became stable in the step, another function is called on every element. This function, called *Promote*, is used to update “weak” values to their stable counterpart. The idea is that if no changes happen in a step, the values computed will not change in future steps. Therefore, also the “weak” values can be considered as fixed. Fig. 7.8 shows an example of a step-by-step evolution on a small portion of a circuit. The elements are starting from a reset condition and a random visiting order is considered.

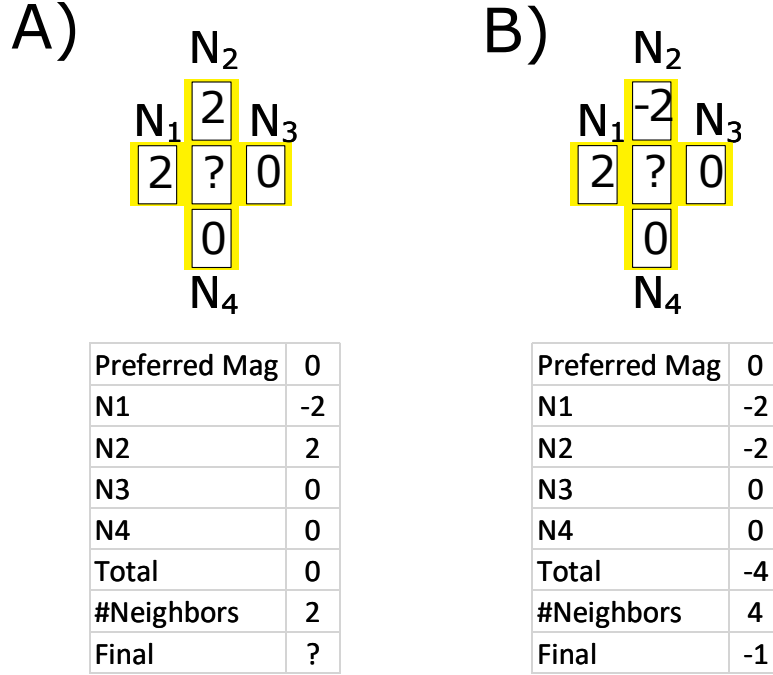


Figure 7.7: Examples of interaction evaluation in the behavioral simulation for MV layout. (A) shows an example where the sum of the neighbors is 0. Therefore the new value will be equal to the previous one. (D) is similar to the previous configuration but here the neighbors' influence sums up to -2 . In this case, this value is normalized due to the number of interacting elements.

7.1.2 Physical Algorithm

The Physical simulation engine for iNML technology was also developed. It proves the possibility to define different types of simulation engines for the same technology, using the same core. The model adopted in this simulation engine has been described in [13]. The approach is the same described for the micro magnetic simulator. Differently from those applications, here the basic elements evaluated are not cubes of a mesh grid, but the single magnets. Therefore the LLG equation is solved for each element in the layout. This simulation engine can be used to perform simulation of circuits in a reasonable time, taking into account the physical properties of the elements. The classes and methods specifically developed for the iNML physical simulator are now presented.

The simulation controller, in this case, receives exactly the same parameters as the behavioral one. Furthermore, the interaction distance can be modified by the user. The physical parameter structure is defined as the one presented in the

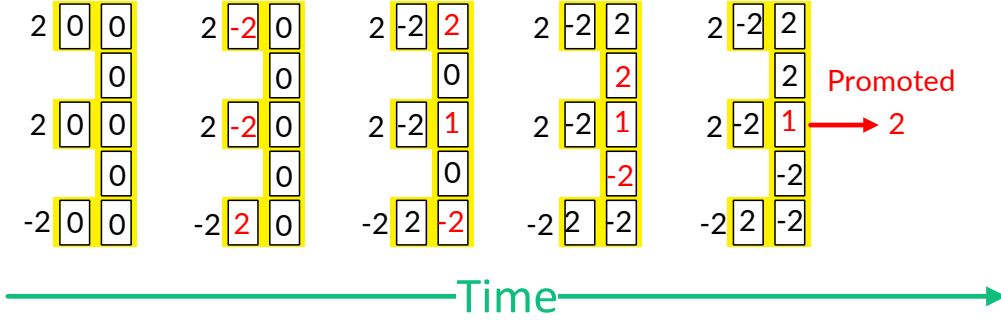


Figure 7.8: Example of evolution of an MV in iNML behavioral simulation. The elements in the circuit start from a reset condition(all zeroes). The input values are placed on the left of the elements. The first step the magnets near the inputs (red values) switch with antiferromagnetic coupling. Since the number of neighbors is 2 they assume immediately a stable value. In step number two, top right and bottom right elements reach a new stable value, while the central one will assume value equal to 1 (“weak 1”) since its number of neighbors is greater than 2. In the next step other two magnets, in red, get the new stable value. The central one is in the same condition. In the final step, the central magnet is the only one that is evaluated, and therefore it is promoted to its stable version.

previous case. Three more members are available in this version of the struct:

X Grid Size X axis length of the layout grid.

Y Grid Size Y axis length of the layout grid.

Z Grid Size Z axis length of the layout grid.

These three values are used to easily calculate the actual coordinates for the elements. Each of them is equal to the magnet size on that axes plus the inter magnet distance along the same direction. In fact, the physical elements are associated with a geometry with real coordinates. The offset is set to 200 nanometers, to avoid negative coordinates. The center of the elements are placed on the layout grid. Also, input and output pins are associated with a rectangular prism. Furthermore, the output pins are fixed in the reset state, simulating a magnet belonging to the next hypothetical clock zone. During element translation dimensions of the single element are modified based on the layout. In MagCAD it is possible to modify the physical dimensions for each magnet independently. These changes are saved in the layout file, and the simulator can read the new sizes and define the correct

geometry. When the geometry is defined the constructor of the iNML element is called. Each element is associated with its demagnetization tensor. This property, that is a matrix, depends on the geometry and is saved as a member of the element. The elements of the matrix are computed following the method described in [1]. Currently, the tool supports only rectangular shaped nanomagnets. It is possible to define new geometry and therefore calculate the demagnetization tensors for those magnets. The edges used in the physical simulation was sub classed in order to add the coupling matrix. This three by three matrix is used to compute the coupling value of the magnetic field of neighboring elements. The constructor of the iNML physical edge receives as parameters distance and direction as usual, plus the matrix relative to the interactions of the elements. The matrix is constructed using the following formula:

$$\mathbf{C}^{(i \rightarrow j)} \approx \frac{V^{(i)}}{4\pi r_{ij}^3} \begin{bmatrix} 3\hat{r}_x^2 - 1 & 3\hat{r}_x\hat{r}_y & 3\hat{r}_x\hat{r}_z \\ 3\hat{r}_y\hat{r}_x & 3\hat{r}_y^2 - 1 & 3\hat{r}_y\hat{r}_z \\ 3\hat{r}_z\hat{r}_x & 3\hat{r}_z\hat{r}_y & 3\hat{r}_z^2 - 1 \end{bmatrix}$$

The formula defines the generic matrix of the coupling effect of element i referred to element j . The scalar part is composed by the ratio between the volume of element i and distance. The elements of the matrix are computed using the unit vectors of the distance. The direction stored in the edge is divided by the distance resulting in the unit vectors used to compute the matrix elements. This matrix will be used during *Compute New Value* function. The clock sources are computed as in the behavioral case, but the physical dimensions are considered. The resulting clock elements are rectangular prism with dimensions in the order of nanometers. Finally, the default values for some quantities, like active clock field, are defined. Currently, this quantities are hard coded in the simulator executable, but the user will be able to change them from the ToPoliNano GUI. After loading the testbench the simulation can start. In order to increase the flexibility, logic values present in behavioral testbenches are translated to the corresponding physical value. It is also possible to assign directly physical values in the testbench.

The *Compute New Value* function implemented for the physical engine solves an ODE (ordinary differential equation). The equation is solved for each element and is the same presented in 5.1. In order to solve the differential equation a stepper available in the *Boost* libraries were used. It is possible to select among different steppers because they are based on common API. In order to work with a *Boost* stepper a class needs to be defined. This class is the so called right-hand side of the equal sign in the equation. The class, *RhsLLInteraction*, needs to override the *()operator*. It is called by the stepper in the *do_step* function. Furthermore, the operator has to be defined with three parameters: the state of the system, its derivative and the time. State type can be defined depending on the target phenomenon while time is a double. Since the simulator solves the equation for

the magnetization vector, the state type is a three element vector, representing the three components of the magnetization. The differential equation can be also written as:

$$\frac{\partial \mathbf{M}}{\partial t} = -\gamma (\mathbf{M} \times \mathbf{H}_{eff}) - \frac{\alpha\gamma}{M_s} (\mathbf{M} \times \mathbf{M} \times \mathbf{H}_{eff})$$

Inside the *RhsLLInteraction* class the values of γ , α and M_s are saved as member variables. The *()operator* code is shown in listing 7.1.

```
state_type first_term = cross_prod<state_type>(M_, Heff_);
state_type second_term = cross_prod<state_type>(M_,
    first_term);

dMdt_[0] = -gamma_*first_term[0] -alfa_*gamma_/Ms_*
    second_term[0];
dMdt_[1] = -gamma_*first_term[1] -alfa_*gamma_/Ms_*
    second_term[1];
dMdt_[2] = -gamma_*first_term[2] -alfa_*gamma_/Ms_*
    second_term[2];
```

Listing 7.1: Code of the *()operator* in the RHS class.

In the code the *ccross_prod* function is called to compute the cross product showed in the formula. Then, the derivative term is composed. The H_{eff} is received in the constructor of the *RhsLLInteraction*. Actually, the *Compute New Value* function is used to compute the H_{eff} , effective magnetic field, of the element. This term embeds all the influencing fields, both from neighboring elements and clock sources. The H_{eff} is computed according to the following equation:

$$\mathbf{H}_{eff}^{(i)} = -\mathbf{N}^{(i)}\mathbf{M}^{(i)} + \sum_{j=neighbors} \mathbf{C}^{(i \rightarrow j)}\mathbf{M}^{(j)} + \mathbf{H}_{ext}^{(i)}$$

In this formula $\mathbf{N}^{(i)}$ is the demagnetization factor, obtained by multiplying the demagnetization tensor by the value of the element. $\mathbf{H}_{ext}^{(i)}$ is the external field influencing the magnet. In the simulator only the clock field is considered as external influence. The last part of the effective magnetic field represents the interaction of the other elements: it is obtained multiplying their value by the coupling matrix. After the evaluation of H_{eff} , the solver is called. The solver class provides the stepper selected, and the step is executed. Each time the *do_step* is called, a new object of type *RhsLLInteraction* is instantiated with the current values. Furthermore, the simulation time and simulation step are needed by the stepper.

The operation here described are repeated for all the elements of the *System* at every step of the iteration. Differently from the behavioral simulation engine, the stepper needs to perform a huge amount of calculation, resulting in longer simulation time. To overcome this situation the simulation engine was modified to implement multithreading and speed up the simulation process. Taking advantage from the Qt libraries the *QtConcurrent* class was adopted, in particular the *map*

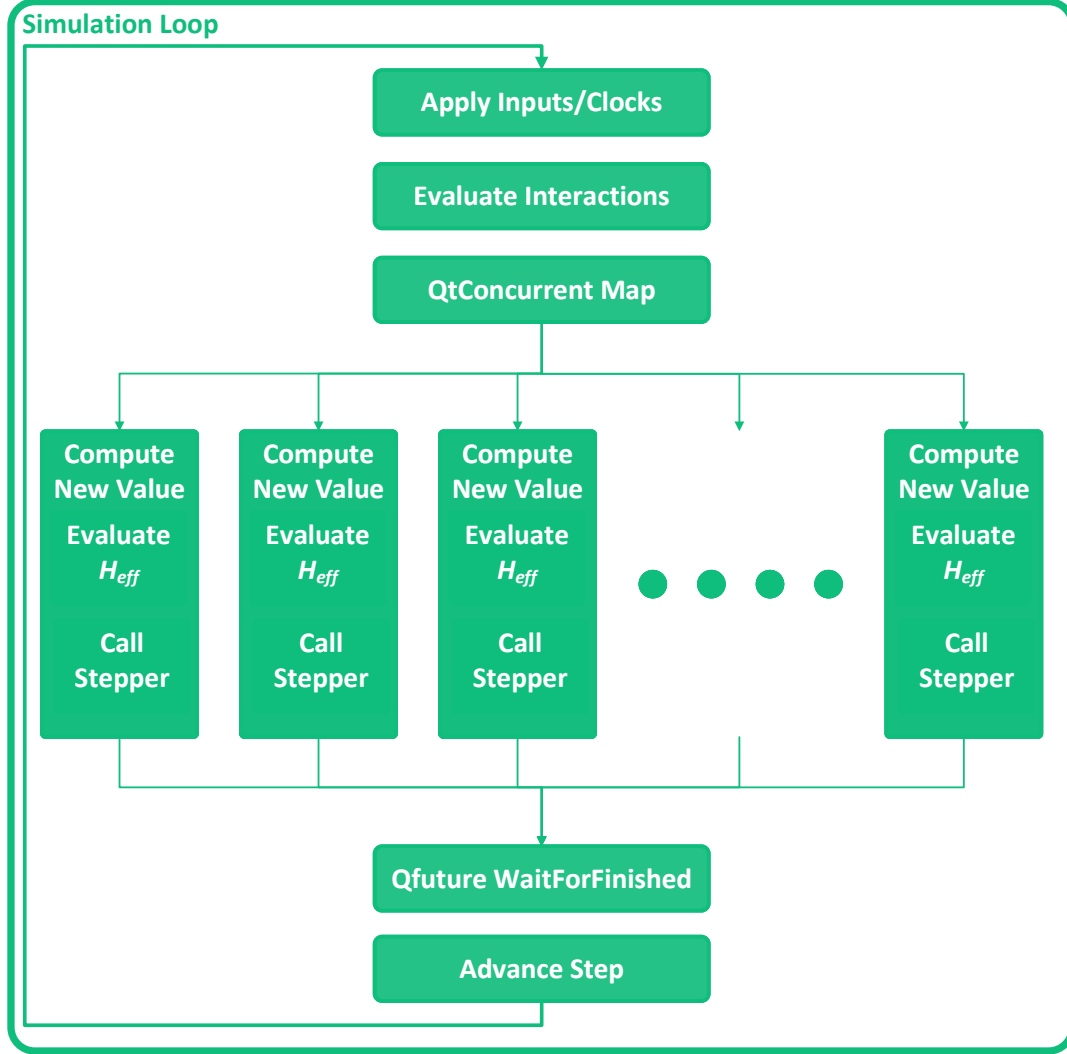


Figure 7.9: Schematic view of the multithreading approach present in the simulator. The `QtConcurrent::Map` functions launches a thread, depending on the number available on the system, and assign to each thread the *Compute New Value* operation of a single element. Each thread evaluates the H_{eff} and calls the stepper for the ODE resolution. The `QFuture` waits that all the threads finish their operation and then the simulator moves to the next step.

method. This method is specifically designed to execute in parallel the same computation on an iterable container. Furthermore, the *QFuture*, another class of the `Qt` libraries, was used to correctly implement the parallel computation. In fact, the

QtConcurrent Map will launch new threads for each element, but the main thread will continue its execution. This would lead to a problem in the synchronization and the step update. The *Qt* framework provides a way to stop the main thread and wait that all the elements of the iterable list are processed. In order to use the *QtConcurrent Map* the evaluation structure needed some modification. The *Map* function needs to specify the operation to be performed on every element. This operation can be a static method that receives a reference to the object iterated. Otherwise, the mapping process works with a member function of the class stored in the sequence iterated. The previous approaches were not compatible with the structure already developed of the simulator. Therefore, another solution was adopted: using Function Objects. This solution is very flexible. A new struct was defined, and the *()operator* implemented. The struct holds a reference to the *System* and all the parameters used during the value computation. Furthermore, the *()operator* is the same code defined for the *Compute New Value* function. It computes the interactions of all the neighboring elements and the clock sources and finally calls the stepper for the ODE. *QtConcurrent* manages autonomously the thread pool and the execution, but it is possible to limit the number of thread used simultaneously to avoid problems during the execution. The multithreading approach led to a great improvement of performances that will be discussed in the next chapter. Fig. 7.9 shows a schematic view of the multithread approach.

The *Advance Step* function performs the update for the values of each element. Differently from the behavioral case, there is no need for checking the stability: the values calculated solving the differential equations are updated and a new step is started.

7.2 MolQCA

Another technology is supported by FCNS. An engine for MolQCA was developed. This interesting technology is also useful to prove the possibility to handle completely different physical phenomena in the tool. Furthermore, the simulation engine for molecules is not based on the time evolution of the molecule but on equilibrium state simulations. In fact, a general model for the molecule dynamic is not yet available. Therefore, a physical simulation based on the time-dependent evolution was not possible. The circuit state evolution is evaluated until the equilibrium is reached, then it is possible to change the input values and search the new equilibrium point. The *Element*, *Edge* and *Simulation Controller* classes were sub-classed for the molecular engine. Furthermore, a new class, handling the molecular characteristics, was defined: *Molecular Parameters*. This class stores all the physical parameters of a molecule available for the simulation. Furthermore, the trans-characteristics of the molecule are stored in this class. Fig. 7.10 shows the

general structure adopted to store the different molecule trans-characteristics. The molecule behavior is described using a set of different curves. Each curve describes the charge value of an aggregate charge with a fixed clock value and dependent on the input value: each aggregate charge is associated with the same number of curves. In the figure, four curves are available for each charge. These curves are quantized and stored inside the *Molecule Parameter* as objects of a specific developed class, called *MolecularTranscharacteristic*. The charge values are stored in a vector with size equal to the number of charges. The bottom-left part of Fig. 7.10 shows an example of three vectors associated with different values of V_{in} . In particular, the first vector is referred to as V_{Min} and the last one refers to V_{Max} . The resulting vector of vectors is then associated with the clock value used to extract the curve. The extreme voltages, the number of charges and the number of elements in each vector are saved inside the *MolecularTranscharacteristic*. Furthermore, the voltage distance among two adjacent values in the vectors is saved in the *step* variable. It is computed with the following formula:

$$\text{step} = \frac{V_{max} - V_{min}}{\#values - 1}$$

where *#values* is the size of each vector. The *step* is used to compute the interpolation when the charges are requested. An instance of this class is used for each defined clock value. The final data structure, top-right of the image, is an hash-map. This hash map is used to store the trans-characteristic objects associated with the clock values. The keys of the map are the possible clock values and a null charge is retrieved if the clock value is not defined. Another information stored in the molecular parameter class is the relative position of each charge with respect to the center of the molecule.

The clock elements used for MolQCA technology are completely different from the iNML ones. A rectangular-shaped clock zone is not sufficient to define correct circuits and a new approach was adopted. The new clock item, called *Amorphous Clock Element*, is based on the principle of adding influenced position to a set stored inside the clock element. In this way, the *Get Clock In Position* will check if the given position is present in the set, and use this information to discriminate among influenced and not influenced elements.

The simulation controller starts reading the molecular parameters. An ad-hoc functions was developed: *Read Molecules*. It receives a list of strings, where each string is the name of a molecule. All the files which name starts with the molecule name are read. The files are searched in a user-defined folder and each file is a trans-characteristic associated with a clock value. A simple reader was developed. It parses the source file and reads from the header the parameters like V_{Max} and V_{Min} . Furthermore, the charge values are stored in the same structure used inside

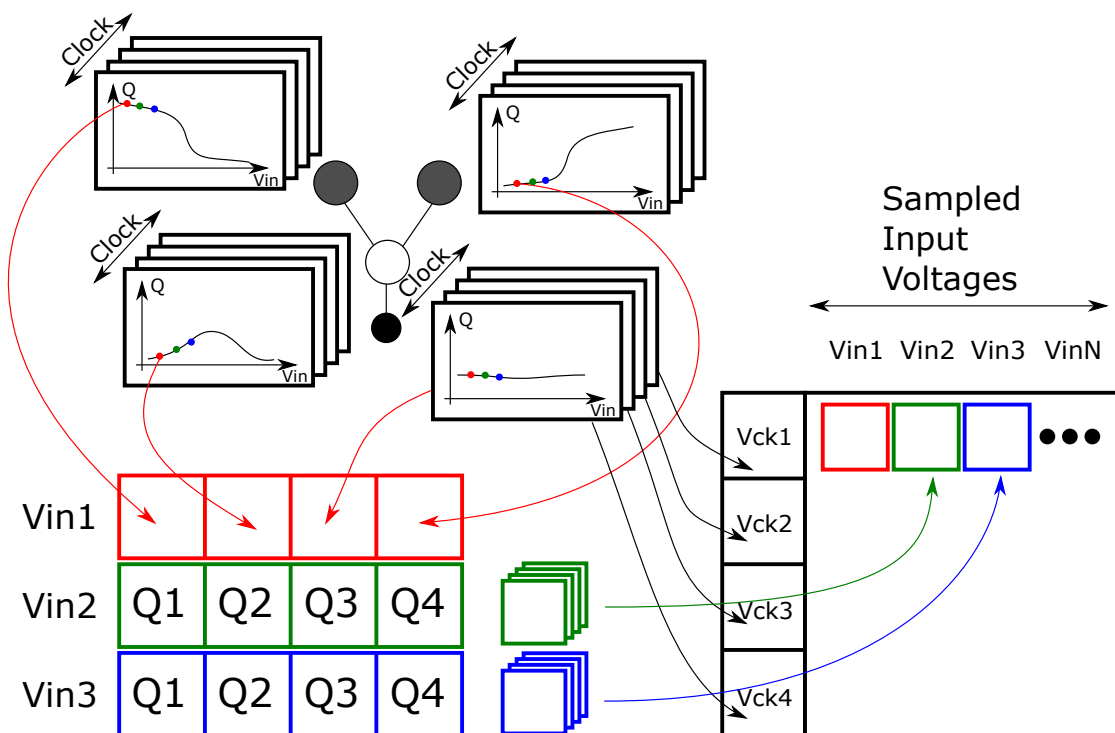


Figure 7.10: Representation of the data structure used to store the molecules trans-characteristics. In this example, the molecule has four dots. For each dot a set of curves, one for each clock value, is available. The curves give the charge over the input voltage. In order to store the curves, the same input voltage is used to sample all the characteristics and define a vector: the size is equal to the number of charges and each element is a charge value. All the vectors resulting from this sampling are named with the corresponding input voltage and stored in an hash-map. The keys of the map are the clock voltages. Each key is associated with a vector where all the input voltages grouped charges are stored.

the *MolecularTranscharacteristic*. When the file is completely parsed the new entry is added to the map of trans-characteristics. All the available molecules are stored in another hash-map, using their name as key of the map.

After loading the molecules, the technological parameters are read from the settings. The MolQCA settings are composed of the following parameters:

Inter molecular distance MagCAD grid dimension in pm.

Inter cell distance Double of the previous quantity since MagCAD cells contain two molecules.

Clock zone number Number of different clock signals.

Layout width Width of the layout expressed in number of cells.

Layout height Height of the layout expressed in number of cells.

Differently from the other technologies, in MolQCA the clock elements are created before the elements. This is needed by the fact that when an element is created, it has to be assigned to a clock zone. To avoid further loops on the elements, clock sources generation was moved before element definition. The number of clock sources is used to instantiate the right number of *Amorphous Clock Element*. Each of them is placed in a non-physical position and saved in the clock list of the *System*. The *QCAItem List* is then processed and every item is translated in the corresponding molecules. As stated before, each item corresponds to two different molecules, excepting the pins that are treated as single molecules. A for loop is used to translate the molecules. Each *QCAItem* holds different properties for the two molecules and the index of the loop is used to access them. If the single molecule is not disabled, its shift values, one for each axis, are read from the item properties and used to define the actual position of the molecule. This position will be considered as the center of the molecule. The *Geometry* class was subclassed in a new geometry, called *MultiPoint*. As per the clock elements, molecules are not defined with a geometrical shape, instead, a single point for each aggregate charge is defined. The new class holds a vector of points where the aggregate charges position is stored. The constructor receives only the size of the vector and the central position (a center is needed it being a *Geometry*). The *Molecular Element* is defined and the first method called after the constructor is *Define Charge Position*. In this method, each point of the geometry is assigned with its actual position in space. The relative positions for the aggregate charges are retrieved from the *Molecular Parameters* class using the molecule type, and then the actual positions are calculated. Before adding the element to the *System* a final operation is performed. In MagCAD it is possible to set a rotation of the molecule on the “Z” axis. In the `Boost::Geometry` libraries the 3D rotation is not defined. Therefore a new function was defined to handle molecule rotation. Using the principle for composition of geometrical transformations and the functions available in the libraries the functions 8 were developed. The *Rotation* and *Translation* classes are available in the *Boost* libraries. The former is defined with an angle, while the latter with x and y displacement values. In order to rotate the molecule around its center a composition of translation and rotation is needed. The rotation is defined around the origin, therefore the molecule has to be translated in the origin, rotated, and finally moved back to its original position. The transformation composed in this way is then applied to each aggregate charge in the molecule. Since the transformation is 2D, the points are projected on the x-y plane. Then the transformation is applied and finally the z coordinate is restored. The new point is replaced in the

Algorithm 8 Molecule rotation algorithm.

Precondition: *angle* is the rotation angle in degree, *Geometry* is the molecule *MultiPoint* object. *Translation* and *Rotation* are classes of the Boost libraries.

```
1: Translation1  $\leftarrow$  ( $-Geometry.x()$ ,  $-Geometry.y()$ )
2: Rotation  $\leftarrow$  angle
3: Translation2  $\leftarrow$  (Geometry.x(), Geometry.y())
4: FinalTransformation  $\leftarrow$  Translation2 * Rotation * Translation1
5: for each Point  $\in$  Geometry do
6:   point2d  $\leftarrow$  (Point.x(), Point.y())  $\triangleright$  Project on x-y plane.
7:   FinalTransformation(point2d)
8:   Point  $\leftarrow$  (point2d.x(), point2d.y(), Point.z())  $\triangleright$  Restore z coordinate.
9:   Geometry.replacePoint(Point)
10: end for
```

geometry to perform the actual rotation.

The operation above described are performed also on the input and output pins, even if only one molecule is defined and the *IsInput* flag is set. Finally, the *Build Interactions* function is called and the edges of the graph are defined. Also for the molecular simulation engine the interaction distance is defined and used to set an influence radius around the molecule. The edges defined for this technology are defined with a vector of vectors of double: each element stores the distance among two aggregate charges: element 1,2 in edge i, j refers to distance from charge 1 of molecule i to charge 2 in molecule j . Fig. 4.13 shows how distances are stored in the edge class. This solution could impact the memory usage of the system, but it was selected in order to improve the simulator performance. The testbench is then loaded from the disk. Input and clock values for MolQCA are voltages. The voltages are used to read the trans-characteristic and derive the actual values. Once the testbench is loaded the simulation loop can be started.

The simulation process is different from iNML technology. In MolQCA, the simulation step can be changed when the equilibrium is reached. Therefore, another loop is needed to perform the simulation. The inner loop will run until the equilibrium is reached. Fig. 7.11 shows the flow chart of the simulation. The *Evaluate Interaction* function visit all the elements of the layout and calls *Compute New Value* on each of them. The new value of the element is computed using the following interaction model.

$$\mathbf{NewValue} = \mathbf{State} + \xi * (\mathbf{CurrentValue} - \mathbf{State})$$

Where ξ is the damping factor used to lower the effective variation of the molecule state. It is used to improve the convergence of the method even if it increases the number of steps to reach an equilibrium condition. **CurrentValue** and **NextValue** are the values of the aggregate charges before and after the interaction evaluation respectively. Finally, **State** is the value obtained from the trans-characteristic. This value is the actual result of the interaction evaluation. In fact, the neighboring elements are used to define the voltage applied to the element. The voltage and the clock value are used to get the data from the trans-characteristic. The two contributes are evaluated as follows. The *Get Value In Position* is called for each clock source and the value accumulated into a variable. Currently, only one clock element is influencing each molecule, but this can be modified in future versions of the simulator. The neighbors instead are used to compute the influencing voltage. This voltage is computed as the difference among the electric potential present on the logic dots of the molecule. In case of bis-ferrocene molecule, only “Dot1” and “Dot2” are considered as showed in Fig. 7.12. For each neighbor the resulting potential is evaluated and then accumulated into a variable using the formula of

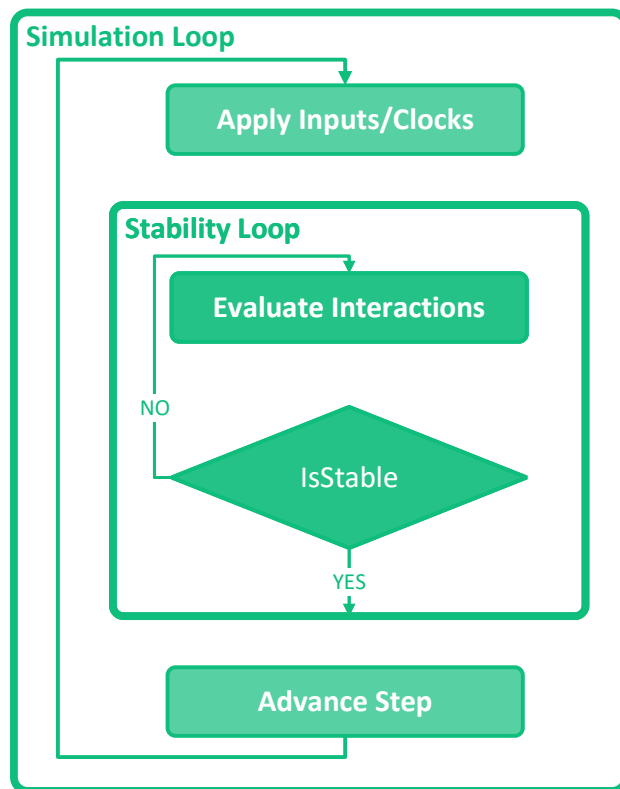


Figure 7.11: Flow chart of the molecular simulation engine. The main simulation loop is composed of a nested loop that is used to evaluate the stability. A new set of inputs and clock values is loaded only after reaching the stability..

the electric potential due to a point charge:

$$V = \frac{e}{4\pi\epsilon_0} * \frac{\mathbf{Charge}}{d}$$

The e in the formula is the electron charge and it is needed since in the trans-characteristics the charges are stored as normalized charges. The remaining terms are the vacuum permittivity, ϵ_0 , the value of the considered aggregate charge and the distance retrieved from the edge data structure. When the two actual potentials have been evaluated the new aggregate charges values are read from the trans-characteristic. This process accesses the right *Molecule Parameter* object using the molecule name. The clock value and the input voltage, obtained with $V_1 - V_2$, are used to get the charges. The clock access the map of characteristics and the correct one is returned. The *Get Charges* performs the interpolation of the data in order to compute the actual values of the charges. The function pseudocode

Algorithm 9 Trans-characteristic interpolation algorithm.

Precondition: *inputVoltage* is the voltage considered during interpolation, V_{min} and V_{max} are the limit of the trans-characteristic. *step* is the voltage distance among two values in the *Charges* vector.

```
1: if inputVoltage  $\leq V_{min}$  then
2:   return Charges[0]
3: end if
4: if inputVoltage  $\geq V_{max}$  then
5:   return Charges[last]
6: end if
7: index  $\leftarrow \lfloor (inputVoltage - V_{min}) / step \rfloor$ 
8: correction  $\leftarrow |(inputVoltage - V_{min}) / step - index|$ 
9: return Charges[index] + correction * Charges[index + 1]
```

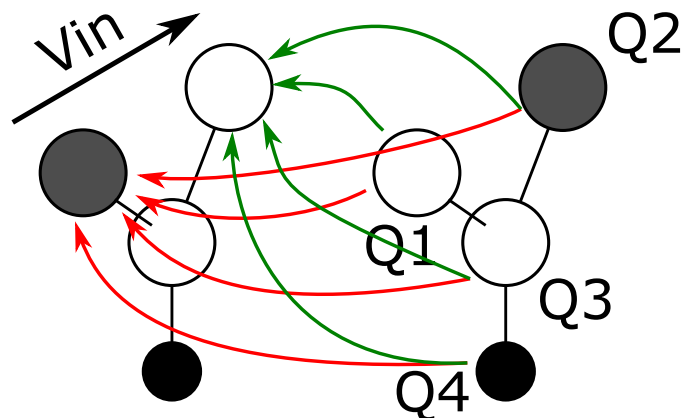


Figure 7.12: Example of interaction of two molecules. The one on the left is evaluated considering the effects of the one on the right. Each dot of the influencing molecule generates a potential on each of the logic dots of the influenced one. The difference among these potentials is used as the input voltage of the molecule.

is shown in 9. These operations are repeated for each element in each step.

In order to determine the equilibrium condition, the *Advance Step* function was modified. In MolQCA the difference among previous and next values is used to determine stability. Each element is compared with the *Stability Threshold*. It is enough that only one difference is greater than the threshold and the step is not considered as stable. The input and clock values are not changed until a stable step is detected. Finally, it is possible to select two modes for output data availability. If the *Complete* flag is set to true, all the steps are sent to the *Data Manger*. Only the stable ones are saved to the file otherwise.

This description concludes the analyses on the technological implementations. In the following chapter the performance of the simulator and the results are presented.

Chapter 8

Results and Performance Analysis

The simulator was tested with different layouts of the supported technologies. Furthermore, it was compared with the ToPoliNano current simulation engine for what concerns the iNML behavioral algorithm. **Mumax3** was used as a reference for the physical version of the iNML, while SCERPA for the MolQCA. The results produced by the simulator are text files. In order to analyze the results, different Python scripts were developed. One script is used to plot the simulation results. It is capable of reading the file generated by the different engines and plot the

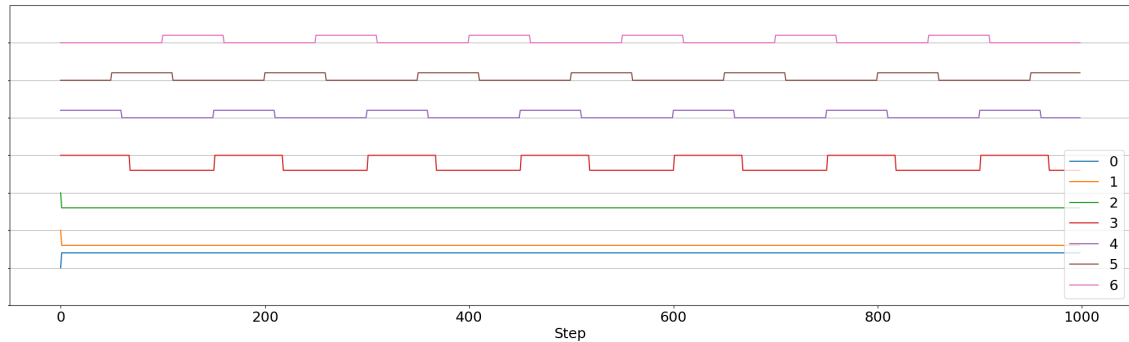


Figure 8.1: Waveforms obtained with the behavioral engine. Each waveform is the logic value of an element in the different time steps.

corresponding significant charts. For example, the iNML behavioral elements are placed in the same chart. Each element is assigned to a color and has a different Y (computed adding an offset proportional to the signal number), while the X coordinate is the simulation time expressed in the number of steps. A legend helps the inspection of the results that are square waves in this case, Fig. 8.1. In the script, it is possible to specify a reduced number of signals: the selected ones will be plotted while the others are omitted from the graph. By doing so it is easier to

verify the behavior of the layout visualizing only input and outputs. In the Physical simulation three charts are used to plot the three components (X, Y, and Z) of the magnetization vector of each element. The resulting module of the vector is equal to the saturation magnetization. Also in this case, each magnet is associated with a color and the legend is present as shown in Fig. 8.2. Finally, the MolQCA is divided

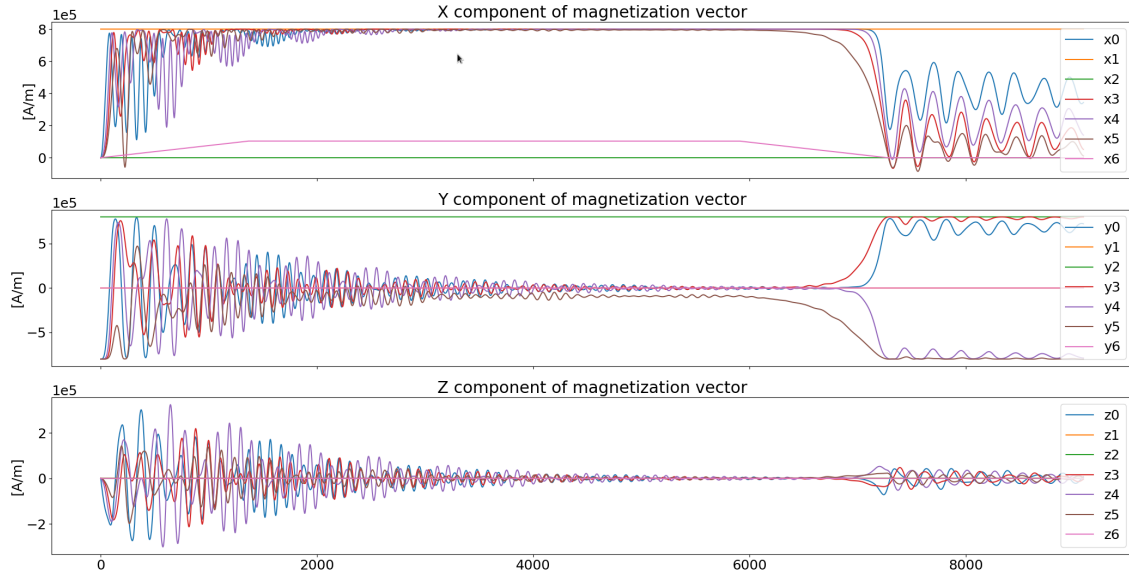


Figure 8.2: Each graph represents one of the components of the magnetization vector. Each magnet is associated with the same color in all the graphs. The legend is used to identify the magnets. The number of steps is present on the X-axis.

into one chart for each aggregate charge (Fig. 8.3). The color scheme and the legend are present to simplify the inspection. Also, the clock sources are added to the plot as additional elements, in the fourth charge plot. This representation is not the final one: a data visualizer is planned as future work. Another script was developed to perform the characterization of some structure in the physical iNML engine. This script defines the circuit names and other parameters and executes several simulations in series. Finally, a verification script is used to test the simulation outputs against expected values for the layout. After defining the truth table for all the magnets the last step is compared. It is also possible to define a threshold and check if the obtained magnetization vector is compliant with the predicted value. The threshold will be clarified in the next section. Furthermore, the simulation time is extracted and used to evaluate the performance of the different simulation engines. The results presented in this chapter are obtained using an Intel Core-i7 7700, equipped with 16GB of RAM running Centos operating system. The

physical simulations for iNML technology were performed on an NVidia TITAN V GPU consisting of nearly four thousand of CUDA cores and equipped with 12 GB of RAM.

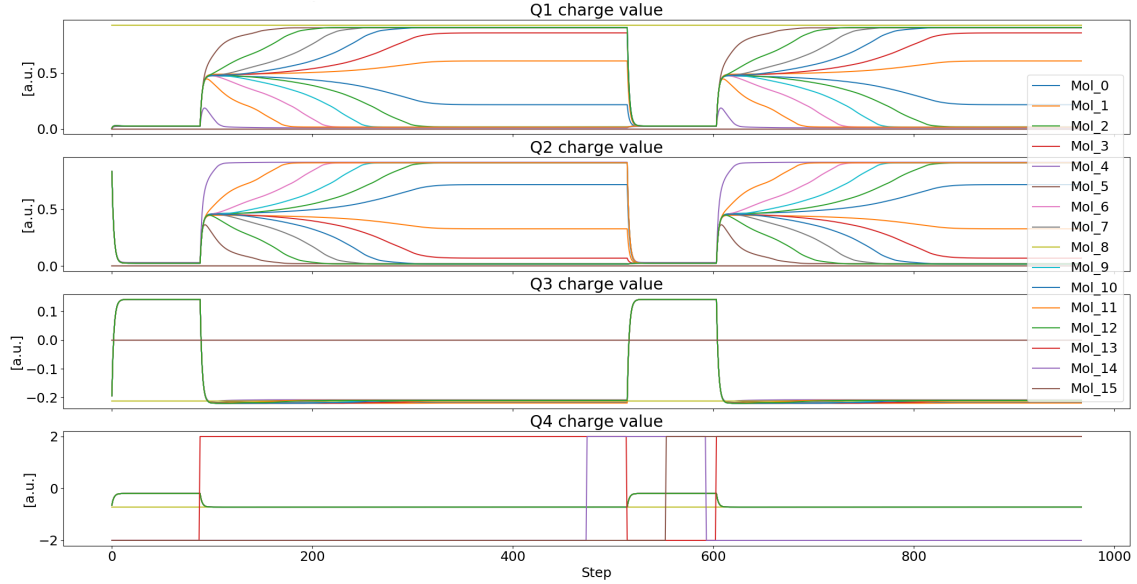


Figure 8.3: Each graph is associated with the charge of an aggregate charge. The charges are expressed in atomic units. on the fourth graph also the clock sources, square waves ranging from -2 to 2 volts, are showed.

8.1 iNML Behavioral

In this section, some layouts simulated with the behavioral model are reported. The first layout presented is a horizontal wire composed of four magnets, Fig. 8.4. The input is set initially to “0” and then changed to “1” after 20 ns. A three-phase clock mechanism is used: the clock phases must be partially overlapped to ensure information propagation. Therefore, each clock will be active for 6 ns and inactive for 9 ns. A step equal to $1e - 10$ is adopted in this case. The X-axis of the plot represents the number of steps: step number 200 corresponds to 20 ns. In Fig. 8.5 the resulting waveforms are presented. When the “Clk1” is active all the magnets are in reset, therefore their value is “0”. When the clock source becomes inactive, the magnets will align themselves in an antiparallel way. The first magnet (labeled with zero) will have a value opposite with respect to the input as the second will be the opposite of the first and so on. After 20 ns the input value is flipped, and the successive evaluation of the magnets will be the opposite of the

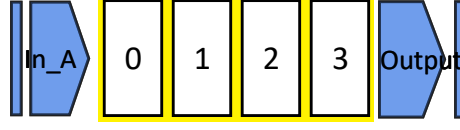


Figure 8.4: Horizontal wire composed of four magnets. Each element is associated with an index that will be used in the simulation graph.

previous one. The simulation of the discussed configuration for a total time of 50 ns results in a simulation time of 33 ms. Fig. 8.6 shows an *AND* and *OR* layout.

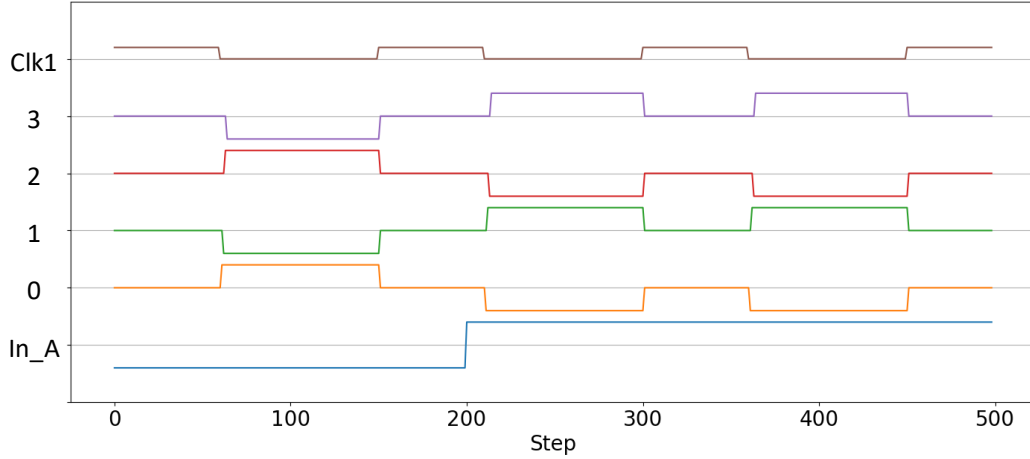


Figure 8.5: Waveforms obtained simulating the horizontal wire.

In this figure, the elements are numbered as they will be plotted in the simulation results. The input value are set to “11” for the first evaluation of the *AND* gate. Then one of the inputs is flipped to “0” and the second clock cycle is evaluated. Finally, the opposite values are applied in the third clock cycle. The *OR* case is similar, however, both the inputs are set to “0” at the beginning, successively the two combinations “10” and “01” are applied. Fig. 8.7 shows the waveform resulting from the simulation. The circles highlight the first evaluation of the central magnet, where the preferred magnetization is normalized and the magnet is still considered unstable. In fact, the value was wrong and the correct evaluation is then performed in the next step. The central magnet does not have the expected magnetization.

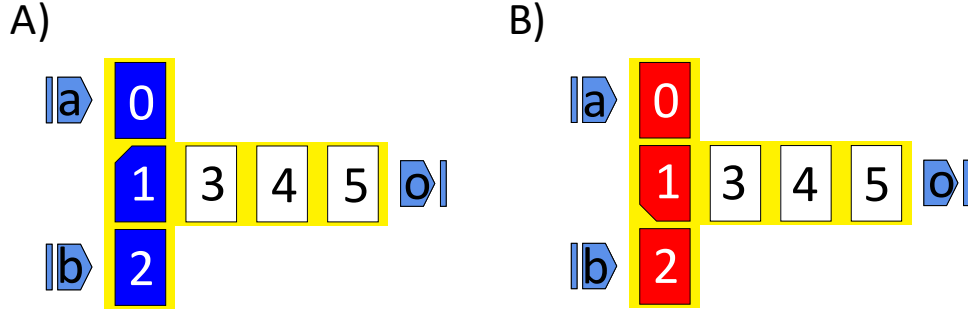


Figure 8.6: Layouts of the iNML logic gates obtained with the slanted edge magnets. (A) OR gate. (B) AND layout.

When both the inputs are equal to “1” it assumes the opposite value. However, this is correct since it is the first magnet of the clock zone and its value will be inverted by the other elements. It is clear that the function of the circuit is correct: element “5” is equal to “1” in the first clock zone and then goes to “0” in the others. A similar behavior is visible in Fig. 8.8. In fact, the circle highlights the

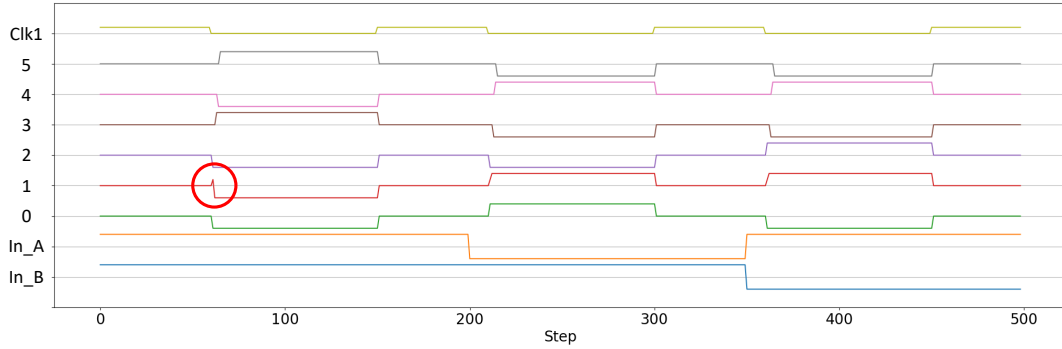


Figure 8.7: Waveforms obtained with the behavioral simulation of the AND gate.

same thing happening in the *AND* gate. Also in this case, the function of the logic gate is verified. The simulation time for both the circuit is 35 ms. An MV is then used as a benchmark. The layout is depicted in Fig. 8.9. The magnets are numbered to simplify the result inspection. The clock signal applied is the same

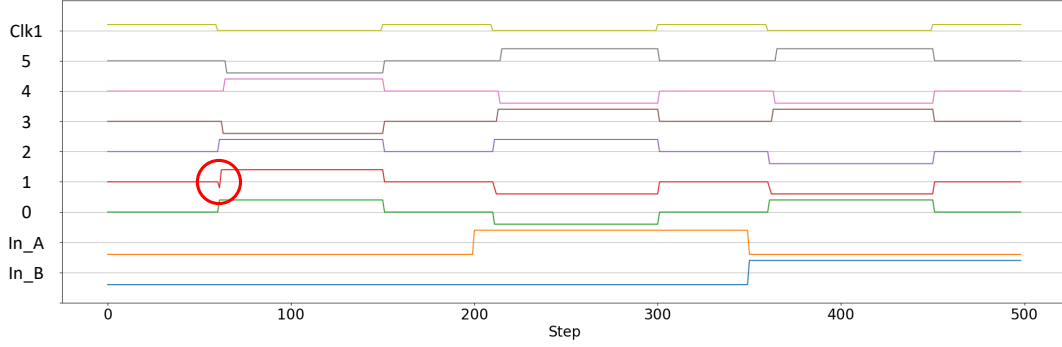


Figure 8.8: Waveforms obtained with the behavioral simulation of the OR gate.

as the previous circuits, while the inputs are set initially to “000”. After 20 ns inputs *A* and *C* are changed to “1” and at 35 ns also the input *B* is changed. The resulting waveforms are presented in Fig. 8.10. Magnet number five is the central one, the actual voter. In the circles, the principle of the behavioral simulation is highlighted. This magnet feels a stable input immediately (element one is directly influenced by the input) while the other neighbors need few more evaluation steps before reaching a stable value. When all the neighbors are stable the correct value is assigned and the final output is present in magnet nine. The simulation time for the majority voter structure is 50 ms. A circuit spanning along two clock zones is now presented. An inverter is placed at the output of the wire shown in Fig. 8.4 and assigned to clock zone two. The circuit is depicted in Fig. 8.11. This layout is used to demonstrate how the simulator handles different clock zones. The applied testbench is the same as the first circuit. Furthermore, the clock zone number two is present. The magnets in the waveforms are numbered as in Fig. 8.11, and the resulting elements of the inverter are named *Inv* plus an incremental index from one to four. The extra element is called *Extra* and the resulting waveforms are available in Fig. 8.12. This figure shows the inverter evaluation, that happens in the second clock zone. The vertical lines highlight the release of the reset for the two clock zones: firstly the clock one is released and the wire is evaluated. Then, the clock two goes to “0” and the inverter is evaluated. The clock one is then activated and the wire magnets reset before loading the new input. The simulation time for this structure was 52 ms. It is possible to see the latency due to the clock sequence: input sampled when the first clock is switched off are available after the second one is released. This aspect is clarified in the last example of behavioral simulations. A relatively complex architecture was simulated: a 2-to-1 multiplexer, Fig. 8.13. This layout is composed of more than fifty magnets, organized in four clock zones

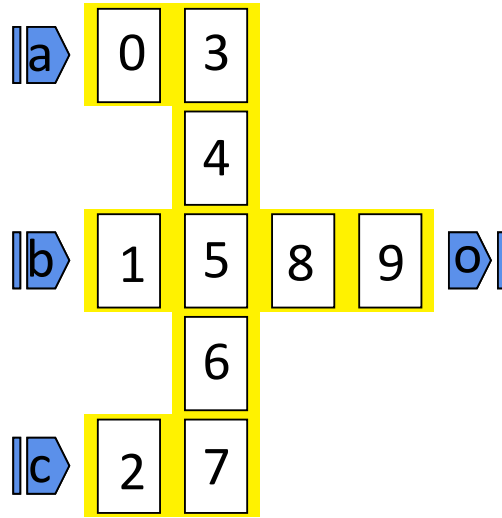


Figure 8.9: Layout of the majority voter.

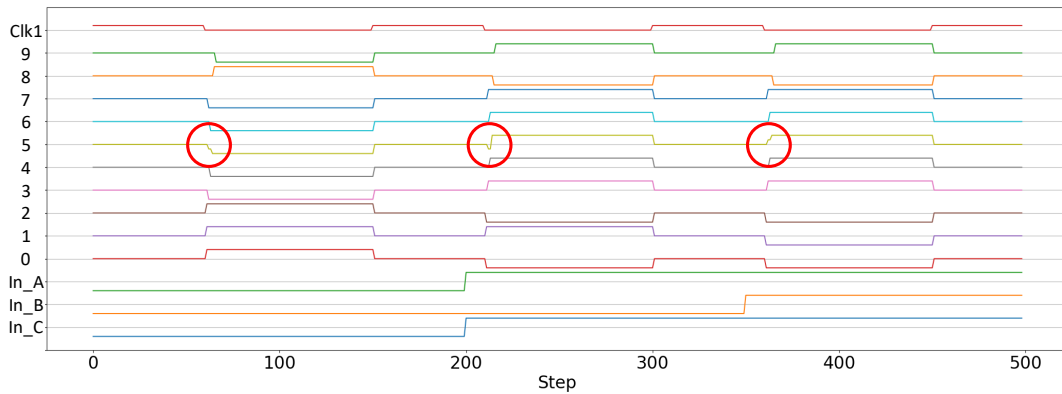


Figure 8.10: Waveforms obtained with the behavioral simulation of the majority voter gate.

(three different clock sources are available). The complete set of waveforms is not reported, but in Fig. 8.14 the clock sources, inputs pin, and the output magnet were selected and plotted. In the figure, the inputs-outputs relation is circled. The inputs sampled in the colored circles result in the corresponding output available in the successive clock cycle. In the same cycle, new inputs are sampled and this is repeated every time clock one is released. This proves that after the first clock cycle,

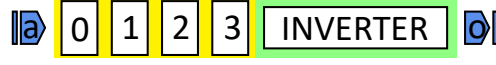


Figure 8.11: Layout of a wire followed by an inverter. The background of the inverter is different since it belongs to another clock zone.

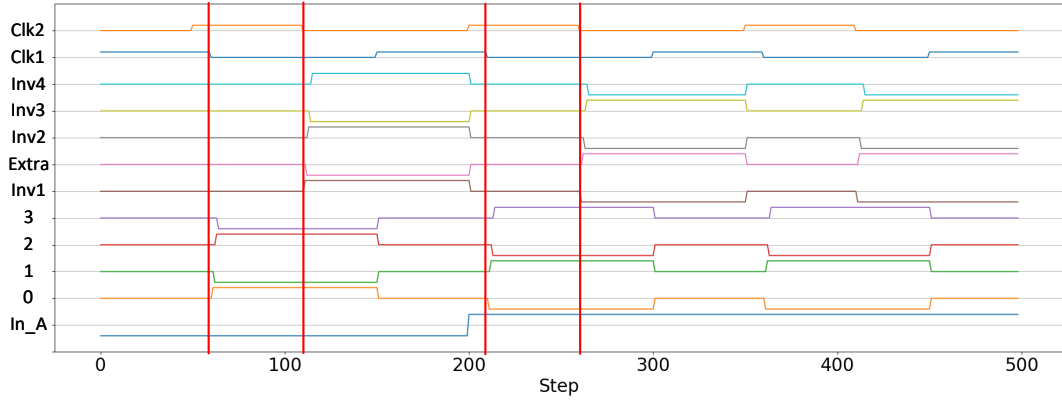


Figure 8.12: Waveforms resulting from the simulation of the wire followed by the inverter. The magnets resulting from the inverter expansion and the extra one inserted are visible.

the actual latency, the throughput is one output for each clock cycle. Considering the performance, the simulation of the mux for 100 ns lasts 300 ms. This confirms that the behavioral simulation engine can be used also for big layouts. Simulation results are available in a very small amount of time.

8.2 iNML Physical

In order to verify the correctness of the model adopted in the physical iNML simulation engine, several simulations were performed. In particular, building blocks normally used in iNML layouts were characterized changing the distance among the elements. This characterization was compared against **Mumax3** in order to have reference results for the different structures. For every circuit, horizontal and vertical distance ranged from 5 to 25 nm and all the possible input combinations were tested. This approach led to fifty simulations for the circuit with one input and two

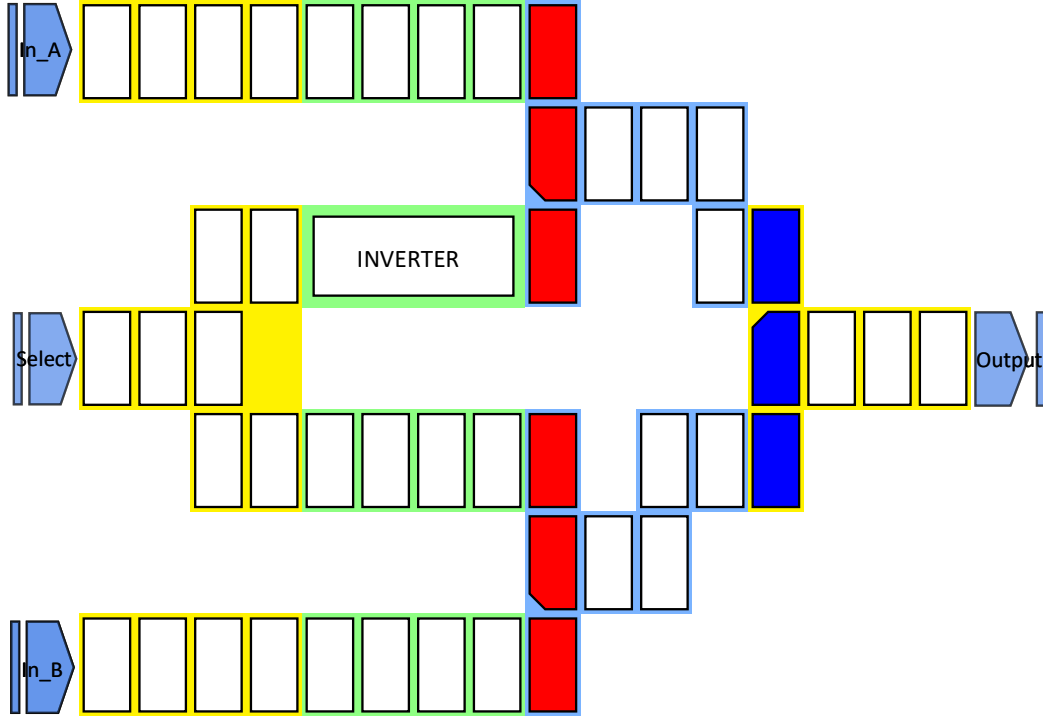


Figure 8.13: Final layout of the 2-to-1 multiplexer: input and output pins are the light blue elements with the name superimposed. Magnets of different colors belong to different clock zones. Logic gates are differently colored and it is possible to see the slanted edge of the central magnet.

hundred simulations for the MV. The characterizations were performed with the clock signal present in Fig. 8.15. This shape is compatible with the *Virtual Clock*: smaller magnets will be kept in reset during the falling part of the signal shape, behaving as if they were in the next clock zone. The clock is composed by a ramp from 0 to the maximum value ($130e3 \frac{A}{m}$) that last 1.5 ns. This value is kept for 5 ns and then an opposite ramp towards 0 is present. Finally, the elements are left for 2 ns without the clock field. All the simulations were performed with a step of $1e - 13$ seconds, export interval set to $1e - 12$ and export step equal to 1000. This configuration results in saving one step every $1e - 12$ seconds. Furthermore, the interaction radius is set to the maximum available: all the magnets interactions are considered. The magnets' dimensions are $60 \times 90 \times 10$ nm. The structure considered for the characterization are the following:

Horizontal Wire four magnets horizontally aligned, plus an input and an output.

Vertical Wire 2 two magnets vertically stacked, an input and an output.

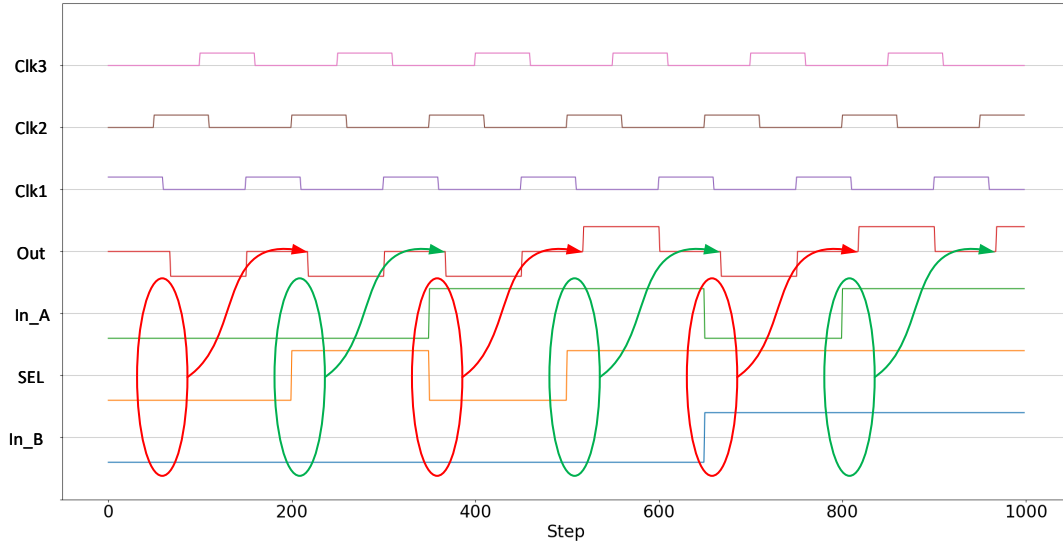


Figure 8.14: Waveforms of the mux simulation. Inputs and clock sources are shown, together with the output magnet. The circles are used to highlight the latency due to the clock mechanism.

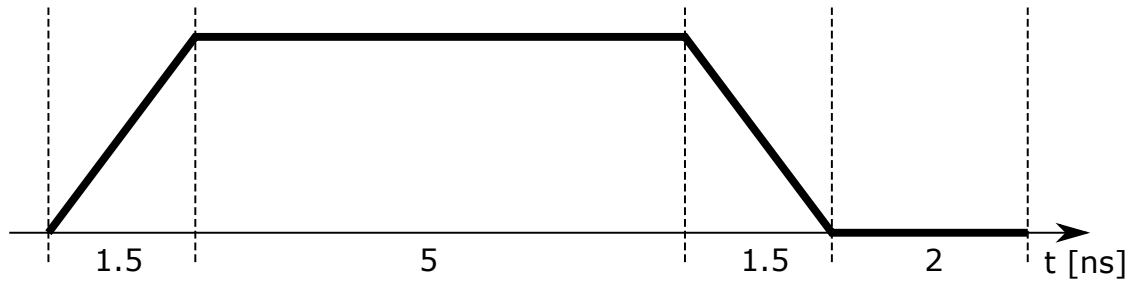


Figure 8.15: Shape of the clock signal applied to all the circuits here described. The duration of the different portions is reported in the figure.

Vertical Wire 3 three magnets vertically stacked, an input and an output.

“L” Shaped Wire 2 two magnets vertically stacked followed by three magnets horizontally aligned, an input and an output.

“L” Shaped Wire 3 three magnets vertically stacked followed by three magnets horizontally aligned, an input and an output.

“**Coupler**” an input aligned with the central magnet of a three elements stack. Top and bottom elements are followed by three magnets and an output.

Majority voter circuit composed of ten magnets where three inputs surround a central magnet. This magnet performs the voter functions and two elements are added to emulate a complete clock zone.

The structures here presented have at least an output magnet: in the behavioral simulation, the magnets in the other clock zones influence every element. Furthermore, the coupling not directed among the vertical and horizontal directions with a reset magnet is very important to determine if a circuit is behaving properly. The output pin was translated in magnet always in reset state placed in the same coordinates. These magnets are not influenced by the other elements of the circuit. For each circuit, an example of waveforms is presented and a table is used to show the characterization results.

The first circuit analyzed is the horizontal wire. The circuit simulated is the same depicted in Fig. 8.4. The waveforms obtained with horizontal distance equal to 15 nm are reported in Fig. 8.16. It is possible to notice that during the reset phase the X component of the magnetization vectors goes up to the saturation magnetization, while the other components almost get equal to zero. When the reset is removed the Y component increases and reaches the saturation magnetization with the opposite direction with respect to the previous magnet. Tables 8.1 summarize the results of the simulations verified with the dedicated script. The threshold used is 50%, meaning that the simulation is marked as correct if each magnet value is a half of saturation magnetization in the correct direction. Only the Y component is measured by the script. The same circuits were simulated using **Mumax3**. The mesh used in **Mumax3** was $5 \times 5 \times 5$ nm. After performing the simulations, the results were compared. The first two tables refer to the result obtained by the FCNS with the different input values, while the other two are the result of the same circuit obtained with **Mumax3**. In the case of the horizontal wire, the two simulators give the same results.

Similarly, the vertical wire with two stacked magnets was simulated. Fig. 8.17.A shows the MagCAD layout for the circuit. The simulation details for input equal to “0” are depicted in Fig. 8.18. Vertical and horizontal distances are 15 nm.

Also in this case the script was used to verify the simulation. The threshold was set again at 50%. The results are summarized in table 8.2. All the combination of horizontal and vertical distance give positive results. In the same table the results obtained with **Mumax3** are reported. The results are the same for both the simulators.

The next circuit under test is again a vertical wire, but three magnets were aligned in this case. Circuit view with the labeled magnets is reported in Fig. 8.17.B.

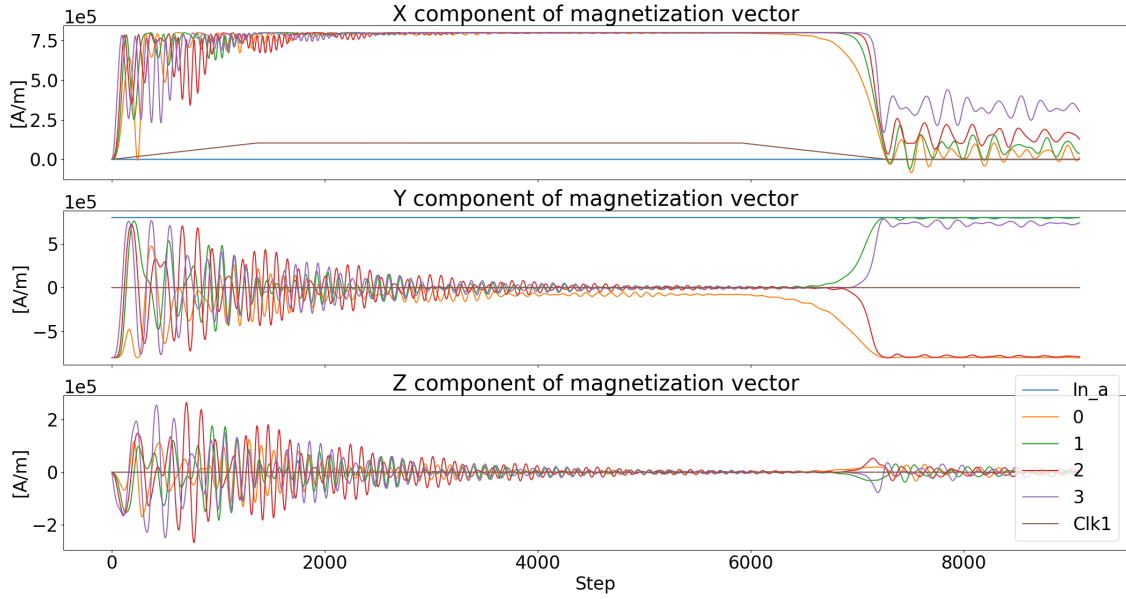


Figure 8.16: Graph of the physical simulation of the horizontal wire.

Table 8.1: Horizontal wire results.

FCNS

$Input = 0$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

$Input = 1$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

Mumax3

$Input = 0$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

$Input = 1$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

The visual inspection of simulation results shows directly problems in the circuit. In fact, none of the distances combination is working for input equal to “0”. Fig. 8.19

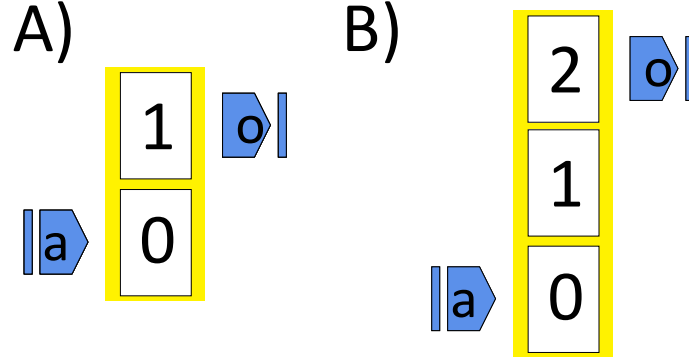


Figure 8.17: (A) Vertical wire with two magnets. (B) vertical wire with three magnets.

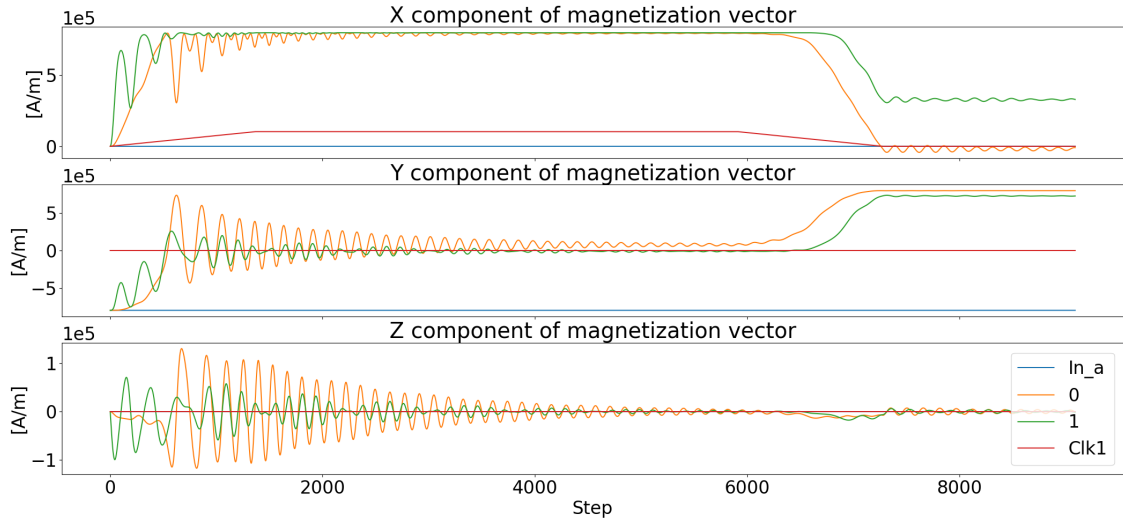


Figure 8.18: Graph of the simulation of the two magnets vertical wire. Both the elements assume the opposite value with respect to the input.

shows the output for both the distances equal to 15 nm. It is possible to see that the magnet labeled with “1” changes its magnetization as soon as the reset is removed towards the wrong direction. This is mainly due to the coupling with the output magnet that is in reset state. In fact, its influence is enough to push the central magnet in the opposite direction before the first magnet near the input acquires the

Table 8.2: Vertical wire two stacked magnets results.

FCNS													
$Input = 0$		Vertical Distance					$Input = 1$		Vertical Distance				
H Dist	5	✓	✓	✓	✓	✓	H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓		10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓		15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓		20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓		25	✓	✓	✓	✓	✓

Mumax3													
$Input = 0$		Vertical Distance					$Input = 1$		Vertical Distance				
H Dist	5	✓	✓	✓	✓	✓	H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓		10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓		15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓		20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓		25	✓	✓	✓	✓	✓

correct magnetization. The top magnet propagates the wrong information present in the central one. The verification script was used and the results are summarized

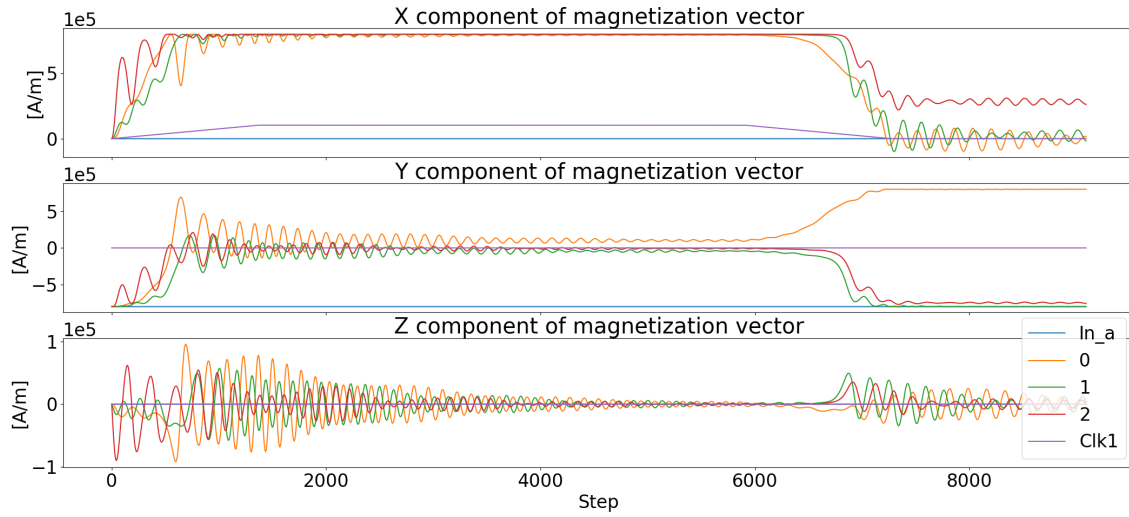


Figure 8.19: Simulation results of a non working configuration of the three elements vertical wire. Two out of the three elements have wrong direction.

in table 8.3. As mentioned before, all the simulations with input equal “0” are wrong. However, the remaining ones are correct but an additional fact needs to be considered. The central magnet is always getting the same value independently from the input, therefore the simulation can not be considered correct, but it is a coincidence.

Table 8.3: Vertical wire three stacked magnets results.

FCNS

<i>Input</i> = 0		Vertical Distance				
		5	10	15	20	25
H Dist	5	✗	✗	✗	✗	✗
	10	✗	✗	✗	✗	✗
	15	✗	✗	✗	✗	✗
	20	✗	✗	✗	✗	✗
	25	✗	✗	✗	✗	✗

<i>Input</i> = 1		Vertical Distance				
		5	10	15	20	25
H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

Mumax3

<i>Input</i> = 0		Vertical Distance				
		5	10	15	20	25
H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

<i>Input</i> = 1		Vertical Distance				
		5	10	15	20	25
H Dist	5	✗	✗	✗	✗	✗
	10	✗	✗	✗	✗	✗
	15	✗	✗	✗	✗	✗
	20	✗	✗	✗	✗	✗
	25	✗	✗	✗	✗	✗

The next structures extend the vertical wire to cover an entire clock zone. Each one is the combination of a horizontal wire and the correspondent vertical wire. The MagCAD layouts for the two structures are reported in Fig. 8.20. As expected, the first circuit works correctly with all the inputs and distances. A graph for input equals “0” and distances equal to 15 nm is reported in Fig. 8.21. The results of the verification script are available in table 8.4.

The other circuit instead shows the same critical issues of the vertical wire with the same amount of vertical stacked elements. Also in this case, the magnet labeled as “1” is wrongly influenced by the diagonal elements. In this circuit, the coupling with the element “3” is the one that causes the error. A positive outcome from this simulation is that the output in reset used to simulate the element in the next clock zones is a good choice. In fact, the behavior is the same as the previous condition. Furthermore, the other elements of the circuit are behaving correctly: from element “2” to element “5” the coupling is correct and an antiparallel orientation is present. Fig. 8.22 shows the outputs values for the wrong configuration. The simulation script was used to extract the results that are reported in table 8.5. Also here the

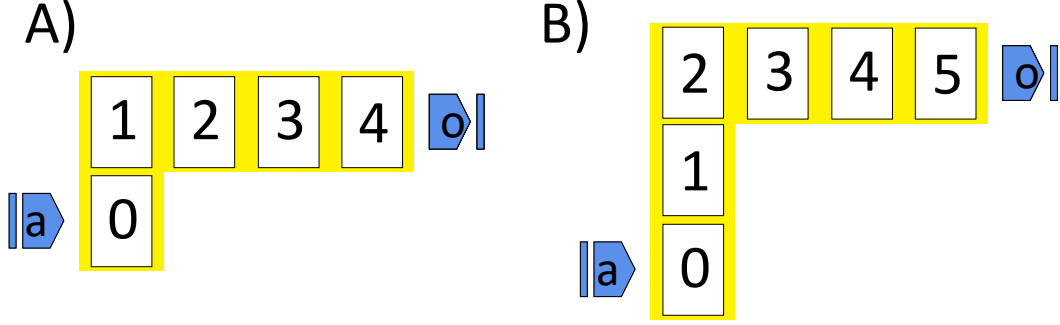


Figure 8.20: (A) “L” shaped wire. Two vertical stacked magnets and other three in order to get length equal four. (B) “L” shaped wire. Three vertical stacked magnets and the remaining to complete the horizontal wire.

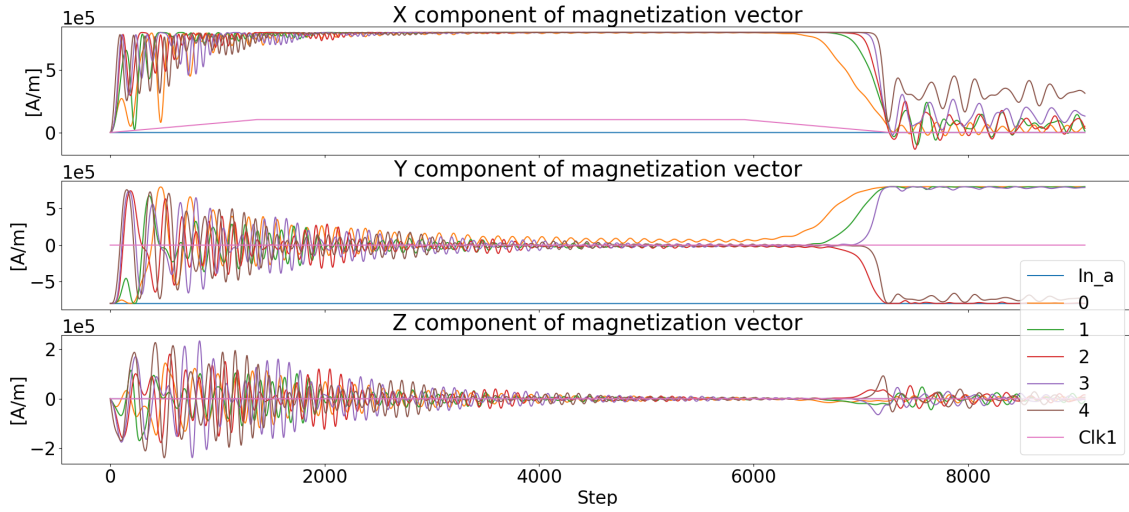


Figure 8.21: Working simulation of the two magnets “L” shaped wire.

correct simulation for input equal to “1” is due to the preferred direction of the wrong element and is not to be considered as actual correct results. In fact, in the tables are reported the simulation performed with Mumax3. The structure has the opposite behavior in this case. This is due to the initial conditions of the solver that lead to a different magnetization direction for the magnet number one. This comparison confirms that the structure is not working and should not be adopted in iNML circuits.

Table 8.4: “L” shaped wire with two vertical stacked magnets results.

FCNS													
$Input = 0$		Vertical Distance					$Input = 1$		Vertical Distance				
H Dist	5	✓	✓	✓	✓	✓	H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓		10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓		15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓		20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓		25	✓	✓	✓	✓	✓

Mumax3													
$Input = 0$		Vertical Distance					$Input = 1$		Vertical Distance				
H Dist	5	✓	✓	✓	✓	✓	H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓		10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓		15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓		20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓		25	✓	✓	✓	✓	✓

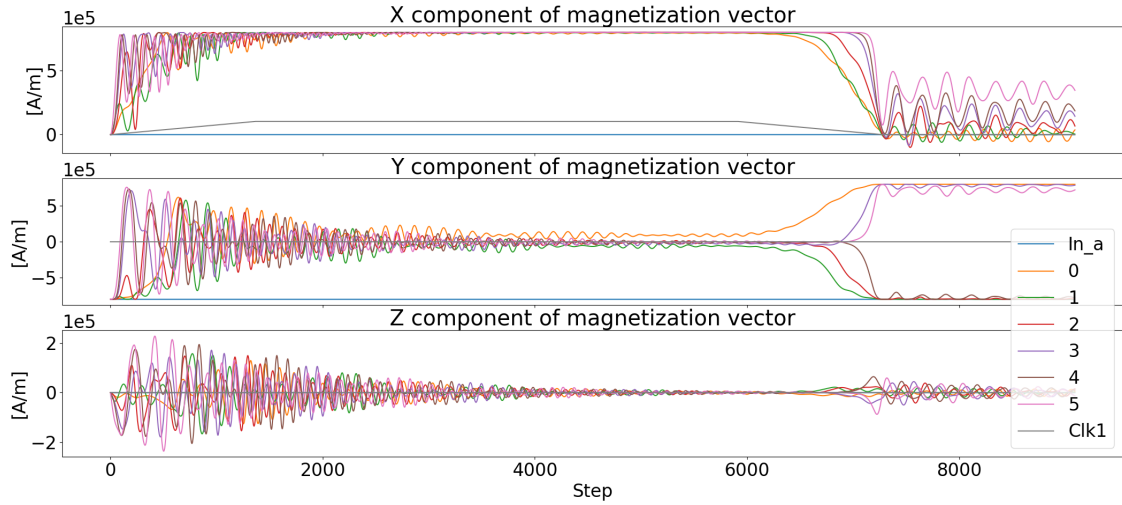


Figure 8.22: Wrong simulation of the three magnets “L” shaped wire.

The next circuit simulated is the “Coupler” structure. This circuit is used to split an input signal in two, doubling the fan-out of a wire. The adopted structure, composed of nine magnets is presented in Fig. 8.23. Two output pins are present, one at the end of each branch of the structure. The circuit was simulated and the

Table 8.5: “L” shaped wire with three vertical stacked magnets results.

FCNS

$Input = 0$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✗	✗	✗	✗	✗
	10	✗	✗	✗	✗	✗
	15	✗	✗	✗	✗	✗
	20	✗	✗	✗	✗	✗
	25	✗	✗	✗	✗	✗

$Input = 1$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

Mumax3

$Input = 0$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✓	✓	✓	✓	✓
	10	✓	✓	✓	✓	✓
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

$Input = 1$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✗	✗	✗	✗	✗
	10	✗	✗	✗	✗	✗
	15	✗	✗	✗	✗	✗
	20	✗	✗	✗	✗	✗
	25	✗	✗	✗	✗	✗

resulting values for distances equal to 15 nm and input equal “1” are reported in Fig. 8.24. From the graph it can be notice that the first magnet to change after

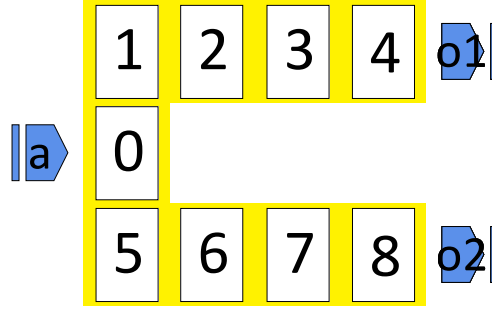


Figure 8.23: Coupler structure used to double the fan-out.

the reset is released is the central one. It assumes value opposite with respect to the input. Then the two magnets above and below the central one (“1” and “5”) align themselves in the same direction. The remaining elements behave like in the horizontal wire. Furthermore, the input information is doubled. The simulation script with threshold at 50% was utilized and the results are reported in table 8.6. Every combination of input and distances gave good results. Furthermore, the

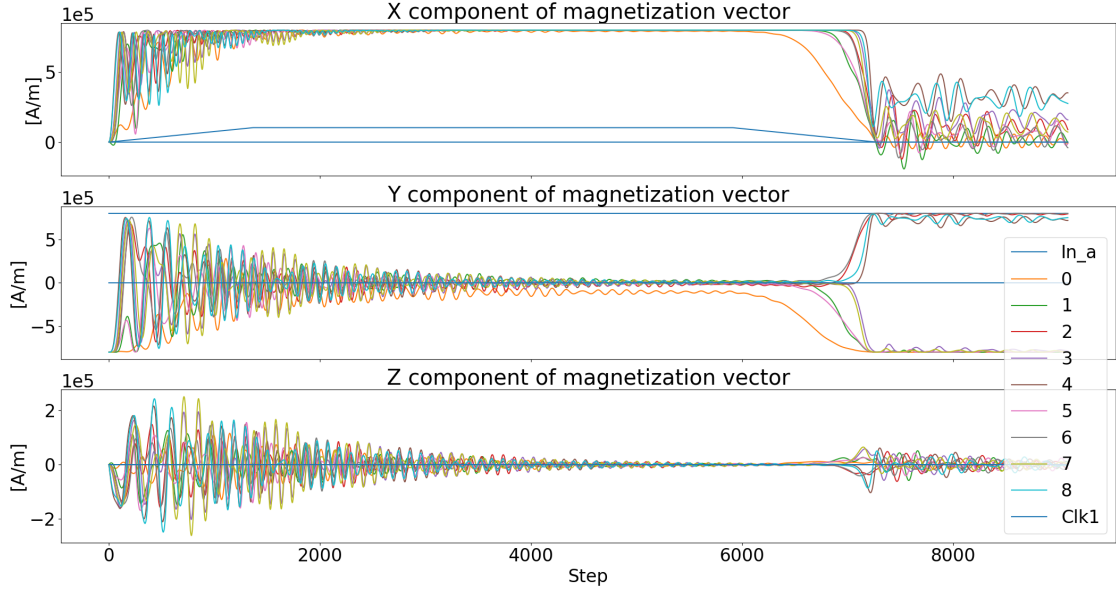


Figure 8.24: Simulation waveforms of the coupler. All the magnets are oriented correctly.

results obtained with **Mumax3** are reported in the table. It can be noticed that two configurations are not working correctly using the other simulator. Considering that the physical simulation engine is an approximate model of the one used by **Mumax3**, it is possible that some corner cases are not correctly evaluated.

Finally, the last structure analyzed is the majority voter. The circuit is presented in Fig. 8.25. This structure is the most complex due to the high number of diagonals couplings. In fact, the virtual clock mechanism was adopted to ensure the correct evaluation of the voter magnet. In the figure it is possible to notice that some magnets are smaller than the others. These elements, due to their dimensions ($60 \times 70 \times 10$), need a smaller reset field. This means that they will evaluate their value when the bigger ones are already stabilized. However, magnets placed too near the others avoid the correct behavior of the circuit. Fig. 8.26 shows an example of simulation where the correct information is retrieved at the output pin. Horizontal and vertical distance are set to 25 and 15 nm respectively and inputs are equal to “010”. In the graph can be noticed that the central magnet, labeled with number five, begins to switch some steps later with respect to the bigger ones. Furthermore, its direction is the same of the one on its left (magnet number one): an antiparallel coupling should be present but the other neighbors “vote” in the opposite direction. Finally, the two magnets on the right (labeled with eight and

an external field too strong, behaving therefore in the wrong manner. Different

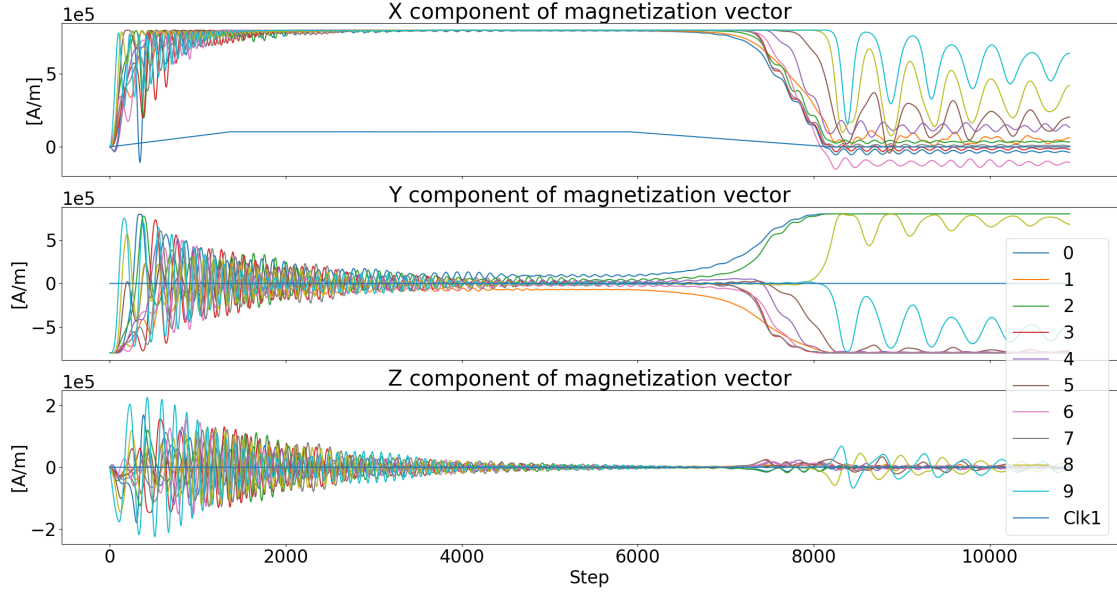


Figure 8.26: Simulation waveforms of the MV. Inputs are equal to “010”.

threshold values were used in the verification script in order to evaluate correctly the results. In fact a value of 50% is too restrictive: after 10 ns the magnets are not yet completely evaluated. In particular, the smaller magnet near the output magnets feel a strong external field directed on the X axis, resulting in a smaller component on the Y direction. In complete circuits, the magnets of the surrounding zones are reset and released in order to improve the propagation. For these reasons a threshold at 10% was used in this case. In order to reduce the number of tables only the results for the input “111” are reported. The other tables (*Input = all*) are used to have an overall representation of the circuit behavior. These results are in table 8.7. In these tables the ✓ means that every input combination was correct. A ✗ is present otherwise, even if only one input pattern was wrong. The tables show that the two simulators give slightly different results. However, the main differences are for vertical distance of 5 nm. This very small distance among the elements is a corner case that can be wrongly interpreted by the FCNS engine. However, the main scope of the developed tool is to give an alternative where the correctness of the result is trade for simulation time: FCNS is in fact faster than **mumax3** even if it runs on a CPU instead of a high parallel GPU. Table 8.8 shows the average execution time of the presented characterizations for each circuit. The table shows that FCNS outperforms **Mumax3** for every considered circuit. The mesh grid used in those simulations could be refined, leading to bigger simulations time.

Table 8.7: MV results.

FCNS

$Input = 111$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✖	✖	✖	✖	✖
	10	✖	✓	✓	✓	✖
	15	✓	✓	✓	✓	✓
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

$Input = all$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✖	✖	✖	✖	✖
	10	✖	✖	✖	✖	✖
	15	✓	✓	✓	✖	✖
	20	✓	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

Mumax3

$Input = 111$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✖	✖	✖	✖	✖
	10	✖	✓	✓	✖	✖
	15	✖	✓	✓	✓	✓
	20	✖	✓	✓	✓	✓
	25	✓	✓	✓	✓	✓

$Input = all$		Vertical Distance				
		5	10	15	20	25
H Dist	5	✖	✖	✖	✖	✖
	10	✖	✓	✖	✖	✖
	15	✖	✓	✓	✓	✖
	20	✖	✓	✓	✓	✖
	25	✖	✓	✓	✓	✓

Table 8.8: Simulation time, in seconds, comparison among FCNS and Mumax3.

	FCNS	Mumax3
Horizontal wire	6.14	112
Vertical wire 2 magnets	5.45	126
Vertical wire 3 magnets	5.82	119
L shaped wire 2 magnets	6.48	110
L shaped wire 2 magnets	6.87	106
Coupler	8.43	106
Majority voter	9.15	195

However, the number of elements is small, and the parallel computation on the GPU possible in Mumax3 is not exploited completely. A complexity analysis was performed using different benchmarks.

In order to make a comparison of the execution times among the two simulators, different structures were simulated. These structures are not used to verify the correctness of the engine, it was already demonstrated before, but only as “big” layouts capable of stressing the simulators. The circuit’s inputs and outputs are not considered in this case. The selected structures are squares composed of an increasing number of elements. Starting from a 4×4 block, doubling each time both the dimensions up to a 32×32 square of magnets. The magnets are just placed one

near the other: Fig. 8.27 shows the 4×4 version. These structures give the complexity analysis of the simulation engines. The simulation times over the number of elements are reported in Fig. 8.28. This figure shows the simulation times obtained by **Mumax3** and FCNS. Two curves are referred to the first simulator. Different mesh grid dimensions were used. The red curve refers to a $2 \times 2 \times 5$ grid, while the green one to a $5 \times 5 \times 5$ grid. FCNS was simulated with an infinite interaction radius (solid blue curve) and fixing the interaction radius to $2e - 6$ m (dashed blue curve). The last value of the radius is enough to include all the neighbors up to the 16×16 circuit, but it limits the number of neighbors in the bigger circuit. For the smaller circuits (up to 8×8) FCNS is faster than the other simulator. The more refined grid needs longer simulation time also on the GPU architecture. The 16×16 circuit shows similar execution times for FCNS and the coarse grain grid. The trend is inverted and FCNS is outperformed by **Mumax3** using the $5 \times 5 \times 5$ grid. However, the fine-grain grid still requires a longer time. The blue curve highlights the quadratic complexity $O(n^2)$ of FCNS. Fixing the interaction radius the complexity is reduced and a linear complexity is achieved. **Mumax3** performance, presented in [43], results in linear time for the fine mesh grid, while an exponential complexity that saturates in the coarse grain case. This is due to the GPU architectures: the high parallelism available is not completely exploited in the $5 \times 5 \times 5$ case. On the contrary, the higher number of cells in the finer mesh results in a higher complexity.

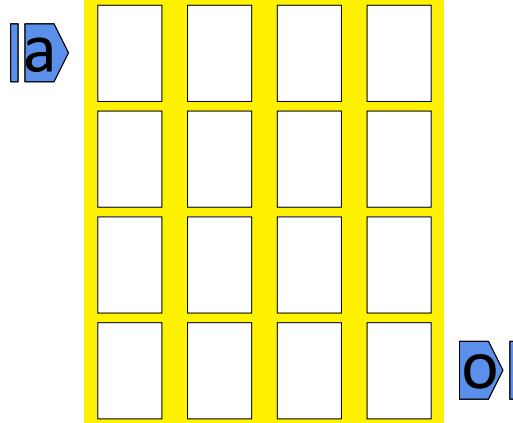


Figure 8.27: Example of circuit used to test the performance of the simulator.

The graph here reported shows the worst-case scenario for FCNS. In normal circuits, the area occupation is not as dense as in these examples. For example, the multiplexer circuit (bounding box 16×8) has only 52 elements. FCNS considers an element for each magnet while **Mumax3** needs to define the entire circuit area and

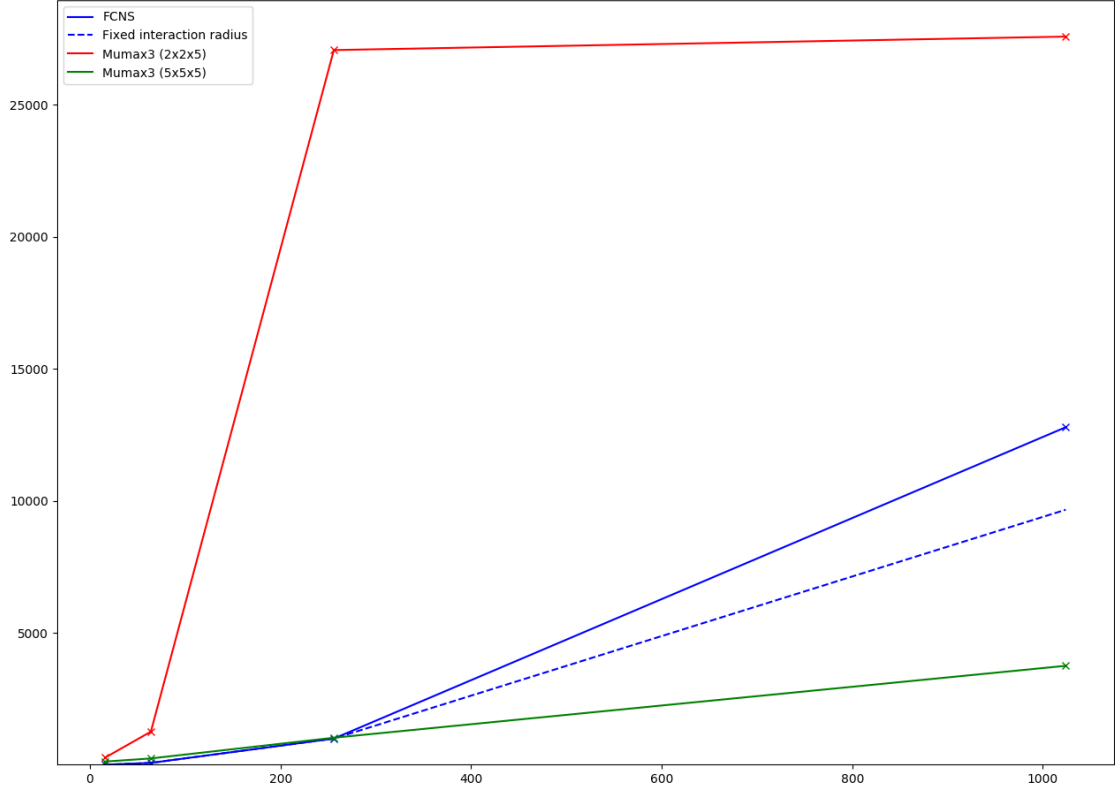


Figure 8.28: Performance graph comparing FCNS and **Mumax3**. Different mesh grid dimensions are used in **Mumax3**. The dashed line shows the performance of FCNS fixing the interaction radius at $2e - 6$ m. On the X-axis, the number of elements is reported. It can be computed multiplying the dimensions of the circuits: $4 * 4$ results in 16 elements, while $16x16$ is associated with 256 elements. The “x” marks on the chart refer to the measured circuit. On the Y-axis the elapsed time in seconds.

have a huge number of mesh elements. Furthermore, it is complex to describe the circuit in order to be simulated by **Mumax3**, while it is possible to use MagCAD or ToPoliNano to produce a file compatible with FCNS, reducing also the preparation phase.

8.3 MolQCA

The last simulation engine was tested with the three main components of a MolQCA circuit: wire, inverter, and MV. The circuit behavior was verified by inspection and a MatLab version of the algorithm (SCERPA) was used to compare the simulation performance. Finally, a wire composed of five hundred molecules was simulated for one step in order to extract simulation time for complex circuits. The settings used for the molecular simulations are the following: stability threshold is set to $2e - 7$, the damping factor to 0.6. and the interaction radius is set to infinite. The reset is applied and a step is evaluated, then the clock is raised and the information is propagated in the second step. The two steps are repeated changing the value of the inputs.

The first circuit analyzed is the molecular wire. It is composed of twelve molecules, belonging to the same clock zone as shown in Fig. 8.29. The simu-

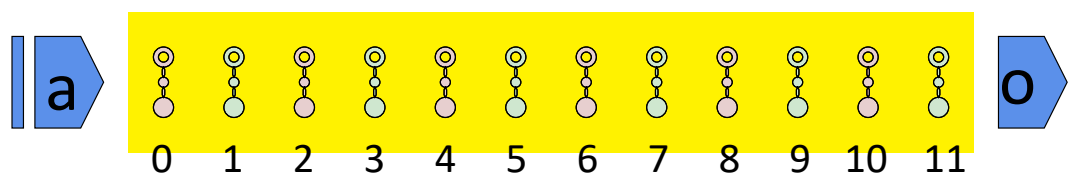


Figure 8.29: Molecular wire structure. Twelve molecules are placed in the same clock zone.

lation was performed, input initially is “0” and then it is set to “1” in the second part of the simulation. In Fig. 8.30 the waveforms are reported. The input equal “0” means that the charge for the input molecule is completely located in Q1. When the charge is in Q2, the molecule is equal “1”, finally, the molecule is said in reset if the charge is missing from the two logic dots. In the figure, it is possible to see the sequence of the molecules assuming their values. This figure plots all the intermediate steps evaluated by the simulation engine. In fact, the portion of the simulation where the clock is active (molecules are being evaluated) requires a higher amount of steps while the ones in reset are limited at a few of them even if the clock signal is a square wave. Fig. 8.31 shows the same simulation but only the stable steps are present in the plot. The graph is similar to a behavioral simulation, where all the molecules behave like square waves. Even if this last picture is easier to understand, the other one is more interesting. Furthermore, from the second image, it is not clear the values of the first molecules of the wire. The first picture, instead, shows clearly that the molecule near the input is the first one that

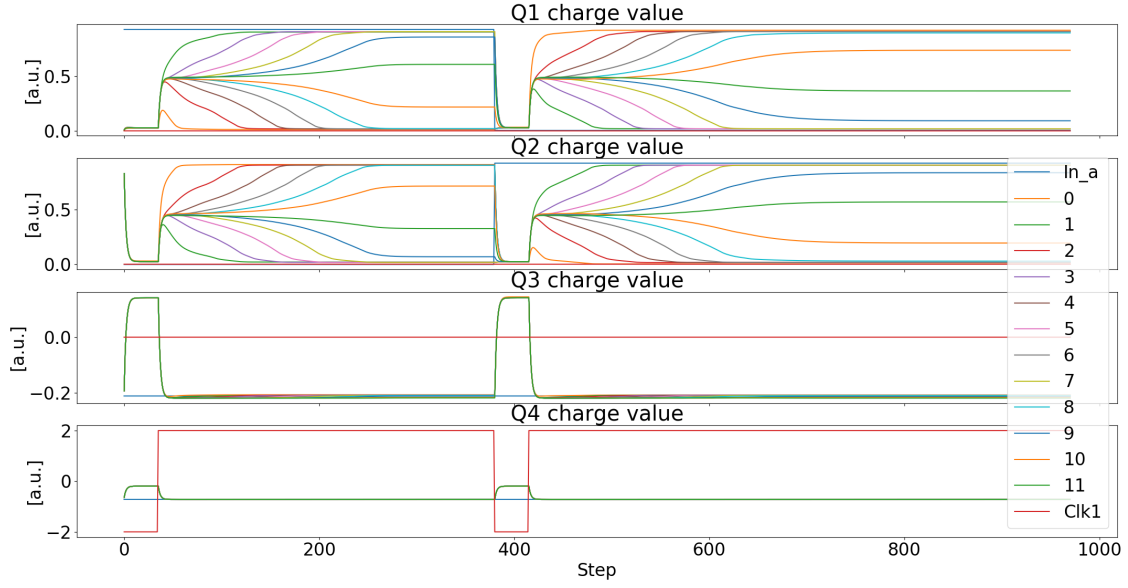


Figure 8.30: Simulation results of the horizontal wire.

is stabilized, and then it is followed by the remaining molecules. It can be noticed that the charge of the last molecule of the wire is never placed entirely in one of the two logic dots. This is due to the fact that the last molecule feels a lower effect from the other molecules: in fact, there are no more molecules on the right that will “help” the charge to move completely to one of the logic dots. The simulation time for the wire is 432 ms.

The inverter was then simulated. In MolQCA the inverter structure is more complicated than iNML. The simulated inverter is presented in Fig. 8.32. Input and output are doubled and two-molecules buses are used instead of wires. This design was preferred because doubling the number of molecules increases the stability of the circuit. The resulting circuit is composed of 64 molecules and two inputs. Three clock sources are available and also in this case, this condition results in latency from inputs to outputs. The clock signals are partially overlapped in order to guarantee the information propagation. The simulation results are reported in Fig. 8.33. If all the molecules’ values were present in the plot it would have been impossible to understand the circuit behavior. Therefore, only few molecules are selected and plotted in the figure. Furthermore, the charges of the last two dots were summed and plotted together to increase the readability of the plot. “Q1” and “Q2”, the logic dots, are more interesting. At the beginning the inputs are set to logic “0”, so the charge is in the first logic dot. The charge in molecule zero and

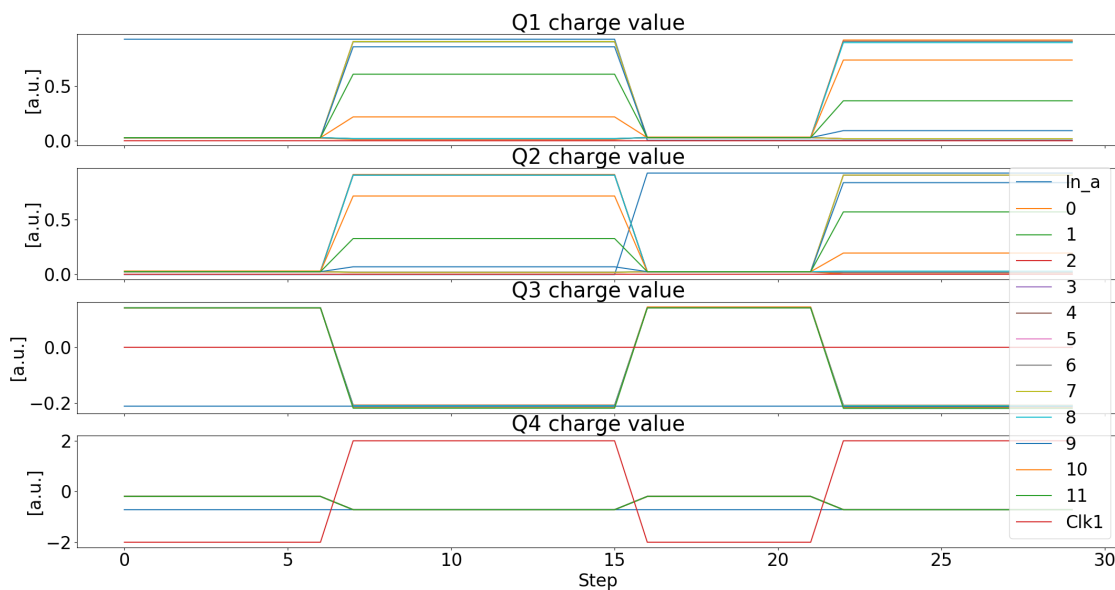


Figure 8.31: Simulation results of the horizontal wire. Only the stable step are used in the plots.

one is completely localized in the first dot, the same as the input. The same thing happens for molecule two and three. However, these molecules are in the second clock zone and the evaluation begins when the corresponding clock signal is high. Finally, the last two molecules show the opposite localization of the charge. This confirms the inverter behavior. The inputs are then flipped and the same process is repeated with the new values. The simulation of the inverter considering the stability of three clock zones with the two inputs last 17 seconds.

Finally a majority voter was tested. Fig. 8.35 shows the MagCAD layout of this circuit. Also in this case the wire are replaced with two-molecules buses and three clock zones can be identified. In particular, the central block where the majority function is performed is placed entirely in a different clock zone. Input and output wires are placed in clock zone one and three respectively. This choice is need to reduce the interactions from other molecules during the majority evaluation. Every inputs combination was tested but only three of them are here reported: “111”, “010” and “001”. Furthermore, only few elements are placed in the plot and the charges of Q3 and Q4 are placed in the same graph. The plot of the simulaion is reported in Fig. 8.34.

As expected the molecules labeled with “0”, “1” and “2” are evaluated in the first clock zone, and they are oriented as the input pins. In the third clock zone,

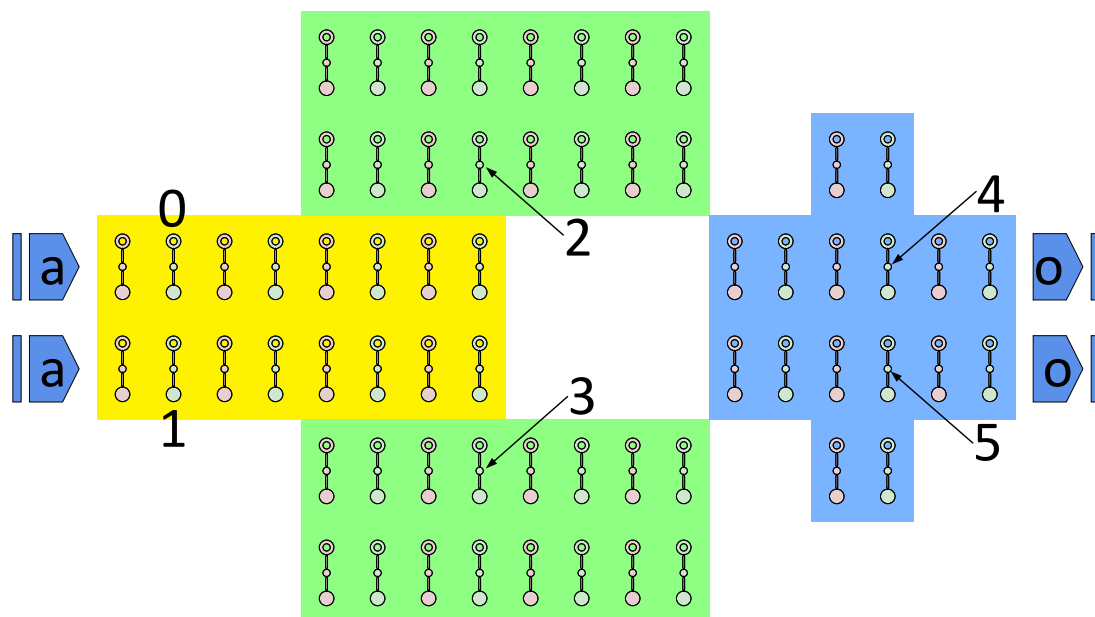


Figure 8.32: MagCAD layout of the inverter based on MolQCA technology.

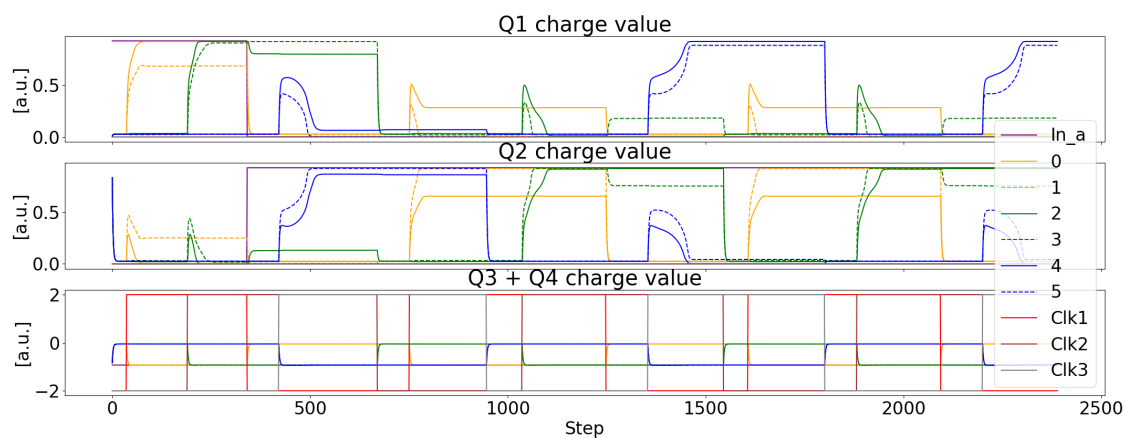


Figure 8.33: Simulation results of the molecular inverter. Both input values are showed. Solid and dashed lines are used to show the couples of molecules. In each couple, the molecules have the same logic value but the charges are slightly different. This is due to a non symmetric behavior of the molecules.

two molecules are reported. This shows that one of the two can have all the charge

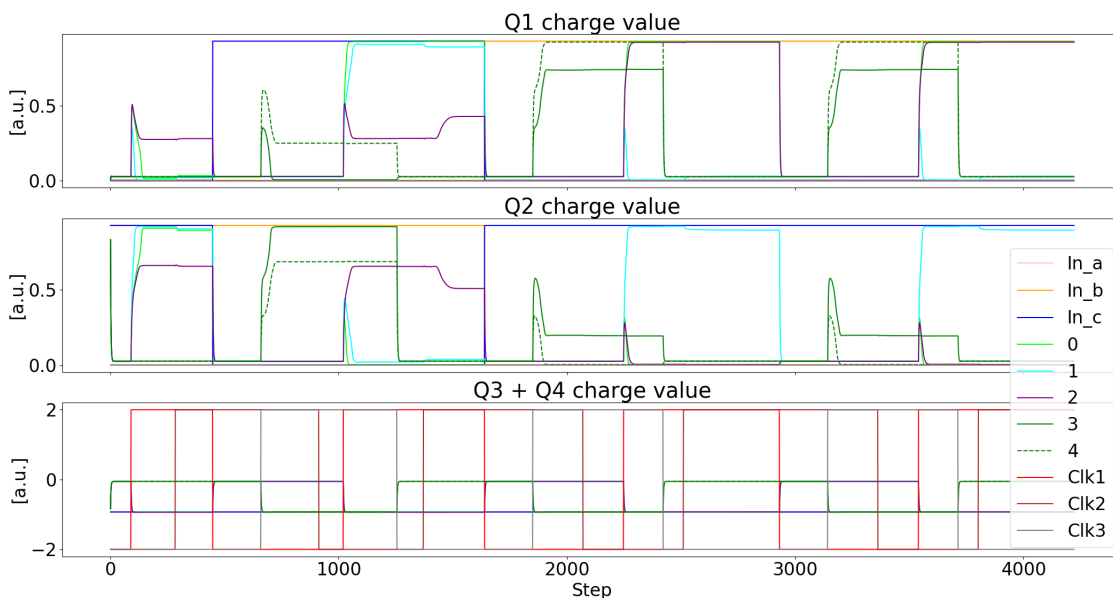


Figure 8.34: Simulation results of the molecular MV. One molecule per input wire is plotted. Two molecules, solid and dashed, are used to show the final output.

in one of the logic dot, but not in the other. The other molecule, on the contrary, shows the opposite behavior. This is due to the fact that the molecules in the bus are not completely symmetric. The inputs pattern is then changed and the correct new value is present at the output. The simulation of the MV for the three input pattern lasts roughly 30 seconds. The figure shows that more than four thousands of convergence step were evaluated.

FCNS molecular engine was compared with SCERPA [3], a Matlab script implementing a similar version of the simulation algorithm. Unfortunately, also SCERPA is in the development phase and a complete comparison is not reported. However, the simulation correctness can be visually verified from the waveforms. To evaluate the performance of the two tools a very long wire of 500 molecules in the same clock zone was simulated for only one convergence step. SCERPA took almost eighteen hours to complete the simulation. FCNS, on the contrary, was able to complete the same computation in only sixty-two minutes. This result could be even improved inserting the multithreading approach also in the molecular engine.

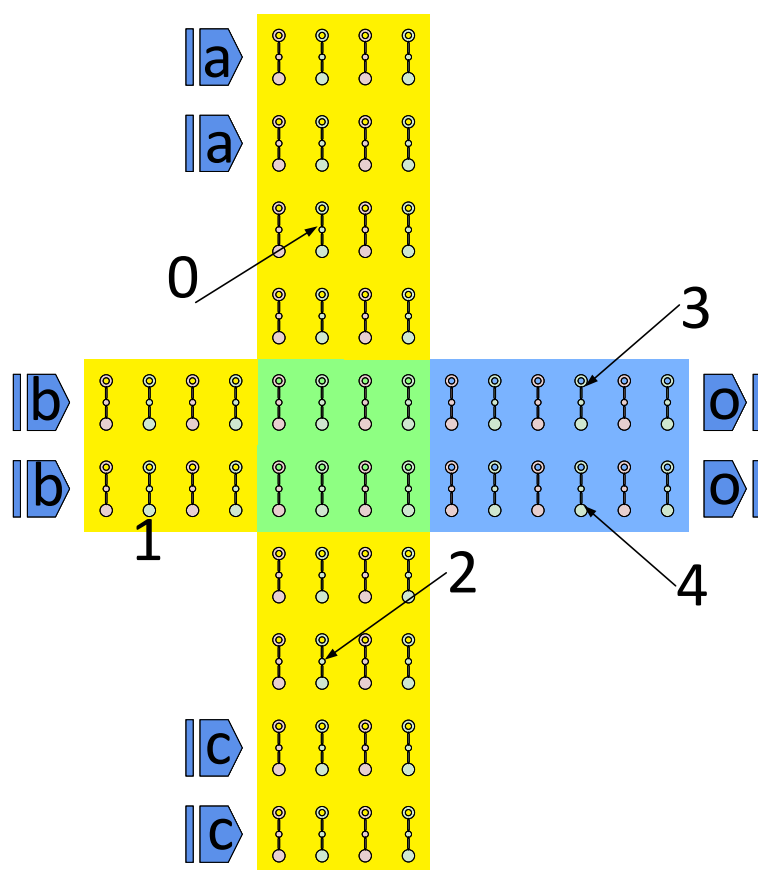


Figure 8.35: MagCAD layout of the MV based on MolQCA technology.

Chapter 9

Conclusion and Future Works

The contributions described in this thesis were developed in order to improve the analysis and the development of beyond-CMOS technologies. The framework covers the design and the simulation phase. Furthermore, the possibility of studying the effect of parameters variation in the technologies is a key point. Every part of the framework was developed with a “general” approach. Starting from the support of different technologies and how the modularity of the tools is able to adapt to the different needs. The user can select different values for the technological settings in the layout. Furthermore, change the constraint during the automatic generation of the circuits. The new simulator defines a general structure, independent of the selected technology. Furthermore, it is possible to define different approaches for the simulation: the definition of the interaction is enough to define a new simulation engine. The simulator principle is adaptable to all the FCN technologies.

The possibility of varying physical parameters, like element position and rotation, in the simulation phase can be a huge help in the study of the manufacturability and reliability of the technologies. With the ToPoliNano framework it is possible to take into account manufacture defects and process variations also in this stage of the research.

ToPoliNano and MagCAD are available online and used by the scientific community. The simulator will become part of the ToPoliNano tool. During my Ph.D., the tools were firstly released online and then updated frequently to include the developed functionality.

Future works are planned on the entire framework. Focusing on the simulator, new technologies are being studied and could be supported in the future. pNML technology is among them. However, pNML is different from the other field-coupled technologies. It is based on the same principle but also on the domain walls propagation inside the magnets needs to be modeled and evaluated.

The performance of the simulator could be still increased. Moving the evaluation part on a GPU could reduce even further the simulation time also for the iNML physical simulation engine.

Bibliography

- [1] A. Aharoni. “Demagnetizing factors for rectangular ferromagnetic prisms”. In: *Journal of Applied Physics* 83.6 (Mar. 1998), pp. 3432–3434. ISSN: 0021-8979.
- [2] M.T. Alam et al. “Clock Scheme for Nanomagnet QCA”. In: *International Conference on Nanotechnology*. Hong Kong: IEEE, 2007, pp. 403–408.
- [3] Y. Ardesi et al. “SCERPA: a Self-Consistent Algorithm for the Evaluation of the Information Propagation in Molecular Field-Coupled Nanocomputing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019), pp. 1–1. ISSN: 1937-4151. DOI: 10.1109/TCAD.2019.2960360.
- [4] P. Avouris et al. “Carbon nanotube electronics”. In: *Proceedings of the IEEE* 91.11 (2003), pp. 1772–1784. ISSN: 1558-2256. DOI: 10.1109/JPROC.2003.818338.
- [5] M. Becherer et al. “A monolithic 3D integrated nanomagnetic co-processing unit”. In: *Solid-State Electronics* 115, Part B (2016). Selected papers from the EUROSOI-ULIS conference, pp. 74 –80. ISSN: 0038-1101. DOI: <http://dx.doi.org/10.1016/j.sse.2015.08.004>.
- [6] Robert Brayton and Alan Mishchenko. “ABC: An Academic Industrial-strength Verification Tool”. In: *Proceedings of the 22Nd International Conference on Computer Aided Verification. CAV’10*. Edinburgh, UK: Springer-Verlag, 2010, pp. 24–40. ISBN: 3-642-14294-X, 978-3-642-14294-9.
- [7] S. Breitkreutz et al. “1-Bit Full Adder in Perpendicular Nanomagnetic Logic using a Novel 5-Input Majority Gate”. In: *EPJ Web of Conferences* 75.05001 (2014).
- [8] Stephan Breitkreutz et al. “Nanomagnetic logic: compact modeling of field-coupled computing devices for system investigations”. In: *Journal on Computational Electronics* (2011).
- [9] David Bryan. “ISCAS ’85 benchmark circuits and netlist format”. In: *International Symposium on Circuits and Systems*. Kyoto: IEEE, 1985.

- [10] F. Cairo et al. “Out-of-plane NML modeling and architectural exploration”. In: *Nanotechnology (IEEE-NANO)*, 2015 IEEE 15th International Conference on. 2015, pp. 1037–1040. DOI: 10.1109/NANO.2015.7388798.
- [11] T. Chen. “Overcoming research challenges for CMOS scaling: industry directions”. In: *2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*. 2006, pp. 4–7.
- [12] R.P. Cowburn and M.E. Welland. “Room temperature magnetic quantum cellular automata”. In: *Science* 287 (2000), pp. 1466–1468.
- [13] G Csaba, M Becherer, and W Porod. “Development of CAD tools for nanomagnetic logic devices”. In: *International Journal of Circuit Theory and Applications* 41.6 (2013), pp. 634–645. ISSN: 1097-007X.
- [14] A.I. Csurgay, W. Porod, and C.S. Lent. “Signal processing with near-neighborcoupled time-varying quantum-dot arrays”. In: *IEEE Transaction On Circuits and Systems* 47.8 (2000), pp. 1212–1223.
- [15] M. Donahue and D. Porter. *OOMMF User’s Guide, Version 1.0*. Tech. rep. National Institute of Standards and Technology, 1999.
- [16] *EPFL logic synthesis libraries*. <https://lorina.readthedocs.io/en/latest/index.html>.
- [17] M.J. Frisch et al. *Gaussian09 Revision A.1*. 2009.
- [18] A. Ferrara et al. “3D design of a pNML random access memory”. In: *2017 13th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*. 2017, pp. 5–8. DOI: 10.1109/PRIME.2017.7974093.
- [19] R. E. Formigoni, R. S. Ferreira, and J. A. M. Nacif. “Ropper: A Placement and Routing Framework for Field-Coupled Nanotechnologies”. In: *2019 32nd Symposium on Integrated Circuits and Systems Design (SBCCI)*. 2019, pp. 1–6.
- [20] *Free peer-reviewed portable C++ source libraries*.
- [21] U. Garlando et al. “Architectural exploration of perpendicular Nano Magnetic Logic based circuits”. In: *Integration* 63 (2018), pp. 275 –282. ISSN: 0167-9260.
- [22] Jelle J. W. Goertz et al. “Domain wall depinning from notches using combined in- and out-of-plane magnetic fields”. In: *AIP Advances* 6.5, 056407 (2016). DOI: <http://dx.doi.org/10.1063/1.4944698>.
- [23] M. Graziano et al. “A NCL-HDL Snake-Clock Based Magnetic QCA Architecture”. In: *IEEE Transaction on Nanotechnology* 10.5 (Sept. 2011), pp. 1141–1149. ISSN: 1536-125X.
- [24] Tomaž Hočevar and Janez Demšar. “Computation of Graphlet Orbits for Nodes and Edges in Sparse Graphs”. In: *Journal of Statistical Software* 71.10 (2016), pp. 1–24. DOI: 10.18637/jss.v071.i10.

- [25] A. Imre et al. “Magnetic Logic Devices Based on Field-Coupled Nanomagnets”. In: *2005 International Semiconductor Device Research Symposium* (2005), p. 25. ISSN: 10.1109/ISDRS.2005.1595958.
- [26] *International Roadmap for Devices and Systems*. <https://irds.ieee.org/editions/2018>. 2019.
- [27] *International Technology Roadmap of Semiconductors*. <https://www.semiconductors.org/resources/international-technology-roadmap-for-semiconductors-itsr/>. 2015.
- [28] Judith Kimling et al. “Tuning of the nucleation field in nanowires with perpendicular magnetic anisotropy”. In: *Journal of Applied Physics* (2013).
- [29] T. D. Ladd et al. “Quantum computers”. In: *Nature* 464.7285 (2010), pp. 45–53. ISSN: 1476-4687. DOI: 10.1038/nature08812. URL: <https://doi.org/10.1038/nature08812>.
- [30] C.S. Lent et al. “Quantum cellular automata”. In: *Nanotechnology* 4.1 (1993), p. 49.
- [31] *Mentor Graphics*. <http://www.modelsim.com>.
- [32] M. Niemier and al. “Nanomagnet logic: progress toward system-level integration”. In: *J. Phys.: Condens. Matter* 23 (Nov. 2011), p. 34. DOI: 10.1088/0953-8984.
- [33] Wolfgang Porod et al. “Nanomagnet Logic (NML)”. In: *Field-Coupled Nanocomputing: Paradigms, Progress, and Perspectives*. Ed. by Neal G. Anderson and Sanjukta Bhanja. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 21–32.
- [34] F. Riente, D. Melis, and M. Vacca. “Exploring the 3-D Integrability of Perpendicular Nanomagnet Logic Technology”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.7 (2019), pp. 1711–1719.
- [35] F. Riente et al. “MagCAD: Tool for the Design of 3-D Magnetic Circuits”. In: *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 3 (2017), pp. 65–73. ISSN: 2329-9231.
- [36] F. Riente et al. “ToPoliNano: A CAD Tool for Nano Magnetic Logic”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.7 (2017), pp. 1061–1074. ISSN: 0278-0070.
- [37] F. Riente et al. “Towards Logic-In-Memory circuits using 3D-integrated Nanomagnetic logic”. In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 2016, pp. 1–8. DOI: 10.1109/ICRC.2016.7738700.
- [38] Fabrizio Riente et al. “Controlled data storage for non-volatile memory cells embedded in nano magnetic logic”. In: *AIP Advances* 7.5 (2017), p. 055910.

- [39] N. Rizos et al. “Clocking Schemes for Field Coupled Devices from Magnetic Multilayers”. In: *International Workshop on Computational Electronics*. Beijing, China: IEEE, 2009, pp. 1–4.
- [40] G. Turvani et al. “A compact physical model for the simulation of pNML-based architectures”. In: *AIP Advances* 7.5 (2017), p. 056005. DOI: 10.1063/1.4974015.
- [41] M. Vacca et al. “Magnetoelastic Clock System for Nanomagnet Logic”. In: *Ieee Transaction On Nanotechnology* 13.5 (2014).
- [42] V. Vankamamidi, M. Ottavi, and F. Lombardi. “Two-Dimensional Schemes for Clocking/Timing of QCA Circuits”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.1 (2008), pp. 34–44. ISSN: 0278-0070.
- [43] A. Vansteenkiste and B. Van de Wiele. “MuMax: A new high-performance micromagnetic simulation tool”. In: *Journal of Magnetism and Magnetic Materials* 323.21 (2011), pp. 2585–2591. ISSN: 0304-8853. DOI: <https://doi.org/10.1016/j.jmmm.2011.05.037>. URL: <http://www.sciencedirect.com/science/article/pii/S0304885311003064>.
- [44] Arne Vansteenkiste et al. “The design and verification of MuMax3”. In: *AIP Advances* 4.10 (2014).
- [45] E. Varga et al. “Experimental Realization of a Nanomagnet Full Adder Using Slanted-Edge Magnets”. In: *IEEE Transactions on Magnetism* 49.7 (2013), pp. 4452–4455.
- [46] Marcel Walter et al. *fiction: An Open Source Framework for the Design of Field-coupled Nanocomputing Circuits*. 2019. arXiv: 1905.02477.
- [47] K. Walus et al. “QCADesigner: A Rapid Design and Simulation Tool for Quantum-Dot Cellular Automata”. In: *IEEE Transaction on Nanotechnology* 3.1 (2004). ISSN: 10.1109/TNANO.2003.820815.
- [48] R. Wang et al. “An effective algorithm for clocked field-coupled nanocomputing paradigm”. In: *2016 IEEE Nanotechnology Materials and Devices Conference (NMDC)*. 2016, pp. 1–2. DOI: 10.1109/NMDC.2016.7777166.
- [49] R. Wang et al. “Effect of a Clock System on Bis-ferrocene Molecular QCA”. In: *IEEE Transactions on Nanotechnology* PP.99 (2016), pp. 1–1. ISSN: 1536-125X. DOI: 10.1109/TNANO.2016.2555931.

This Ph.D. thesis has been typeset by means of the T_EX-system facilities. The typesetting engine was pdfL^AT_EX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete T_EX-system installation.