POLITECNICO DI TORINO

Department of Electronics and Telecommunications
Master Degree in Electronic Engineering

Master Thesis

Analysis of the block-based circuit design in molecular Field-Coupled Nanocomputing

Supervisors
Prof. Gianluca Piccinini
Prof. Mariagrazia Graziano
Dott. Yuri Ardesi
Dott. Giuliana Beretta

Candidate
Flavio Lupoli

Academic Year 2022-2023

Summary

Since it was introduced in the '90s, CMOS technology has brought an important improvement to modern electronics, being significantly suitable for the progressive development of ICs in terms of area, number of transistors per chip and channel length. As predicted by Moore's laws in the '70s, the evolution of these features has followed an exponential trend until now. However, this development curve is expected to encounter a saturation soon: in a few years, it will not be possible to scale down devices and raise the number of transistors per chip at the current rate, due to significant short-channel effects (related to the reduced dimensions of the devices) that negatively influence their performances. This technological limit led to the exploration of innovative approaches that could overcome the problem, essentially divided in two main categories: the "More than Moore" approach, which exploits the existing CMOS technology to integrate devices belonging to different domains on the same SoC (System-on-Chip), and the "Beyond CMOS" approach, which considers alternative technologies able to overcome the standard CMOS limits and, at the same time, to realize a logic that works the same way.
This thesis project focuses on Field-Coupled Nanocomputing (FCN), one of the most promising realizations of the latter approach, and in particular its molecular implementation, which uses blocks of 2 molecules each (QCA) coupled with electrostatic fields. Differently from standard CMOS logic, where logic states are defined by charge transport, in this configuration logic information is coded depending on where the charge is localized within the molecules. This brings significant advantages, such as low power, higher operating frequencies and the possibility to realise strongly scaled devices.
The aim of this thesis project is to develop and optimize a Characterization Tool capable of obtaining the in-out characteristic of logic gates and interconnections realised with the molecular implementation of FCN, in the most generalized and automated way possible, considering the physical effects due to their real implementation. The Characterization Tool exploits SCERPA, a MATLAB algorithm that evaluates information propagation in molecular FCN layouts, using an iterative procedure that considers the contribution of externally applied voltages. Its workflow can be divided into two macro-steps: as a first step, it elaborates the data obtained from SCERPA simulations and, depending on the kind of layout, rearranges them in libraries where the inputs and the related outputs are saved; in its second part, it requires the user to insert a combination of inputs and returns the corresponding outputs of the whole structure, consulting the previously created libraries.
The main advantage of the Characterization Tool is that it returns accurate output values if compared to the SCERPA process alone, with a significantly lower time overhead, for any generic layout. In addition, giving the possibility of creating a whole library of components, it allows the user to simulate any combination of these as many times as needed, speeding up the simulations of more complex circuits.

An introductory explanation of the theoretical principles behind the molecular implementation of FCN and SCERPA, together with the related state-of-the-art technological applications and most useful features, is contained in Part I of this thesis project. Furthermore, Part II focuses on the development of the Characterization tool, with the aim of describing how information is elaborated and which data structures are employed, with particular concern for the automation of the process. Finally, Part III deals with the complete simulation and characterization of a XOR gate in order to demonstrate quantitatively the advantages of the depicted method on a more complex layout. In the last part of the document some considerations are highlighted about the obtained results and some possible future improvements for the method are proposed.

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Technology overview

1.1 Moore's laws

Modern electronics technologies and processes strongly rely on the aggressive integration of circuits, which, as a consequence, comes with a particular concern for the capability to improve integration density, power consumption and performance of the device.
Moore's laws predicted with an impressive consistency some fundamental technology development trends: the number of transistors on a chip, the IC's area and the transistor's channel length. [5] In particular, the number of transistors was predicted to double on chips every 1.5 years, following the rule:

$$\frac{N_{tr}}{IC}(t) = \frac{N_{tr}}{IC}(t_0) \cdot 2^{\frac{t-t_0}{1.5}} \tag{1.1}$$

In addition, IC's area would increase by 50% every 3 years and the channel length of the transistors would decrease by 50% over the same amount of time:

$$A_{IC}(t) = A_{IC}(t_0) \cdot 1.5^{\frac{t-t_0}{3}} \tag{1.2}$$

$$L_{CH}(t) = L_{CH}(t_0) \cdot 2^{-\frac{t-t_0}{6}} \tag{1.3}$$

Despite being formulated in the early '70s, Moore's laws are still valid, as shown in Figure 1.1. Throughout the years, the progressive scaling of devices has allowed to reach better performances (i.e. fastest response and lower power consumption) and reduced costs.

1.2 Beyond CMOS technologies

The CMOS circuital technology, introduced in the '90s, revolutioned modern electronics and it is still the most diffused one today. However, the constraints related to the physical structure of the materials involved are expected to become a problem soon. [12] In the near future, it will not be possible to keep up with the trends mentioned before, due to

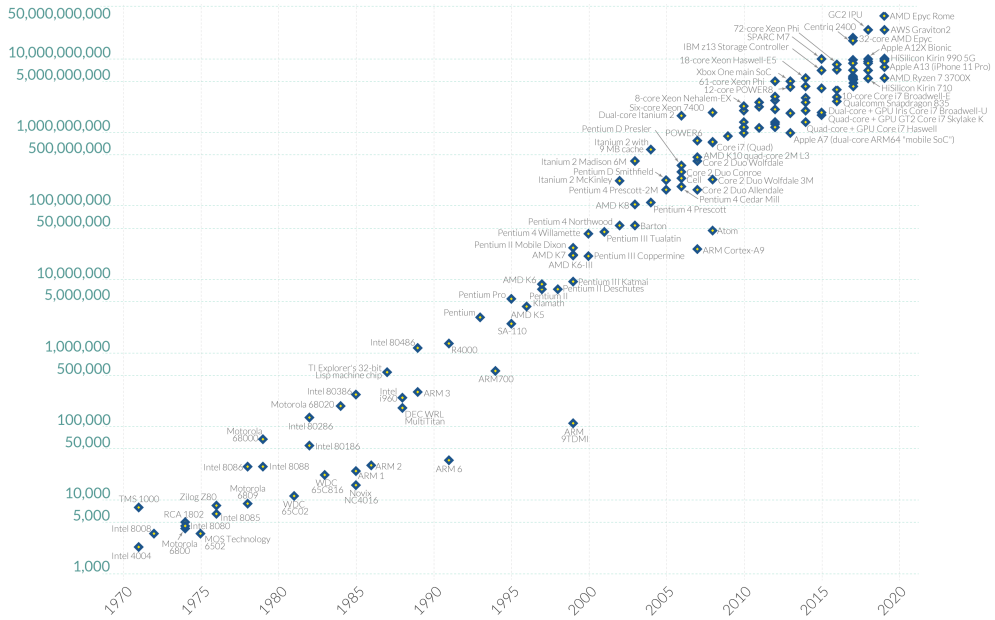


Figure 1.1: Transistor count of some microchips in the years they were first introduced. [10]

the increasing short-channel effects that involve the transistors as their size gets reduced: this is the reason why different technologic approaches have emerged in the last years.

To obtain even more efficient and reliable systems, two main different approaches have been considered so far:

- **More than Moore**: exploiting the existing CMOS technology features to obtain an heterogeneous integration of devices belonging to different domains (for instance RF and analog devices, photonic devices, High Voltage devices or sensors) on the same SoC (System-on-Chip), by interconnecting them.

- **Beyond CMOS**: a cathegory of technologies which allow to realize a logic similar to the CMOS one and are able to overcome its scaling limit, without exploiting the traditional CMOS paradigm.

These two approaches are the most promising perspectives for what concerns the future applications of ICs.

This thesis project focuses on the Beyond CMOS approach, in particular on *Field Coupled Nanocomputing* based on the molecular implementation of Quantum Dot Cellular Automata.

1.3 Field-Coupled Nanocomputing

Field-Coupled Nanocomputing (FCN) is one of the most promising technologies for what concerns the beyond CMOS trend. [4] The main difference with CMOS is that standard logic gates realized with transistors encode binary information following a current-based approach, dragging current from the output to ground to pull down output voltage and from the power supply toward the output to carry the voltage to high level, while FCN employs electrostatic fields and the consequential arrangement of charges to determine and propagate the binary information.
One of the possible architectural implementations on which FCN is based is **Quantum-dot Cellular Automata (QCA)**: in particular, this thesis project focuses on its molecular implementation.

1.3.1 Quantum-dot Cellular Automata (QCA)

QCA is a transistorless approach which exploits electrostatic interactions between Quantum Dots (QDs) organized in cells, in which charge transport is not implied. [11] A simple possible realization of a basic QCA cell would be represented by two series-connected couples of dots, as depicted in Figure 1.2.

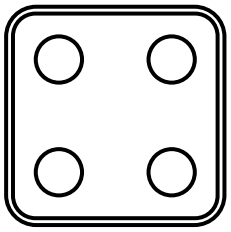
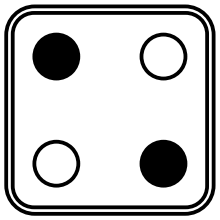


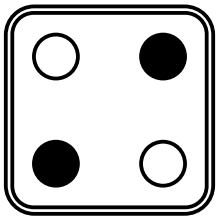
Figure 1.2: Unbiased QCA cell: each circle represents a QD.[6]

In such a configuration, if the cell does not receive any external bias, it stays in an equilibrium state in which no charge transport happens. On the other hand, if the cell is biased, due to Coulomb repulsion the excess electrons (one for each couple of QDs) are forced to opposite corners of the four-dot system, since the QDs at the edges of the cell represent low potential zones where the charges are more likely to stay when influenced by an external electric field. The result is that logic states can be encoded no longer as voltages, but rather by the positions of individual electrons: one configuration encodes a '0'-logic state (Figure 1.3a) and the complementary one encodes a '1'-logic state (Figure 1.3b).
The advantages brought by this implementation are several:

- it gives the possibility to achieve a low-power implementation; [11] [6]
- higher working frequencies can be reached; [11]
- integration is improved, since the system can be scaled at molecular level; [6]
- the mechanism is valid also at room temperature. [11] [6]



(a) QCA cell with logic '0'. [6]

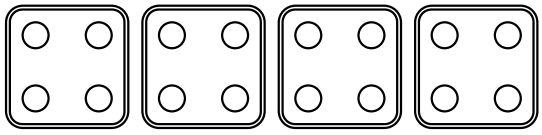


(b) QCA cell with logic '1'. [6]

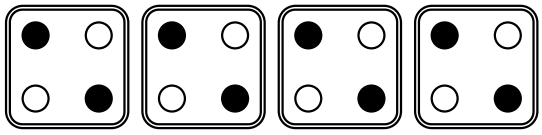
Figure 1.3: Possible logic configurations of a QCA cell. The full dots represent a zone where the charge aggregates.

1.3.2 Realization of basic circuits

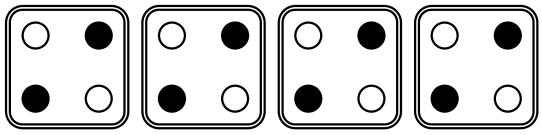
The main idea on which QCA relies consists of creating pipelined-like structures of equal QCA cells, so that for each cell the electrostatic field induced by charges reorganization can determine how the charges of the next QCA will configure, in order to propagate logic information from a starting point to a selected part of the layout. [2]
Basic cells can be rearranged in different ways, depending on the logic function that one wants to synthesize. The simplest circuit that can be built is a wire, composed of a certain number of series-connected cells, shown in Figure 1.4: its aim is just to propagate the logic value generated by an input bias from the first cell to the last.



(a) Unbiased QCA bus structure.



(b) QCA bus structure with logic '0'.



(c) QCA bus structure with logic '1'.

Figure 1.4: Possible configurations of a QCA bus structure. [7]

Another structure that can be obtained with the same method is the Majority Voter which, as the one realized in standard CMOS technology, gives as a result the logic configuration of its central cell (that works as a "conveyer" for the three inputs of the circuit): its truth table is depicted in Table 1.1 and its structure is the one shown in Figure 1.5.
A further structure that can be created is the Inverter. In this case, similarly to a standard

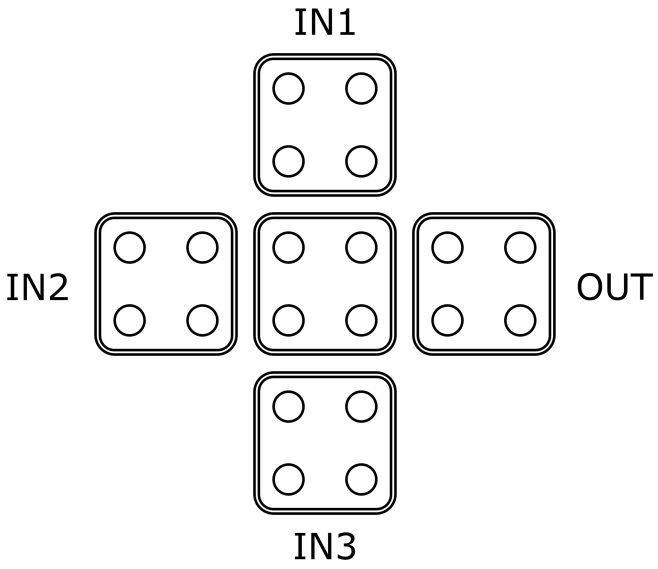


Figure 1.5: QCA representation of the Majority Voter. [7]

Table 1.1: Truth table of the Majority Voter.

IN1	IN2	IN3	OUT
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

CMOS inverter, its aim is to give as output the logic configuration opposite to the input one: the organization of cells depicted in Figure 1.6 allows to obtain this result.

1.3.3 Molecular implementation

One of the possible applications of QDs in QCA is represented by the molecular implementation, in which every couple of QDs is realised exploiting a molecule like, for instance, *bis-ferrocene*. [8]
Bis-ferrocene has a symmetrical structure, shown in Figure 1.7a, composed of:

- two ferrocenes $Fe(C_5H_5)_2$, which are responsible for the logic '0' and logic '1' state encoding. In the graphic representation used until now, each circle into a cell stands for a single ferrocene;

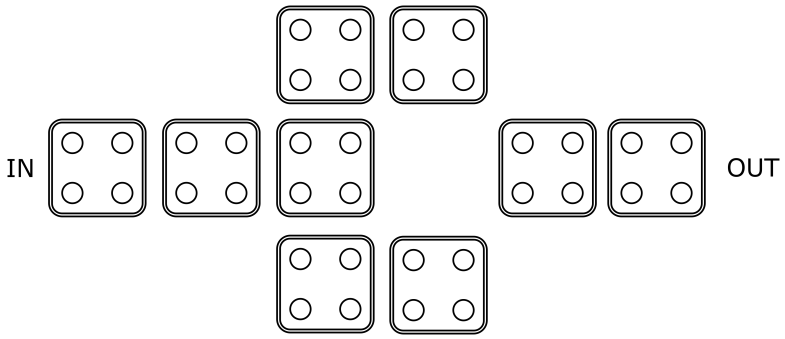
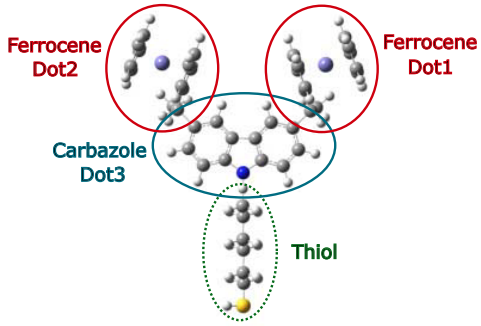
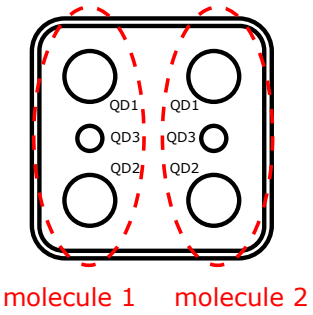


Figure 1.6: MolQCA representation of an Inverter. [2]

- a Carbazole group $C_{12}H_9N$, whose aim is both to allow the connection between the two ferrocenes and the Thiol group and to encode a further logic state;
- a Thiol group, which permits the anchoring of the carbazole group to the substrate surface, usually realized with gold.



(a) Structure of the bisferrocene molecule. [8][2]



(b) QCA structure of bisferrocene. [7]

Each QD in the molecule works as a redox centre, since it is seen by charges as a low potential zone where their aggregation can happen. The working principle is the following: depending on which redox centre the charges mainly aggregate into, a different logic state is represented. As a consequence, in each molecule there are three main QDs which correspond to three distinct logic states, as depicted in Figure 1.7b. If the free charges are localized along one of the two diagonals, either a logic '0' or a logic '1' is represented; if, instead, the charges are forced into the central dots, a 'NULL' logic state (whose importance will be further explained) is represented.
With this implementation, a molecular QCA wire as the one introduced in 1.3.2 would be realized by aligning more than one bis-ferrocene molecules as depicted in Figure 1.8.

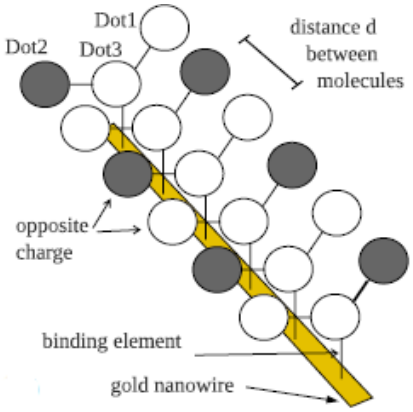
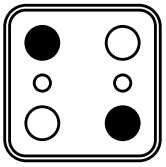


Figure 1.8: Molecular QCA wire of aligned bisferrocene molecules on a gold substrate. [8]

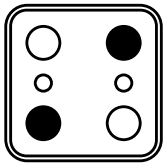
1.3.4 Introducing clocked systems

Considering a pipelined system of QCAs, the electrostatic field induced by a cell can influence the next one, forcing the charges to move to a certain configuration. By exploiting this mechanism, information can propagate from input to output spontaneously, with a certain delay. Since Coulomb interaction is extremely fast this delay is limited, however without an external signal the information cannot propagate correctly for long distances (i.e. in complex circuits), because after a certain distance a conflict between cells is likely to happen and information might get lost: this is the reason why the central QD introduced before can be exploited in a clocked system, which ensures that information is carried on correctly by dividing the circuit in clock zones or *phases*.
Such clock ought to be a "vertical clock", which can be used to determine the logic state for a molecule since:

- it can push the charges either toward QD1 or QD2, bringing the QCA to an APPLIED state;



(a) Bisferrocene QCA cell in APPLIED state with logic '0'. [6]



(b) Bisferrocene QCA cell in APPLIED state with logic '1'. [6]

- it can attract the charges toward QD3, thus bringing the QCA to a NULL or RESET state.

To visualize better the division in phases of a QCA circuit, one can consider the structure depicted in Figure 1.12, in which an eight cell wire is divided in four phases, each composed

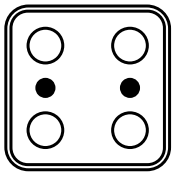


Figure 1.10: Bisferrocene QCA cell in 'NULL' logic state. [6]

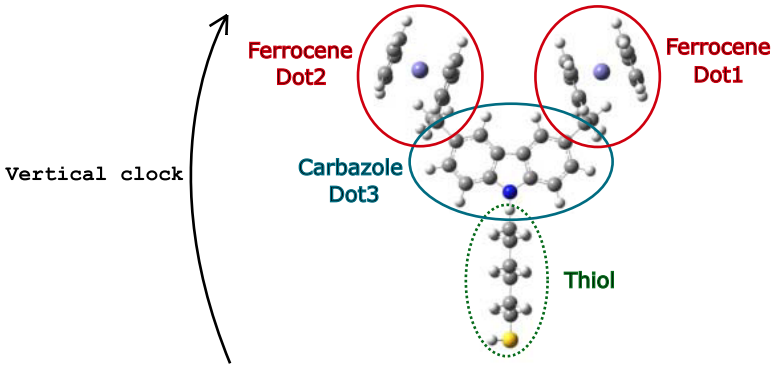


Figure 1.11: Vertical clock electric field applied to the bisferrocene molecule. [2]

of two QCA cells of two molecules each. The same vertical clock is applied to all the QCAs of the same phase.
Vertical clock is not a constant value, but instead varies periodically: one can define a clock cycle of four steps which is repeated for a certain number of times, depending on how much information has to be propagated and on the length of the circuit. The four possible clock states are:

- **Switch**: from an initial RESET state, the applied vertical clock pushes the charges toward QD1 and QD2 according to the propagated information, that is either a logic '0' or a logic '1';
- **Hold**: the same clock remains applied until the QCAs have assumed a stable logic state;
- **Release**: the clock is released toward RESET state and the charges move from QD1/QD2 to QD3;
- **Reset**: charges remain forced to aggregate in QD3 until new information to propagate gets to the pipeline.

Figure 1.14 shows an example of a simple 4-phase QCA wire in which a logic '1' is transmitted from input to output. In a pipeline-fashioned way, every cell starts with a RESET state at T_0, then the first one turns to APPLIED state in T_1 while the others are still kept in RESET and, in T_2, it is forced to keep the same value to influence the following cell,

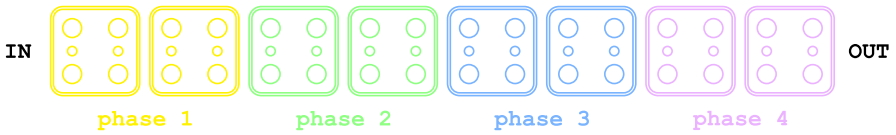


Figure 1.12: Clocked molecular wire divided in 4 phases, each indicated with a different colour. [2]

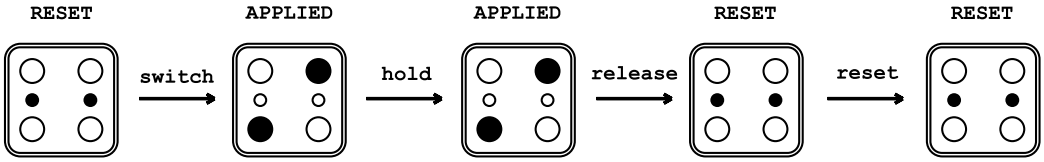


Figure 1.13: Scheme of a clock cycle referring to a single cell.

which assumes the same value; at T_3 it is put to a RESET state again, and the procedure is repeated for the following cells until the end.
It has to be remarked that as in standard digital circuits, the timing with which such operation is executed is crucial: a QCA must be put into a RESET state before being able to receive new information and start a new cycle. This timing constraint ensures the stability of the considered phase.

1.3.5 Improving the performances through *bi-stability*

The working principle illustrated until now can be further improved by achieving *bi-stability*. In layouts similar to the ones described above, the molecules that have no more than one adjacent molecule can suffer border effects, since the electrostatic interaction is stronger if they are in between two stable states. That is the reason why central molecules are more stable if compared to the ones closer to the sides.
In order to achieve bi-stability it is sufficient to implement the layouts by using two adjacent lines instead of one, as shown in Figure 1.15.
The two wires work in the same way, with the advantage that the majority of the molecules of the circuit have three neighboring ones: this guarantees a better reliability due to a more distinct charge separation, which translates into a better delivery of the signal over the layout.

1.4 SCERPA

One of the most innovative methodologies that aim to evaluate the propagation of information in molecular FCN circuits is **SCERPA (Self-Consistent ElectRostatic Potential Algorithm)**, which takes into account the effective physical properties of each molecule and, at the same time, considers it as an electronic device with its own properties. [1] It is an iterative procedure, divided in steps, that exploits the characterisation

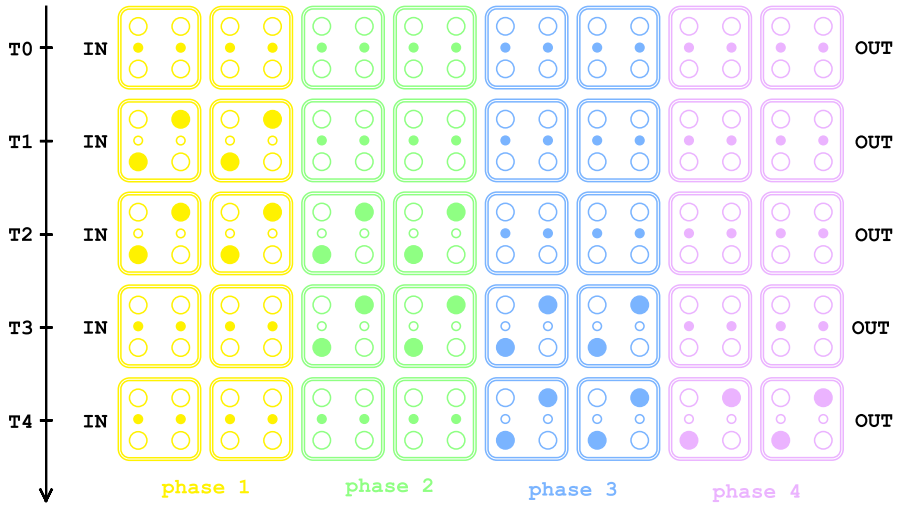


Figure 1.14: Propagation of a logic signal throughout different time steps. Each n-th repetition of the bus is a time step, labeled with Tn.

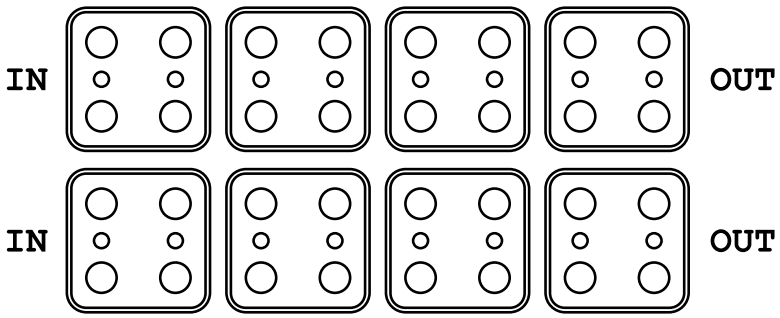


Figure 1.15: QCA bus structure exploiting bi-stability.

of molecules under electric clock fields to enable the simulation of clocked devices and provides a computational cost significantly lower than ab initio simulations[1].
The first step of the algorithm consists of characterizing a single Bisferrocene molecule by studying the response of a *Molecule Under Test* (MUT) to the electric field generated by a second molecule (the *driver*), positioned at a certain distance d, which is modeled as three aggregated charges[2].

[1] *Ab initio* calculations are extremely powerful tools, based on quantum mechanics, to study the behaviour of single molecules or the interaction between different molecules. [1] In the case of molecular circuits, they can be used exploiting point charges in order to emulate external electric stimuli (the *driver* molecules). Despite their accuracy, they are very computationally expensive, which means that the analysis of more complex and bigger circuits cannot be executed entirely with ab initio calculations.

[2] Bisferrocene's QDs can be approximated as aggregated charges, assuming that the charges in each QD are accumulated in an infinitesimal point of space for simplicity. [9]

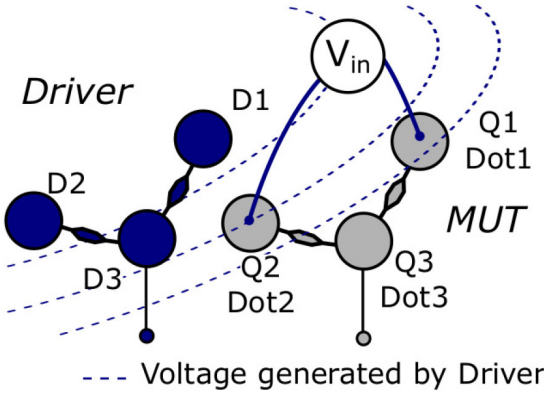


Figure 1.16: Representation of the effect of a driver molecule on a MUT. [1]

In this starting phase, an ab initio calculation gives high accuracy when having to evaluate the charge distribution of a molecule. The aggregated charge per QD of the j-th molecule of a circuit is:

$$Q_n^j = \sum_{i=0}^{N} q_{n,i}^j \tag{1.4}$$

where n refers to the n-th molecule QD while i represents the point charge composing the aggregate one.
The second step of the algorithm consists of modeling the electrostatic behaviour of a molecule, exploiting the evaluated aggregated charges. Given the set of aggregated charges $\{Q_1^j, Q_2^j, ..., Q_{N_{AC}}^j\}$, and associating a position in space to each of them, the voltage generated by a single molecule j in a generic point r of space can be evaluated as:

$$V_j(r) = \frac{1}{4\pi\epsilon_0} \sum_{n=1}^{N_{AC}} \frac{Q_n^j}{d(r_n^j, r)} \tag{1.5}$$

The third step aims to study the intermolecular interaction. The main concept is that, as for what happened with the driver and the first molecule of the circuit, once the generic molecule i receives the influence of molecule j, molecule i is considered as a driver that generates its feedback effect back to all the other molecules, molecule j included. As a consequence, considering the N molecules of a wire[3], the input voltage $V_{in,i}$ of molecule i is evaluated by considering the effects of the driver $V_{D,i}$ and of all the other molecules $V_{j,i}$.
The voltage generated by any charge in the layout, together with the transcharacteristic of the complex molecule used in the construction, enables for an information propagation model:

[3]The description of the process in this case considers a wire for simplicity, but the procedure is valid for every circuit built with this paradigm

$$V_{in,i} = V_{D,i} + \sum_{j=1, j\neq i}^{N_{AC}} V_{j,i}(V_{in,j}, E_{clk}) \tag{1.6}$$

where:

- $V_{in,i}$ is the input voltage of the i-th molecule;
- $V_{D,i}$ is the voltage imposed by external drivers;
- $V_{j,i}$ is the voltage generated by all the other molecules in the layout and depends on their input voltages and the vertical clock;
- E_{clk} is the value of the vertical clock field, known *a priori* and given in input by the user.

Since V_j depends on molecule j aggregated charges, which depend on $V_{in,j}$, the process is represented by a nonlinear system. SCERPA implementation solves this problem first by supposing an initial voltage on all the molecules (i.e. $\{V^0_{in,1}, ..., V^0_{in,N}\}$) then by iteratively evaluating Equation 1.4 to determine the approximated solution of the nonlinear system in the form:

$$V^k_{in,i} = F_i(V^{k-1}_{in,1}, ..., V^{k-1}_{in,i-1}, V^{k-1}_{in,i+1}, ..., V^{k-1}_{in,N}) \tag{1.7}$$

Each iteration of the algorithm is known as *SCERPA step* and depends on the previous one, where F_i denotes the function in Equation 1.4.

1.4.1 SCERPA implementation in MATLAB

The SCERPA MATLAB script is a tool capable of evaluating both graphically and numerically the behaviour of a MolFCN circuit but, despite being faster than a whole ab initio simulation, it can be time demanding. Its time overhead depends on the complexity of the circuit (i.e. the number of molecules), the options selected for SCERPA and the number of steps. Each SCERPA step corresponds to a vertical clock value, which means that the computational time is proportional to the number of clock cycles that have to be simulated.
The implementation of clock states in a cycle in SCERPA is discretized in little steps, in order to simulate a behaviour closer to reality where the switch from one state to another is not instantaneous: this concept is known as *adiabatic switching*. In order to achieve a sufficiently good simulation accuracy, the algorithm has to simulate the intermediate values with additional SCERPA steps. The finer this discretization is, the more accurate will be the simulation, but also the more computational time demanding.

1.5 Building complex circuits

The possibility of obtaining distinguished logic states and propagating binary information in a pipelined way is the starting point from which more complex circuits can be built. Following the De Morgan's theorems, from AND and OR gates and inverters it is possible to synthesize any logic function and this can be applied also to MolFCN circuits.
By looking at the Majority Voter's truth table (Table 1.1) one can notice that, by fixing one of the inputs to a logic '0' or to a logic '1', the result corresponds, respectively, to an AND function and an OR function which has as inputs the remaining two inputs of the Majority Voter.

Table 1.2: Truth tables of AND and OR gates, derived from the Majority Voter's table.

(a) Truth table of the AND gate (IN1 is fixed to logic '0').

IN1	IN2	IN3	OUT
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1

(b) Truth table of the OR gate (IN1 is fixed to logic '1').

IN1	IN2	IN3	OUT
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Once an AND gate or an OR gate have been obtained from a MV, it is possible to exploit their logic function to build bigger circuits. For instance, it is sufficient to connect a MV with one of the inputs fixed to '0' or '1' to an Inverter to obtain a NAND or a NOR gate, respectively. The importance of this implementation lies in the possibility of a bottom-up creation of bigger digital circuits.

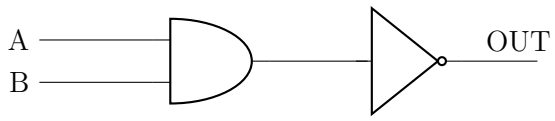


Figure 1.17: Logic representation of the NAND gate, obtained by connecting a NOT gate to an AND gate.

1.6 Beyond SCERPA implementation

SCERPA implementation allows to perform an extremely accurate analysis of MolFCN circuits, starting from the molecules they are made of; it is particularly suited for simple layouts, but it can be applied also to complex and bigger ones. However, for what concerns the more complex systems on which modern digital design flow is based, the time overhead required might not be suitable.
A possible solution to the problem would be to extrapolate, for each circuit, a series of libraries which contain its complete characterization from an electronic and physical point

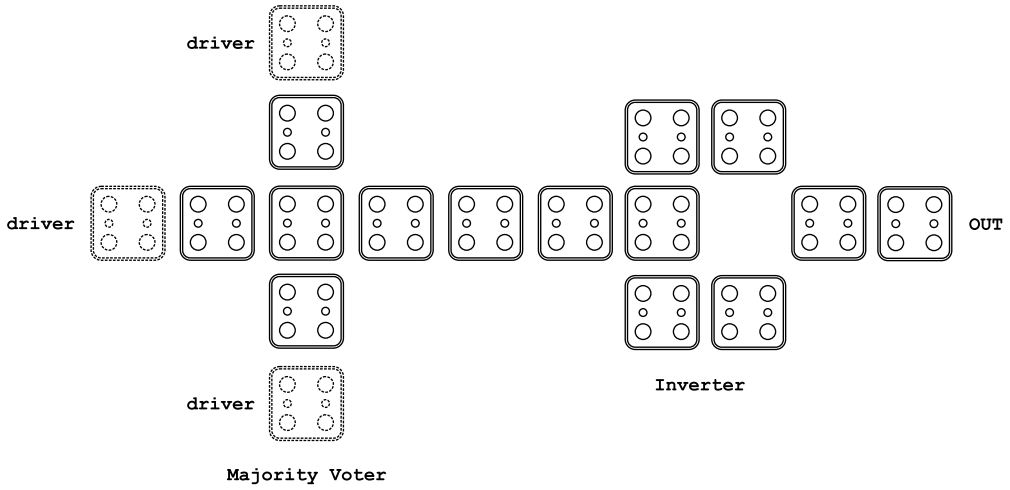


Figure 1.18: MolFCN representation of a NAND gate, with the input of the Inverter connected to the output of the MV.

of view, exploiting the results of its SCERPA simulation. In a more general perspective, once knowing the behaviour of the basic cells of which a complex circuit is composed, instead of simulating the whole circuit with the same procedure it is significantly less time demanding to characterize them and evaluating the system behaviour by fetching the tables obtained with the characterization as libraries[4].
This procedure would still require to simulate small circuits on SCERPA, in order to obtain the data needed for the characterization, but each of them would be simulated only once.
The aim of this thesis project is to employ a block-based circuit design within the MolFCN paradigm, that relies on the characterization of simple circuits through a MATLAB script, in order to build more complex structures and then evaluate their transcharacteristic without paying their full SCERPA simulation overhead.
In particular, a key focus of the project is based on the comparison between the result obtained with the SCERPA method alone and the one obtained with the characterization. Figure 1.19 depicts the workflow followed in the project.

[4]For instance, in the circuit of Figure 1.18, it would be less time demanding to characterize separately the MV and the Inverter, and then evaluate the final result by putting together their transcharacteristics.

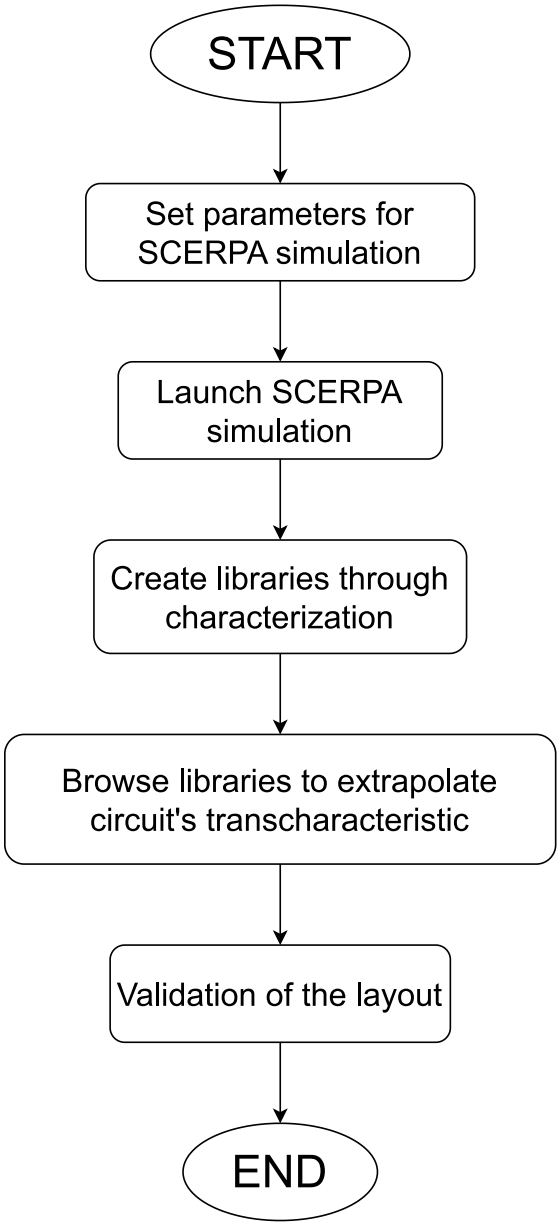


Figure 1.19: General workflow of the project.

Part II

Characterization Process

Chapter 2

From layout definition to SCERPA simulation

2.1 Layout definition in MagCAD

The `MagCAD` software is a very useful graphical layout editor that can be employed to build graphically any MolFCN layout from scratch. [3] After defining the type of molecule to work with (in this case, the *bisferrocene*) and the number of clock phases, the layout can be obtained by assembling QCA blocks by means of a GUI and then assigning inputs and outputs to the circuit. At the end of the creation of the circuit, a `.qll` file is generated. An example of a simple circuit built with MagCAD is shown in Figure 2.1.

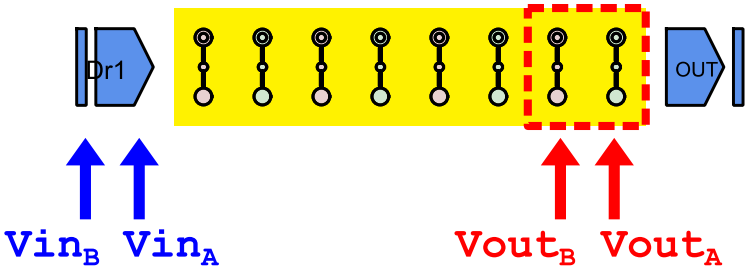


Figure 2.1: Layout of a bus made of 4 cells (i.e. 8 molecules), each one belonging to the same phase.

This kind of representation requires a definition of *input voltage* V_{in} and *output voltage* V_{out} that remains consistent also where a similar layout is connected in cascade to the one under test. The structure represented in Figure 2.1 actually contains three kinds of objects that must be defined:

- **DrX labels**: each of these refer to the X-th driver, where drivers are molecules with

the same shape of a standard QCA that model the externally applied voltages (as introduced in section 1.4). These can be modeled in two ways:

  - **SingleMolDriverMode**: the driver label is equivalent to a unique molecule, i.e. half a QCA;
  - **DoubleMolDriverMode**: the driver label is equivalent to a couple of adjacent molecules, i.e. a whole QCA;

- **QCAs**: cells that represent the body of the structure, allowing the information to propagate. In this notation, molecules with the same colour belong to the same clock phase;

- **OUT labels**: layout outputs, to identify where the layout finishes and, as a consequence, the output direction. They are particularly useful since, within SCERPA, each output corresponds to a dummy molecule (or a couple of dummy molecules) which is positioned after the last molecule of the layout: considering a pipeline structure where the output of a block is connected to the input of another block, this mechanism allows to study information propagation between different circuits.

The DoubleMolDriverMode paradigm is the one used in this development of the whole characterization method. For what concerns the body of the layout, it will always be developed in bus mode to guarantee bi-stability (subsection 1.3.5); considering the example of Figure 2.1, its equivalent bus layout would be the one in Figure 2.2.

In order to reach definition compliance, if an input is build by two molecules, each layout output must rely on the same number of molecules: the bus paradigm doubles the molecules, the drivers and the output labels. The V_{in} and V_{out} definitions described until this point can be extended also for bus cases.

A generic output can face up to three different directions, since MolFCN is a planar technology. A circuit can have outputs that are oriented horizontally (with angle = 0°) like the one depicted in Figure 2.2, but also vertically (either with angle = 90° or 270°), meaning that the signal propagates upwards or downwards. Figure 2.3 and Figure 2.4 show an example of these two cases.

A standard notation has to be introduced in order to identify the voltages on the different drivers and output molecules, since they are going to be associated to distinct values. As a general rule, the voltage whose name ends with **A** is always the right one while the **B** is always the left one; in DoubleMolDriverMode, the same applies for **C** and **D**, respectively. Furthermore, if the 4 molecules are aligned horizontally (as the outputs of Figure 2.3) the order of the letters is **DCBA**, while if there are 2 molecules up and 2 down (as the outputs of Figure 2.2) the upmost couple is represented by **BA**. This general principle is valid for both drivers and outputs, whose name starts respectively with *Vin* and *Vout*.

MolFCN circuits can also have outputs in 2 different directions, as in Figure 2.5: the relative notation for this case adds a subscript 'u' or 'd' if the orientation of the output is respectively upwards or downwards.

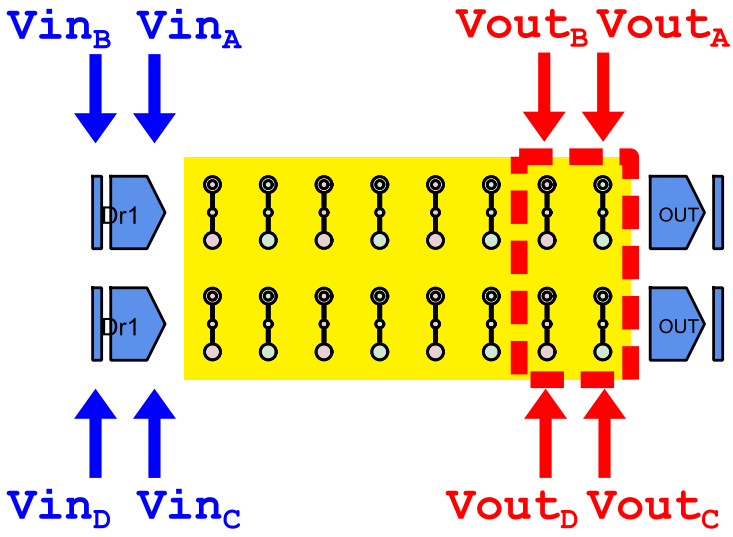


Figure 2.2: Representation of a 4-cell bus in bus mode.

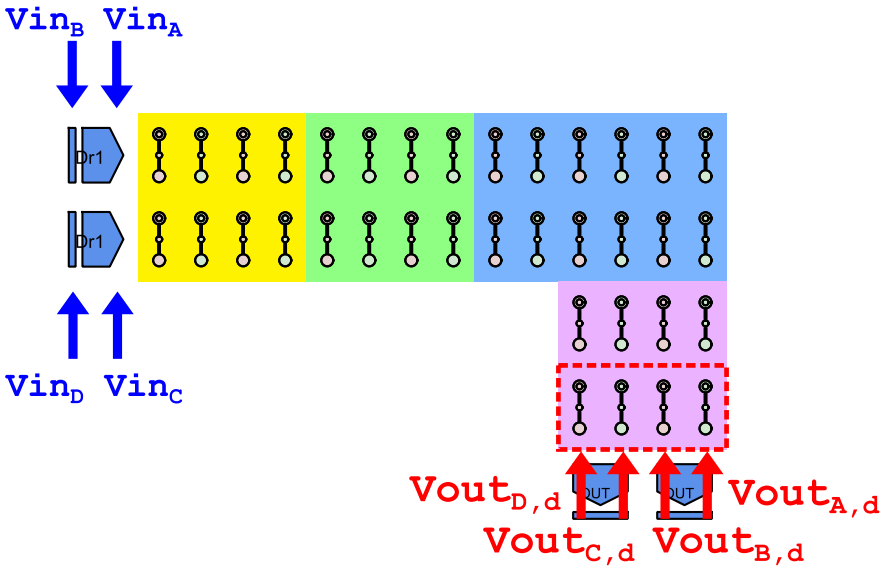


Figure 2.3: 4-phases 'L-connection' block with downward oriented outputs.

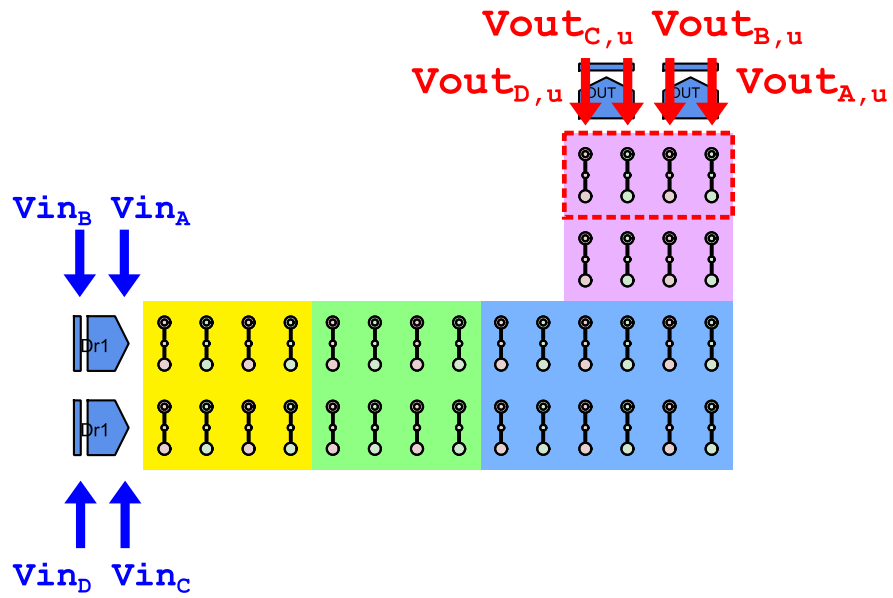


Figure 2.4: 4-phases 'L-connection' block with upward oriented outputs.

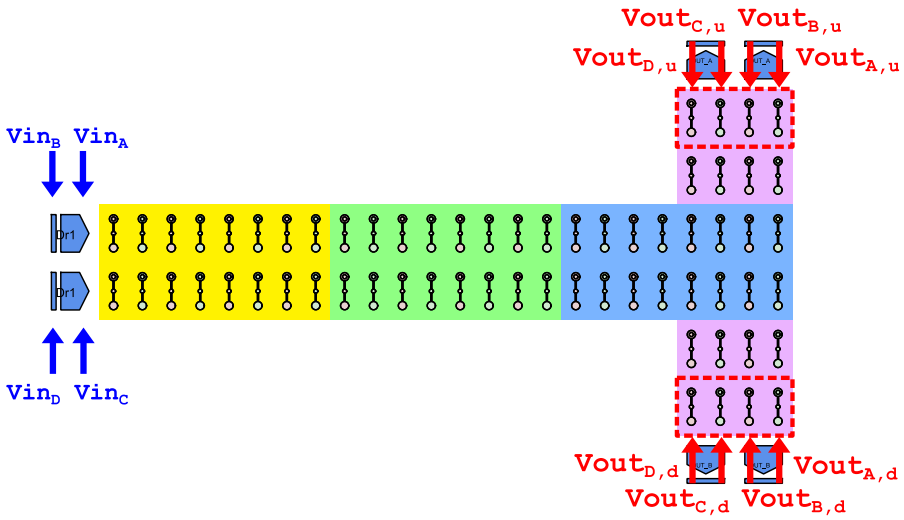


Figure 2.5: Example of a 'T-connection' circuit, where the outputs go both downwards and upwards (having an orientation angle of, respectively, 90° and 270°).

2.2 The *launch.m* script

The *launch.m* script is the MATLAB script that manages every operation related to the whole simulation process. It has the aim of collecting information about the circuit,

inserted by the user, into dedicated data structures in order to be able to run either the SCERPA simulation of the layout or its characterization.
The most important data collected by the script are:

- the circuit layout, taken from the corresponding .qll file generated by MagCAD;
- the external signals involved, that correspond to the driver parameters contained in `driverPara` and the clock signal values;
- the settings related to the SCERPA algorithm, saved in `settings` and `plotSettings`;
- the settings related to the Characterization process, saved in `charSettings`.

After this step, it allows the user to select the operative mode between **Debug Mode** and **User Mode**.

2.2.1 *Debug Mode* and *User Mode*

Debug Mode and *User Mode* are two modes that exploit the same set of data (defined in the *launch.m* script), but give different outputs: the Debug Mode is focused on the creation and the validation of the layout, while the User Mode is focused on creating the library files that will be elaborated further throughout the method[1]. In particular:

- Debug Mode: it is designed to plot the $V_{in} - V_{out}$ characteristic given the input voltages specified in the launch script, since the graphical way is the best one to evaluate the behaviour of a layout[2], but also to save the results of SCERPA simulations in dedicated files.
- User Mode: it is designed to test almost every possible input combination to achieve the complete characterization of the cell, by creating a library with high coverage. The results are stored in `.csv` files in a tabular form, which makes them easier to fetch for the elaboration of such data. An information file is produced together with the tabular file, in order to store additional data as the latency of the circuit, the clock signal voltage and so on. This thesis project focuses mainly on the features of the User Mode, since the characterization tool takes the results of SCERPA simulations as a starting point for its development.

The choice for the Debug or User mode is provided by the user by setting the flags `DebugMode` and `LibEvaluation` to 0 or 1 depending on the necessity. Table 2.1 shows the different possibilities of operation.

[1]The library files obtained within the User Mode are `.csv` files that contain the combinations of inputs and outputs of the specific circuit that has been simulated in Debug Mode.

[2]When running SCERPA, there is the possibility to select between different modes according to the desired result. The most complete one is the `'generateLaunchView'` mode, which, as the name suggests, allows to generate library files that describe the transcharacteristic of the circuit but also to generate graphs that illustrate its logic distribution, the potential and so on.

Table 2.1: DebugMode and LibEvaluation flags operation modes.

DebugMode	LibEvaluation	Operation
0	0	Launch the Characterization Tool
0	1	Behaviour evaluation through libraries
1	0	Launch SCERPA simulation
1	1	Test every input combination fetch on library

2.3 Creation of drivers and clock values

Among the most relevant data imported from launch.m there are the values of the drivers and of the vertical clock of the circuit: in a clocked system as the one that is being considered, input values have to be created coherently with the clock so that information can reach the end of the circuit properly without being lost or corrupted.
Driver values are saved on MATLAB in cell arrays of which every cell contains the voltage assigned to the driver at the n-th time step of the simulation; they can be assigned through 4 labels: **'0'**, **'1'**, **'sweep'** and **'not_sweep'**, where:

- **'0'** means that the cell array related to the driver contains the voltage assigned to logic '0' repeated for every cell;
- **'1'** means that the cell array related to the driver contains the voltage assigned to logic '1' repeated for every cell;
- **'sweep'** means that the values contained at the beginning of the cell array are obtained by a linear sweep of numbers from logic '0' to logic '1';
- **'not_sweep'** means that the values contained at the beginning of the cell array are obtained by a linear sweep of numbers from logic '1' to logic '0'.

Whichever the value assigned to a driver is, it has to be guaranteed that the length of its array is coherent with the number of steps of the whole simulation, to guarantee that the driver is valid from its beginning to its end.
On the other hand, the creation of the vertical clock is straightforward and does not depend on the drivers: given a clock cycle divided in 4 phases as the one modeled in subsection 1.3.4, it is created in MATLAB by assembling the arrays of the 4 phases as shown in 2.3.

```
clock_low = -2;
clock_high = +2;
clock_step = 5;

% assembling the clock cycle
pSwitch =  linspace(clock_low, clock_high, clock_step);
pHold =   linspace(clock_high, clock_high, clock_step);
pRelease =  linspace(clock_high, clock_low, clock_step);
pReset =   linspace(clock_low, clock_low, clock_step);
pCycle = [pSwitch pHold pRelease pReset];
```

Listing 2.1: Creating the vertical clock signal.

The clock step indicates the number of clock values between its low and high logic level: the higher it is, the less abrupt is the transition from one state to another (but also the longer the simulation).

2.3.1 *buildDriver.m*

The buildDriver.m function receives data about the names of the drivers and their logic values, the clock signal features (cycle length, step) and the layout of the circuit and returns the structure valuesDr, in which each row contains the name of the driver followed by its values for every clock step of the simulation. With a number of N drivers, valuesDr has N $\times$ 2 rows since also complementary drivers are considered[3].

Each row of valuesDr is built from scratch assembling three smaller arrays, depending on the kind of label assigned to the driver. The first array (of just one cell) contains the name of the driver, while the second array contains the values given to the driver according to the label, either '0', '1', 'sweep' or 'not_sweep'; the third and last array is added as a "filler", which is needed for compliance with the Reset cycle of the clock and in layouts with more than one repetition of clock phases. The reason is that, as highlighted in section 2.3, the driver array must be long at least as the number of time steps of the whole simulation: in structures with N clock phase repetitions, the output that corresponds to a certain input can be read only after N clock cycles, but the data generated in the first two sub-arrays covers just 1 clock cycle. The "filler" covers the remaining N-1 clock phases and it consists of repeating the value of the last cell of the second array for N-1 cycles.

The rows related to complementary drivers are obtained by just inverting the signs of the rows of the corresponding drivers.

Table 2.2 shows an example of a possible form of valuesDr. An important detail that has to be highlighted is that, in case the parameter $N_{sweepsteps}$ is different from 1, each value in the table is replied $\times$ $N_{sweepsteps}$ times instead of just once.

[3]Since the DoubleMolDriverMode is considered.

Table 2.2: Example of valuesDr with two drivers Dr1 and Dr2, respectively with 'sweep' and '1' input. Arrays are separated by double vertical lines.

Dr1	1.5	0.5	-0.5	-1.5	..	-1.5	-1.5	-1.5	-1.5	..
$Dr1_c$	-1.5	-0.5	0.5	1.5	..	1.5	1.5	1.5	1.5	..
Dr2	-1.5	-1.5	-1.5	-1.5	..	-1.5	-1.5	-1.5	-1.5	..
$Dr2_c$	1.5	1.5	1.5	1.5	..	1.5	1.5	1.5	1.5	..

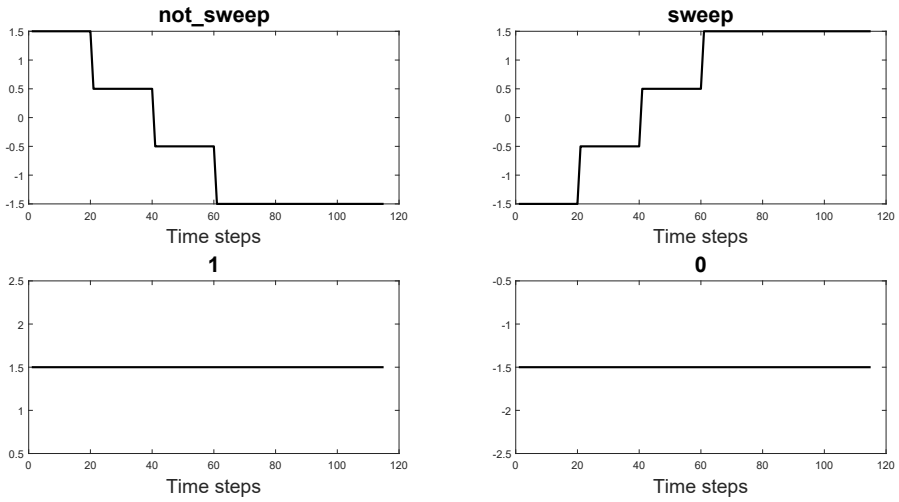


Figure 2.6: Waveforms of the drivers in the 4 cases, in case of only 1 repetition of phases.

2.3.2 *buildClock.m*

Similarly with respect to buildDriver.m, buildClock.m has the role of returning the matrix `stack_phase`, which contains the values of the vertical clock. In particular, *stack_phase* has a number of rows equal to the number of clock phases, meaning that the same clock is applied to equal phases of the circuit in case of a repetition of these. The first row of the matrix, representing phase 1 of the circuit, is built by repeating the clock cycle (stored in the variable `pCycle` received from launch.m) N times and then adding a filler array at the end of it, in order to guarantee the last reset state to the molecules.
Equation 2.3.2 and Equation 2.3.2 show how N and filler are evaluated: $N_{sweepsteps}$ is a simulation parameter, phaserep is the number of times the clock phases are repeated, pReset is an array containing the clock values related to the reset phase.

$$N = (N_{sweepsteps} \times N_{combinations}) + phaserep - 1 \tag{2.1}$$

$$filler = pReset \times (N_{clockregions} - 1) \tag{2.2}$$

After the first row of stack_phase has been built, the other phases are obtained just by circularly shifting its content of a quantity k such that:

$$k = length(pReset) \times (i - 1) \tag{2.3}$$

where i = 1 for the second row, i = 2 for the third row and so on, until i = $N_{clockregions}$ for the last row. The clock waveforms that correspond to the rows of stack_phase of a circuit of 4 phases are depicted in Figure 2.7.

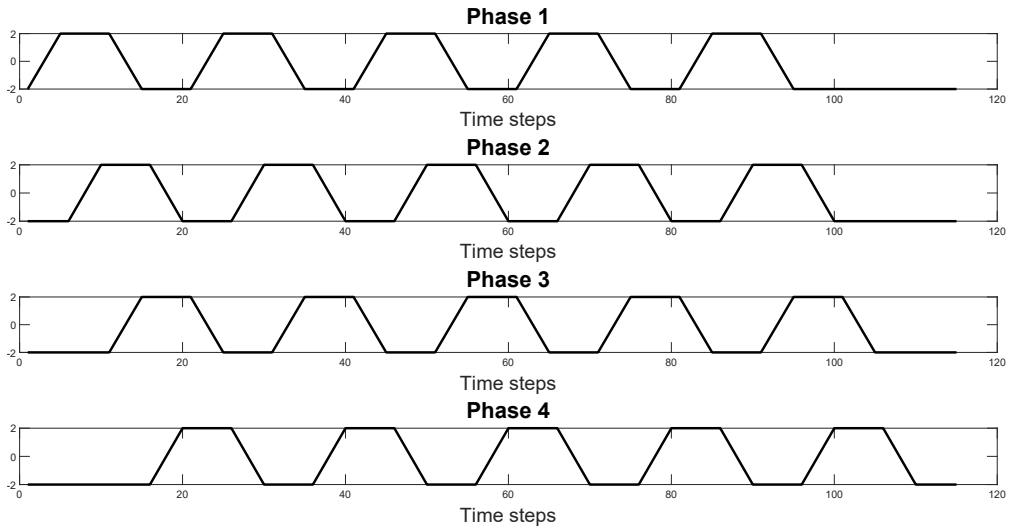


Figure 2.7: Waveforms of the clock signals on each phase of a 4-phase circuit.

Figure 2.8 shows an example of the waveforms of the vertical clock and of the drivers, together with one of the corresponding outputs of the circuit. In this case, the values assigned to DrA and DrB are respectively 'sweep' and 'not_sweep' and, as a consequence, during the HOLD stage of the CK_4 clock phase (i.e. the phase to which the outputs belong) the output corresponds to logic value '1'.

2.3.3 Simulating more than one combination

A further implementation of buildDriver.m consists of building the drivers with more than one combination of inputs, exploiting the pipelined behaviour of the circuit. Also in this case the function returns ValuesDr, which has the same number of rows since the drivers' names are still the same, but an increased number of columns since more values are assigned to the drivers. The struct is built from scratch by assembling different arrays, as in the previous case:

1. The first block of the array is just one cell and contains the names of the drivers and their complementary ones;

2. the second block is composed of the several combinations of inputs: every array of values of the related combination is concatenated in the same order in which the

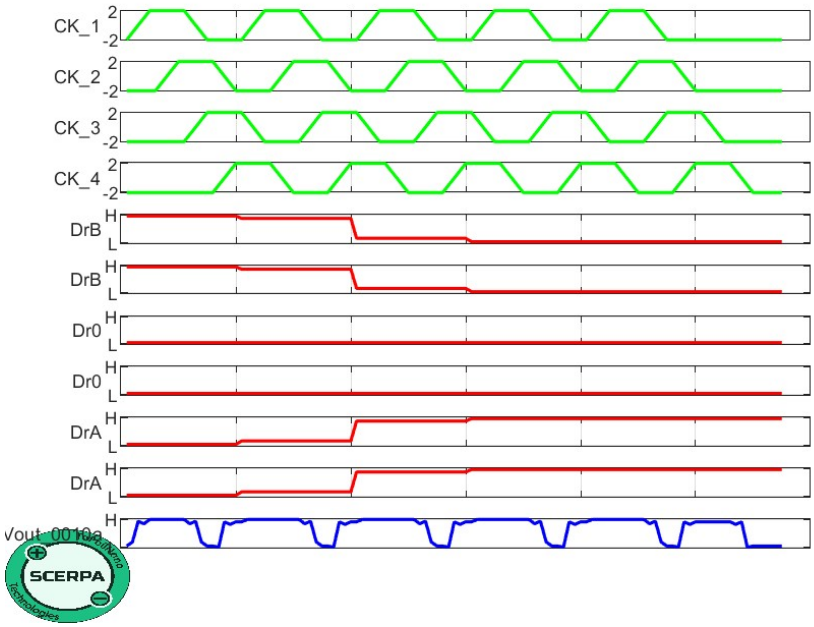


Figure 2.8: Waveforms of drivers, vertical clock and one of the outputs.

inputs are given. For instance, if the inputs are ['0' '1', '1' '1', '0' '0'], the 3 respective arrays of values are concatenated in order;

3. the third block is the same "filler" of the procedure used for just one combination of inputs, that, as explained before, depends on the repetition of phases in the circuit.

For what concerns the code, `driverNames` and `driverModes` contain respectively the names of the drivers and their input logic values: every column of the matrix driverModes contains the values of the corresponding driver in order.

```
driverPara.driverNames = {'Dr1','Dr2','Dr3'};
driverPara.driverModes = {'sweep','1','sweep';'0','1','1'};
```

Listing 2.2: Example of 2 combinations of inputs assigned to the drivers.

Figure 2.9 describes graphically the difference between this method and the one depicted before, highlighting the "pipelined" behaviour: in particular, it refers to what happens for 1 driver. With N drivers, the total overhead is obtained multiplying $\times$N times.
The possibility of simulating more than one combination of inputs is fundamental for circuits like the Majority Voter, for which one combination is not enough to have a complete library of all the values. Since it has 3 input values, that means there are $2^3 = 8$ possible combinations of inputs if considering only the '0' and '1' logic values and $4^3 = 64$ possible combinations of inputs if considering also 'sweep' and 'not_sweep'.
The difference for what concerns the simulation times is shown in the further chapters.

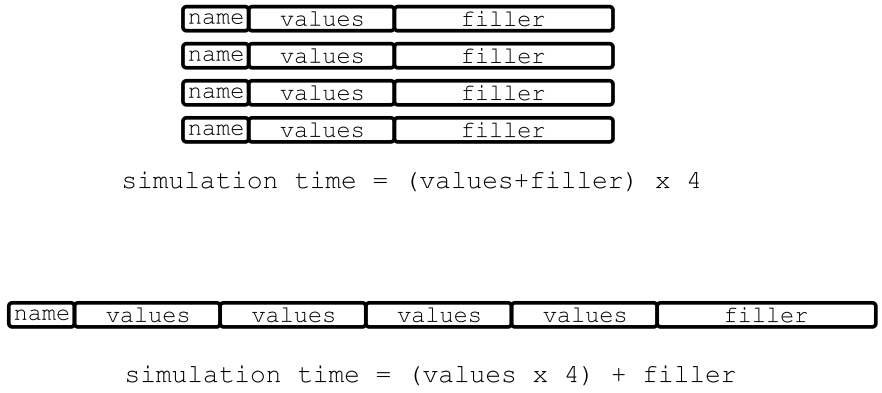


Figure 2.9: Difference between the time overhead of the two approaches.

2.4 Adding an output termination

The concept of bistability can be exploited also for what concerns the outputs: output molecules introduced until now receive information from the molecules on their left, but they are not influenced by any source on their right. For experimental purposes, one solution to this problem could be represented by attaching a further set of molecules to the end of the circuit. This "appendice" added to the circuit, for simplicity, is made by a bus of molecules equal to the ones of which the main circuit is made that has the same length of a clock phase, as shown in Figure 2.10.

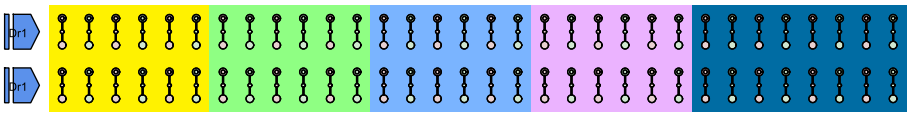


Figure 2.10: Bus wire circuit with a termination added at the end.

To add this termination in the most automated way possible to any circuit, the function `add_termination.m` has been created. Its workflow is the following:

1. It imports the .qll file of the circuit, with the driver names and, in particular, the output names, values and positions;

2. it refreshes the .qll file by adding further QCAs after the end of the circuit, based on:

   - the output angle(s) of the layout (0°, 90° or 270°) to understand the direction on which the QCAs are going to be added;
   - the length of the termination, which can be either a custom one (with customLength flag set by 1 from the user) or a default value, set to 8 molecules;

- the labels of the body molecules, since the names of the new ones are going to be created automatically according to these.
- the presence of a bus-type layout, which implies that, instead of a single line of molecules, a double line has to be created. IN this case, also drivers and outputs have to be doubled accordingly.

3. it refreshes valuesDr, since the drivers have to be coherent with the new length of the circuit;

4. it refreshes stack_phase, since the new circuit has a new clock phase[4].

The function gives as output a modified version of the circuit, saved in a new .qll file obtained by adding the string '_termination' to the original name.
The addition of a termination influences the handling of the circuit's outputs. The instructions introduced until now implied that, apart from the circuit, a set of output molecules were identified, distinguished from the "body" QCAs of the circuit, from which of course the actual output values could be read; that feature was guaranteed since those molecules were the last ones of the layout.
In this version of the circuit, instead, there are no formal "output molecules" as the ones defined before, because it does not represent an effective circuit but just an *ad hoc* modified version to improve stability. The molecules from which the output value can be read are the ones belonging to the first cell(s) of the termination, which are immediately on the right of the last molecules of the body of the structure: this is a consequence of the fact that the true circuit remains the same and the values are read from the same molecules of the version without termination, but this time they are more stable thanks to the fact that the remaining molecules of the terminations are located on their right.

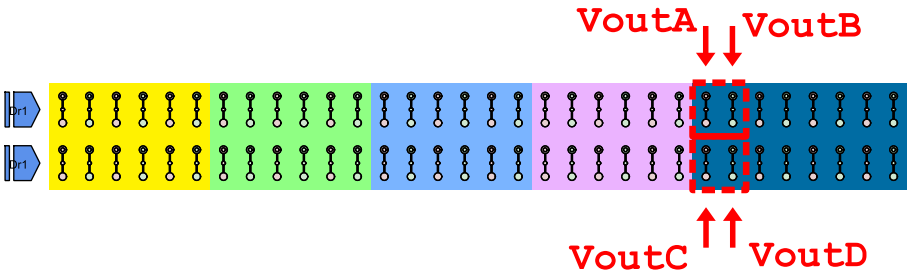


Figure 2.11: Schematic of the layout showing the position of the actual outputs in the case of a bus structure.

The way to read properly the mentioned values is presented further in this presentation.

[4]The termination is being added to the circuit just for simulation purposes. This means that it does not represent a physical "appendice" of the circuit, i.e. when the circuit is combined with other blocks it is not considered for the characterization. It is just useful to study the stability of the outputs in a single basic block.

It has to be remarked that SCERPA, receiving the circuit with the termination, performes a simulation in a way identical to the one regarding the original circuit. The kind of parameters are exactly the same, the only difference stands in the identification of the "output" labels.

Chapter 3

Characterization Tool

3.1 The *characterization.m* script

The MATLAB script *characterization.m* represents the central part of the characterization process. In order to be executed, the flags debugMode and LibEvaluation are set both to 0, since its objectives are not related to the Debug mode and are based on creating new libraries, rather than reading them.
The aim of the script is to return:

- a certain quantity of .csv files (one for each output of the circuit) where each row contains the input voltage values of the circuit and the corresponding output voltage. The name of such files will be derived from the related output, as it will be illustrated further;
- a .txt file named **info.txt** where other important information regarding the circuit is gathered (for instance, the number of clock phases, the latency[1] and the area of the circuit).

The procedure followed by the script can be summarized by the flowchart in Figure 3.1.

[1]By definition, the latency of a system is defined as the time required for an input to become an output in terms of clock cycles. In this particular case, the latency refers to how many times a pattern of clock phases is repeated

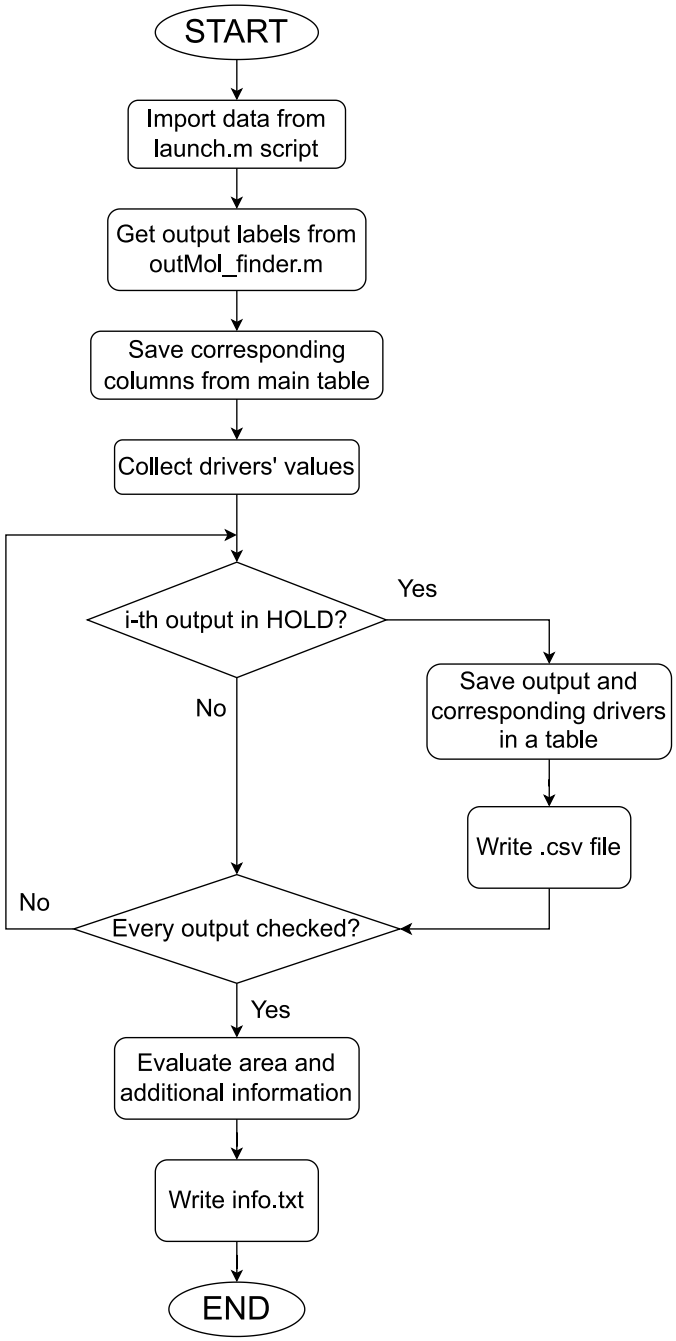


Figure 3.1: Flowchart of the *characterization.m* script.

3.1.1 Paths definition

At its very beginning, the function defines the paths where the outputs are going to be saved: the folder `charResults` is set as the main directory, while a folder with the

characterized circuit's name is created inside it (if it does not already exist) by removing the string '_qll' from the name of its .qll file.

```
% Paths definition
simulation_path = charSettings.out_path;

if ~isfolder(fullfile(BBcharPath,'charResults'))
    % creating main characterization directory
    % (if it doesn't already exist)
    mkdir(fullfile(BBcharPath,charResults));
end
charPath = fullfile(BBcharPath,'charResults');

% defining the name of the directory
% that will contain characterization results
dir_name = erase(file,'.qll');
if ~isfolder(fullfile(charPath,dir_name))
    mkdir(fullfile(charPath,dir_name));
end

 % path of the directory related to the specific circuit
dirPath = fullfile(charPath,dir_name);
```

Listing 3.1: Definition of the paths.

3.1.2 Importing data from launch.m

The script imports the output files obtained by the previously executed SCERPA simulation of the circuit: *Additional_information.txt*, which contains in tabular form information about each molecule of the circuit in different time instants, and *simulation_output.mat*, which is a MAT-file that stores all the variables and data structures which are going to be exploited throughout the characterization process. The code lines are reported in 3.1.2.

```
table = readtable(fullfile(simulation_path,
        'Additional_Information.txt'));

% saving all MATLAB data into a struct
tableNMol = load(fullfile(simulation_path,
            'simulation_output.mat'));
```

Listing 3.2: Importing simulation data.

As it can be noticed in Figure 3.2, the data in Additional_information.txt are basically the vertical clock values and the voltages on each molecule of the circuit for each time step of the simulation, along with the drivers of the circuit. It is fundamental to remark the types of labels that distinguish the information:

#time	CK_0003a	CK_0003b	...	CK_0028a	CK_0028b	Vout_0003a	Vout_0003b	...	Vout_0028a	Vout_0028b	driver_0015a	driver_0015b	driver_0016a	driver_0016b
1	-2	-2	...	-2	-2	0.093	0.093	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
2	-2	-2	...	-1	-1	0.094	0.093	...	-0.1	-0.1	0.926	-0.924	0.926	-0.924
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17	-1	-1	...	-2	-2	0.18	-0.134	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
18	0	0	...	-2	-2	0.784	-0.36	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
19	1	1	...	-2	-2	0.794	-0.352	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
20	2	2	...	-2	-2	0.827	-0.346	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
106	2	2	...	-2	-2	0.731	-0.096	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
107	1	1	...	-2	-2	0.645	-0.066	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
108	0	0	...	-2	-2	0.547	-0.004	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
109	-1	-1	...	-2	-2	0.115	0.129	...	-0.059	-0.079	0.9264	-0.924	0.926	-0.924
110	-2	-2	...	-2	-2	0.093	0.093	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
115	-2	-2	...	-2	-2	0.093	0.093	...	-0.059	-0.079	0.926	-0.924	0.926	-0.924

Figure 3.2: Example of part of the content of an Additional_information.txt file, represented in tabular form.

- The **#time** columns count the n-th time step of the simulation;
- each **CK_000X(a|b)** column refers to the clock values of the molecule X, where X is the number that univocally identifies it. The 'a' or 'b' distinguishes the 'standard' molecule from its complementary;
- the same nomenclature refers to **Vout_000X(a|b)** columns, which contain the voltage values of the molecules with name X, either standard or complementary;
- the **driver_00Y(a|b)** columns contain the drivers of the circuit. Also in this case 'a' and 'b' refer to standard and complementary cases, but the Y is not the same reference as X, which only applies to clocks and voltages.

As previously stated, the script imports other information from *launch.m* regarding the circuit layout, the drivers and the outputs:

- the relative paths of the SCERPA simulation;
- the name of the .qll file that has been simulated with SCERPA;
- information about the drivers and the clock (for instance, latency and number of clock regions);
- the angle of the termination and the `busLayout` flag (which tells whether the circuit has a bus layout or not).

3.1.3 The *outMol_finder.m* function

The outputs of the circuit given by SCERPA refer to the voltage values of the molecules right after the last body molecules of the circuit, that correspond to the first molecules of the termination; since it is necessary to get the voltage values of the last cells of the circuit, the only way is by starting from the ones given by the simulation. The *outMol_finder.m*

script has this exact aim: starting from the coordinates of the outputs of the simulation (which are saved in the ttableNmol struct), it evaluates the actual coordinates of the real outputs based on the angle of the termination and then it associates these coordinates to the names of the molecules to which they correspond. It is a fundamental operation, since it allows to obtain automatically the names of the outputs that are going to be used further in the characterization function. This association can be executed automatically for a circuit with any kind of output angle.
Moreover, the function returns both the names of the clock signals related to the original outputs and the actual outputs, considering that the label of a molecule is the same both for clock and voltage value: for instance, if a molecule's label is 'Vout_0001a', its clock label is 'CK_0001a'.
Figure 3.3 depicts a flowchart which summarises the procedure.
The workflow can be divided into the following steps:

1. The number of outputs and the coordinates of the termination outputs (saved in the format [x y z]) are read from the struct stack_output contained in tableNMol, shown in Figure 3.4, and saved.

2. Depending on the angle of the output termination, the actual output coordinates are obtained from the termination ones by shifting their y coordinate value (vertical shifting) or z coordinate value (horizontal shifting), as shown in Figure 3.6 and Figure 3.5.

3. Now that the actual output coordinates have been saved, their corresponding name has to be found in the 'stack_mol' struct, whose organization is the one in Figure 3.7: similarly to 'stack_out', it contains the coordinates of all the molecules in the circuit, but also their names (expressed in the 'identifier_qll' field).

4. Finally, all the coordinates of the molecules of the circuit are scanned: each element in the field 'position' of 'stack_mol' is compared with both each output coordinate and termination coordinate. If the coordinates coincide, the names of the outputs are derived by concatenating the string 'Vout_' to the related element of 'identifier_qll', in order to obtain the label 'Vout_xxx0(a|b)' that is the same as the headings of Additional_Information.txt.

In the end, the function returns a struct with the names of the actual outputs and the names of the clock signals of both the termination outputs and the actual outputs, since they need to be compared in the characterization.m function to check for the HOLD states. The relative code lines are shown in 3.1.3.

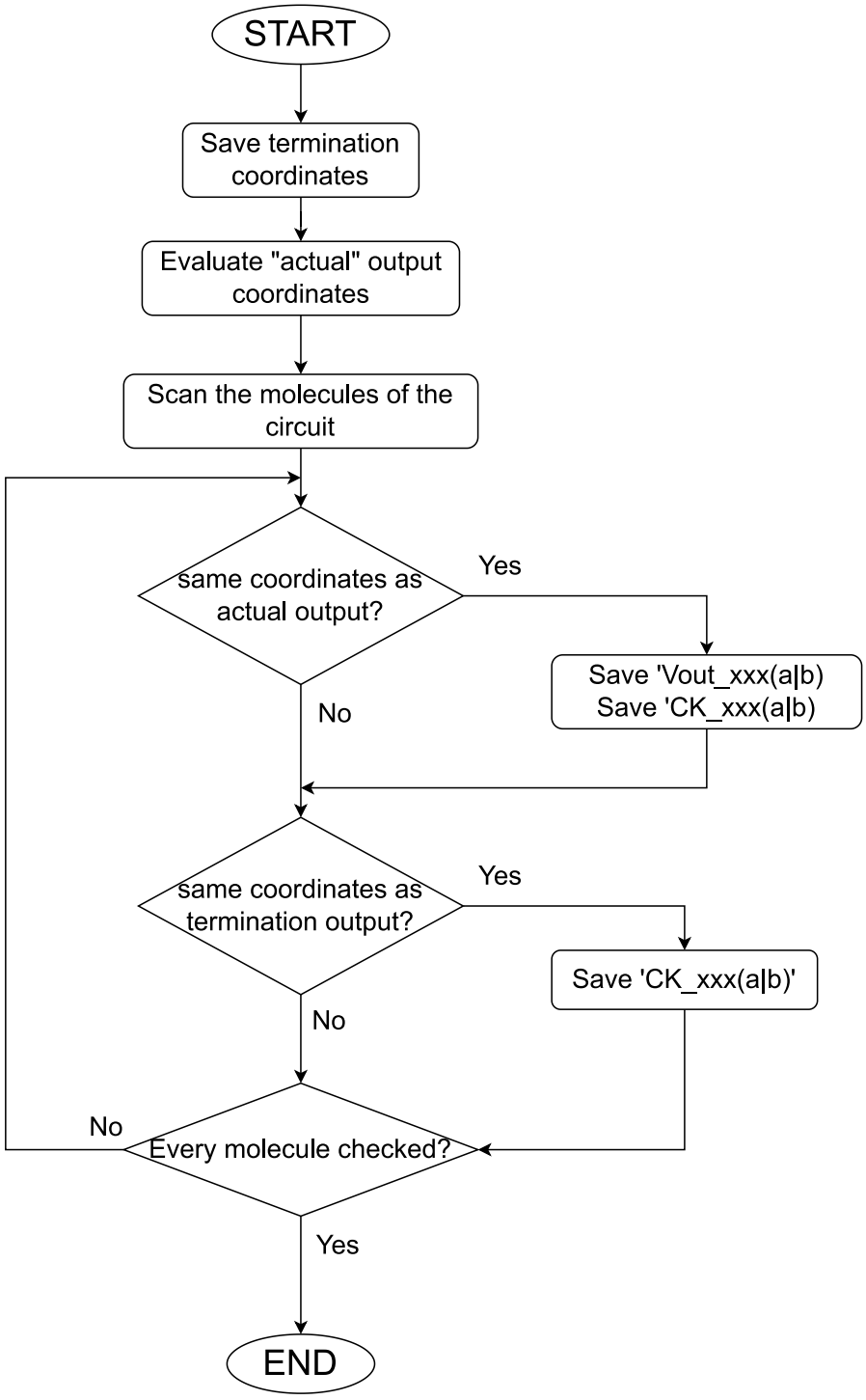


Figure 3.3: Flowchart that summarises the process of outMol_finder.m

Fields	charge	identifier	position	molType	identifier_qll	interactionRXlist	Vprobe
1	1x4 struct	'OUT'	'[0 1 19]'	1	'0001'	1x35 double	NaN
2	1x4 struct	'OUT'	'[0 0 19]'	1	'0002'	1x35 double	NaN

Figure 3.4: Content of the stack_output struct. The 'position' column contains the coordinates of the molecules, in the format [x y z].

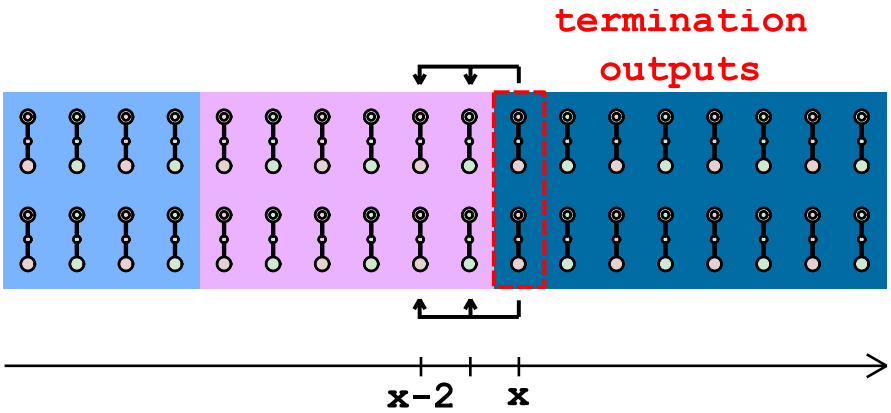


Figure 3.5: Closeup of the termination of the bus wire layout to visualise the horizontal coordinate shift.

```
%number of outputs
output_data.N_outputs = N_outputs;
%array with all molecules' positions
output_data.pos_mol = pos_mol;
%actual outputs' coordinates
output_data.out_coord = out_coord;
%'Vout_00xx'
output_data.output_labels = cell2mat(output_labels);
%'CK_00xx'
output_data.clock_labels = cell2mat(clock_labels);
%'CK_00xx'
output_data.term_clock_labels = cell2mat(term_clock_labels);
```

Listing 3.3: Data returned by outMol_finder.m

3.1.4 Organizing information in tables

Once the output labels are known, the first parsing operation can take place: each column of Additional_information.txt whose heading has the same label of a clock (either the clock signal of an actual output or the clock signal of a termination output) or an output

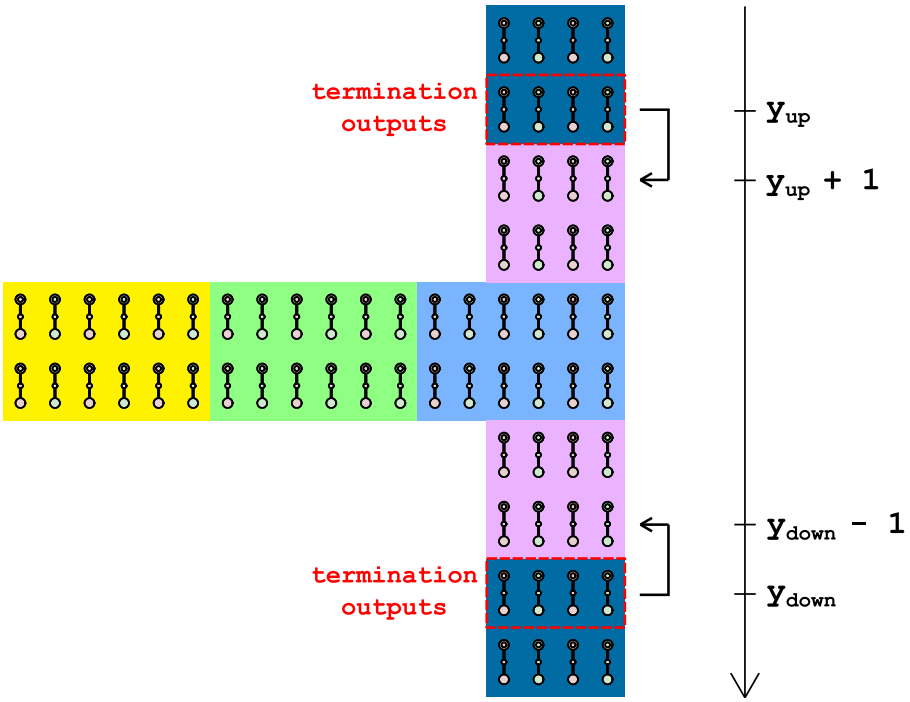


Figure 3.6: Closeup of the termination of the T-connection layout to visualise the vertical coordinate shift.

voltage is entirely copied from the structure in Figure 3.2 in a new table; the same thing applies for the relative driver values. This first distinction is needed to understand which kind of data are going to be processed: there is now one table containing the drivers, one containing the outputs and two containing the clock signals (of which one is related to the original outputs and one to the actual outputs).
Taking as example the same circuit of Figure 3.2, the resulting tables would be:
Every column of the drivers' table related to complementary drivers is removed, thus saving space and simulation time without losing significant data[2]. This operation is trivial since the complementary drivers are labeled in tableNMol with the same name of the original ones, with an added '_c' string to their end.
Furthermore, since in the bus layout mode there are two equal copies of each driver, in order to distinguish them a subscript (either 'a' or 'b') is added to the driver's name: for instance, instead of two drivers both named Dr1, there are now $Dr1_a$ and $Dr1_b$.
In order to be able to read the tables correctly in a second parsing operation, the **delay** parameter has to be introduced. As in standard digital circuits, the output information obtained by giving a certain input is not available immediately after the input is delivered:

[2]Complementary drivers can be easily derived knowing that they have opposite sign with respect to standard drivers.

Fields	charge	identifier	position	molType	Vext	identifier_qll	phase	interactionRXlist	clock
1	1x4 struct	'Mol_1'	'[0 1 15]'	1	0	'0003a'	4	1x39 double	-2
2	1x4 struct	'Mol_2'	'[0 1 14]'	1	0	'0003b'	4	1x39 double	-2
3	1x4 struct	'Mol_3'	'[0 0 15]'	1	0	'0004a'	4	1x39 double	-2
4	1x4 struct	'Mol_4'	'[0 0 14]'	1	0	'0004b'	4	1x39 double	-2
5	1x4 struct	'Mol_5'	'[0 0 17]'	1	0	'0005a'	4	1x36 double	-2
6	1x4 struct	'Mol_6'	'[0 0 16]'	1	0	'0005b'	4	1x38 double	-2
7	1x4 struct	'Mol_7'	'[0 1 17]'	1	0	'0006a'	4	1x36 double	-2
8	1x4 struct	'Mol_8'	'[0 1 16]'	1	0	'0006b'	4	1x38 double	-2
9	1x4 struct	'Mol_9'	'[0 1 11]'	1	0	'0007a'	3	1x38 double	-2
10	1x4 struct	'Mol_10'	'[0 1 10]'	1	0	'0007b'	3	1x36 double	-2
11	1x4 struct	'Mol_11'	'[0 0 11]'	1	0	'0008a'	3	1x38 double	-2
12	1x4 struct	'Mol_12'	'[0 0 10]'	1	0	'0008b'	3	1x36 double	-2
13	1x4 struct	'Mol_13'	'[0 0 13]'	1	0	'0009a'	3	1x39 double	-2
14	1x4 struct	'Mol_14'	'[0 0 12]'	1	0	'0009b'	3	1x39 double	-2
15	1x4 struct	'Mol_15'	'[0 1 13]'	1	0	'0010a'	3	1x39 double	-2
16	1x4 struct	'Mol_16'	'[0 1 12]'	1	0	'0010b'	3	1x39 double	-2
17	1x4 struct	'Mol_17'	'[0 1 7]'	1	0	'0011a'	2	1x30 double	-2
18	1x4 struct	'Mol_18'	'[0 1 6]'	1	0	'0011b'	2	1x28 double	-2
19	1x4 struct	'Mol_19'	'[0 0 7]'	1	0	'0012a'	2	1x30 double	-2
20	1x4 struct	'Mol_20'	'[0 0 6]'	1	0	'0012b'	2	1x28 double	-2
21	1x4 struct	'Mol_21'	'[0 0 9]'	1	0	'0013a'	2	1x34 double	-2
22	1x4 struct	'Mol_22'	'[0 0 8]'	1	0	'0013b'	2	1x32 double	-2
23	1x4 struct	'Mol_23'	'[0 1 9]'	1	0	'0014a'	2	1x34 double	-2

Figure 3.7: Organization of 'stack_mol'.

CK_0005a	CK_0005b	CK_0006a	CK_0006b	Vout_0005a	Vout_0005b	Vout_0006a	Vout_0006b	driver_0015a	driver_0015b	driver_0016a	driver_0016b
-2	-2	-2	-2	-9.10E-02	-9.10E-02	9.30E-02	9.31E-02	0.9264713	-0.924	0.9264713	-0.924
-2	-2	-1	-1	-6.26E-02	-8.89E-02	7.70E-02	9.42E-02	0.9264713	-0.924	0.9264713	-0.924
...	...	...	...	...	...	...	...	...	...	...	...
-1	-1	-1	-1	-1.27E-01	-1.07E-01	1.30E-01	1.09E-01	0.9264713	-0.924	0.9264713	-0.924
0	0	0	0	8.24E-02	-5.96E-01	4.81E-01	-3.03E-01	0.9264713	-0.924	0.9264713	-0.924
1	1	1	1	6.44E-02	-6.11E-01	4.93E-01	-2.97E-01	0.9264713	-0.924	0.9264713	-0.924
2	2	2	2	8.27E-02	-6.51E-01	5.45E-01	-3.51E-01	0.9264713	-0.924	0.9264713	-0.924
...	...	...	...	...	...	...	...	...	...	...	...
2	2	2	2	8.77E-02	-6.57E-01	5.55E-01	-3.71E-01	0.9264713	-0.924	0.9264713	-0.924
1	1	1	1	6.62E-02	-6.00E-01	4.89E-01	-2.82E-01	0.9264713	-0.924	0.9264713	-0.924
0	0	0	0	8.13E-02	-5.99E-01	4.83E-01	-2.14E-01	0.9264713	-0.924	0.9264713	-0.924
-1	-1	-1	-1	-1.28E-01	-1.02E-01	1.29E-01	1.15E-01	0.9264713	-0.924	0.9264713	-0.924
-2	-2	-2	-2	-9.10E-02	-9.10E-02	9.30E-02	9.31E-02	0.9264713	-0.924	0.9264713	-0.924
...	...	...	...	...	...	...	...	...	...	...	...
-2	-2	-2	-2	-9.10E-02	-9.10E-02	9.30E-02	9.31E-02	0.9264713	-0.924	0.9264713	-0.924

Figure 3.8: Example of the tables obtained after the first modification.

one has to wait for a certain amount of time, i.e. a delay, before being able to determine it. Since the timing unit, in this approach, is represented by time steps, the delay between inputs and outputs can be evaluated as:

$$delay = phasesRepetition \times cycleLength \tag{3.1}$$

where phasesRepetition refers to how much times equal phases of the circuit are repeated in the layout and cycleLength is the parameter defined in subsection 1.3.4.

3.1.5 The *rename_outputs.m* function

Once the numeric values are organized in the correct way, there is still another modification to consider. Up to this point, the names of the outputs were expressed in the form 'Vout_00xx', which refers to the molecule on which they are located. Since the whole process has the goal of being the most generic possible, it is more appropriate to name the outputs after the *position* of the molecule to which they belong with respect to the whole circuit, in order to be able to trace them in an univocal way. For this reason, considering the notation depicted in section 2.1, the form 'Vout_Y' is more suited for the aim[3].
The function *rename_outputs.m* has the aim of taking a set of output coordinates and returning their corresponding modified labels with the 'Vout_Y' notation. Table 3.1 shows an example of this modification for a wire bus structure with horizontal propagation (α = 0°), while Table 3.2 shows it for a T-connection block (α = 90°, 270°).

Table 3.1: Renamed outputs in a bus wire.

Coordinates	Original name	Modified name
[0,3,18]	Vout_0005b	Vout_B
[0,3,19]	Vout_0005a	Vout_A
[0,2,18]	Vout_0006b	Vout_D
[0,2,19]	Vout_0006a	Vout_C

Table 3.2: Renamed outputs in a T-connection block. The 'd' and 'u' subscripts refer to the downwards or upwards direction.

Coordinates	Original name	Modified name
[0,9,26]	Vout_0005b	Vout_B_d
[0,9,27]	Vout_0005a	Vout_A_d
[0,9,24]	Vout_0006b	Vout_D_d
[0,9,25]	Vout_0006a	Vout_C_d
[0,4,26]	Vout_0005b	Vout_B_u
[0,4,27]	Vout_0005a	Vout_A_u
[0,4,24]	Vout_0006b	Vout_D_u
[0,4,25]	Vout_0006a	Vout_C_u

The flowchart that summarises the function is reported in Figure 3.9. Since the goal is to execute it in the most generic and automated way possible, the algorithm follows the same procedure regardless of the kind of circuit:

[3]This notation is helpful to indicate clearly the position of the molecules that belong to the most external blocks of the circuit, where the outputs are, and it improves significantly the readability.

- It evaluates the maximum and minimum coordinates in the horizontal (z-axis) and vertical (y-axis) direction, in order to check for the position of the molecules;
- It reads the angle of propagation: with just one angle the labels are 4, while for two angles the labels are 8;
- It assigns the labels to the molecules according to their position: in a bus wire, for instance, it is sufficient to associate the four ABCD labels since the 4 molecules are positioned in a square-like pattern and the min-max values can be easily assigned. In a structure with more angles, as a T-connection, in a dichotomic way it first distinguishes the direction of propagation and then exploits the relative position with the ymax-ymin-zmax-zmin points.

As a result, the new labels are saved in a cell array of the same dimension of the input one, with the outputs in the same order.

3.1.6 Writing the .csv files and info.txt

The final part of the process is the one concerning the creation of the .csv files, one for each output of the circuit.
A second parsing step is performed on the data structures created until now: for each output, the content of the two clock tables is compared for each row (i.e. for each time step of the simulation) and, if the values both correspond to the +2 clock value, the output values and the driver values of the same row of the respective columns are copied into two further tables (respectively) since they are both in HOLD state[4].
More specifically, the i-th rows from which the output values are extracted are the same with respect to the clock rows in HOLD, whereas the corresponding drivers are extracted from the (i-delay)-th since, as explained in subsection 3.1.4, taking inputs and correlated outputs at the same time instant does not make sense. Figure 3.10 shows an example of the process in case of a bus wire with phasesRepetitions = 1 and delay = 20.
Once these modifications have taken place, the resulting output's table, together with the corresponding table of drivers, are merged together into a single one; the headings of such a table correspond to the output's name and the drivers' names. In the end, the table is converted into a .csv file. Table 4.1 shows a possible output of the process.
The process is repeated for every output in the same way, so that at the end there are as many .csv files as the outputs of the circuit.

3.1.7 Writing the info.txt file

The characterization.m function also provides an additional file, *info.txt*, with further information about the circuit. It contains the latency of the circuit, the number of clock

[4]Referring to subsection 1.3.4, the HOLD state is the state in which the values are stable and can be read properly.

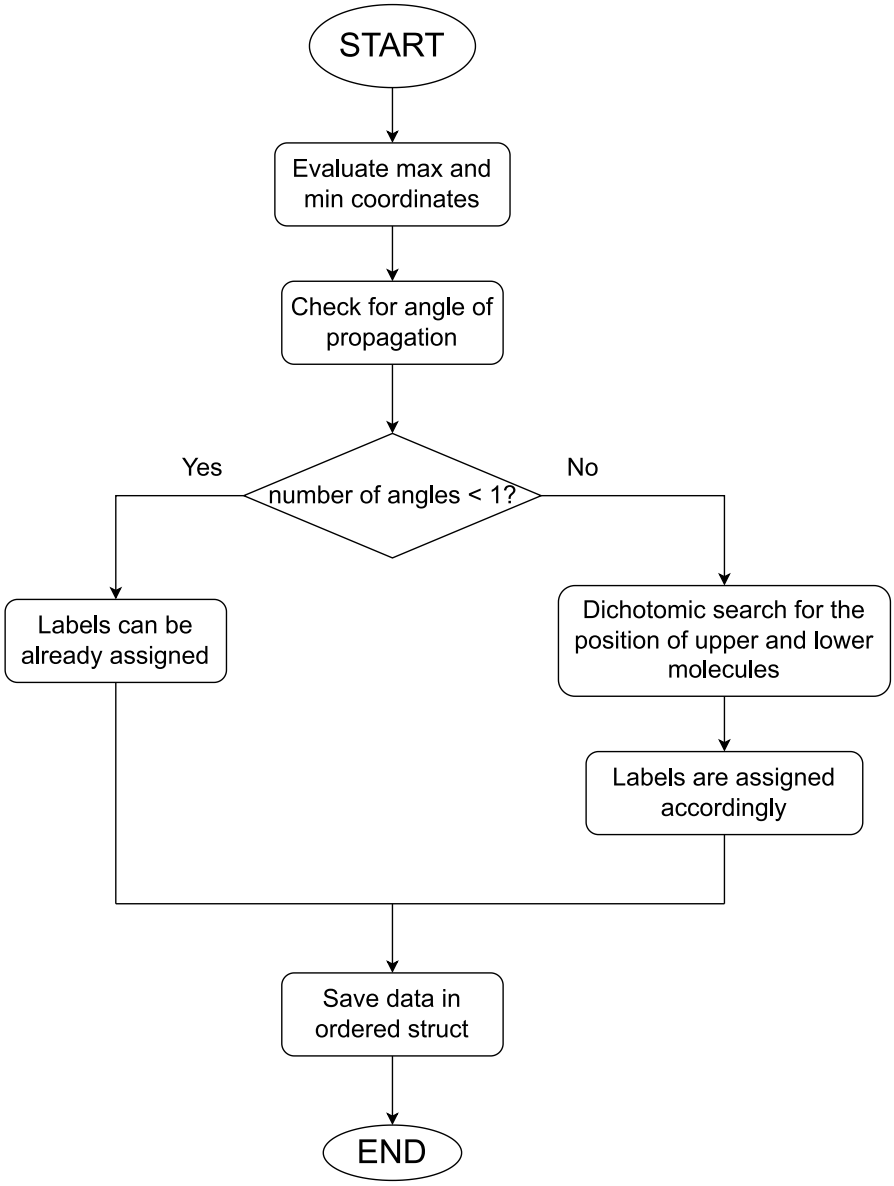


Figure 3.9: Flowchart of the function `rename_outputs`.

phases and the maximum driver voltage, obtained from the data given in the launch script, but also the names of the outputs and the approximate area of the circuit.
The method used to evaluate the area of the circuit is the following:

1. The maximum and minimum coordinates of the circuit in both dimensions, in terms of molecules (respectively, y_{min}, y_{max}, z_{min} and z_{max}), are extracted from from the MAT file 'simulation_output.mat' in order to evaluate the number of cells N_{cells};

time	CK_0005a	CK_0005b	CK_0006a	CK_0006b	Vout_A	Vout_B	Vout_C	Vout_D	Dr1_a	Dr1_b
1	-2	-2	-2	-2	-9.10E-02	-9.10E-02	9.30E-02	9.31E-02	9.26E-01	9.26E-01
2	-2	-2	-2	-2	-6.26E-02	-8.89E-02	7.70E-02	9.42E-02	9.26E-01	9.26E-01
3	-2	-2	-2	-2	6.91E-02	-7.26E-02	1.58E-01	1.08E-01	9.26E-01	9.26E-01
4	-2	-2	-2	-2	1.08E-01	-6.66E-02	1.60E-01	1.08E-01	9.26E-01	9.26E-01
5	-2	-2	-2	-2	1.08E-01	-6.66E-02	1.60E-01	1.08E-01	9.26E-01	9.26E-01
6	-2	-2	-2	-2	1.08E-01	-6.66E-02	1.60E-01	1.08E-01	9.26E-01	9.26E-01
...	...	...	...	...	...	...	...	...	...	...
i	2	2	2	2	3.54E-01	-7.66E-01	8.32E-01	-3.55E-01	7.58E-01	7.58E-01
i+1	2	2	2	2	3.54E-01	-7.66E-01	8.32E-01	-3.55E-01	7.58E-01	7.58E-01
	...	...	...	...	...	...	...	...	...	...
i+delay	2	2	2	2	3.55E-01	-7.65E-01	8.32E-01	-3.55E-01	-6.78E-01	-6.78E-01
i+1+delay	2	2	2	2	3.55E-01	-7.65E-01	8.32E-01	-3.55E-01	-6.78E-01	-6.78E-01
	...	...	...	...	...	...	...	...	...	...
i+2*delay	2	2	2	2	-7.64E-01	3.55E-01	-3.47E-01	8.26E-01	-0.924	-0.924
i+1+2*delay	2	2	2	2	-7.64E-01	3.55E-01	-3.47E-01	8.26E-01	-9.24E-01	-9.24E-01
	...	...	...	...	...	...	...	...	...	...
i+3*delay	2	2	2	2	-7.64E-01	3.55E-01	-3.47E-01	8.26E-01	-0.924	-0.924
i+1+3*delay	2	2	2	2	-7.64E-01	3.55E-01	-3.47E-01	8.26E-01	-0.924	-0.924
...	...	...	...	...	...	...	...	...	...	...

Figure 3.10: Table rows that show how the outputs and corresponding drivers are extracted. Each output is associated with the drivers of the same colour.

Table 3.3: Example of a table showing the output Vout_A and its relative inputs.

Vout_A	$Dr1_a$	$Dr1_b$
-0.76	0.75	0.75
-0.76	0.92	0.92
0.36	-0.95	-0.95
0.36	-0.67	-0.67

2. The distance between Dot1 and Dot2 inside a single molecule, Δy, is calculated by considering the value 'y' of the struct 'charge'[5];

3. The distance between the Dot1 of two adjacent molecules, Δz, is evaluated in the same way, with the value 'z'.

In the end, the value of the area is given by:

$$A_{total} = \Delta y \cdot \Delta z \cdot N_{cells} \tag{3.2}$$

Figure 3.11 shows an example of the discussed variables for what concerns the layout of a T-connection termination.

[5]'Charge' is one of the fields of 'stack_mol', contained in tableNMol, which tells the coordinates of the charges in the three dimensions

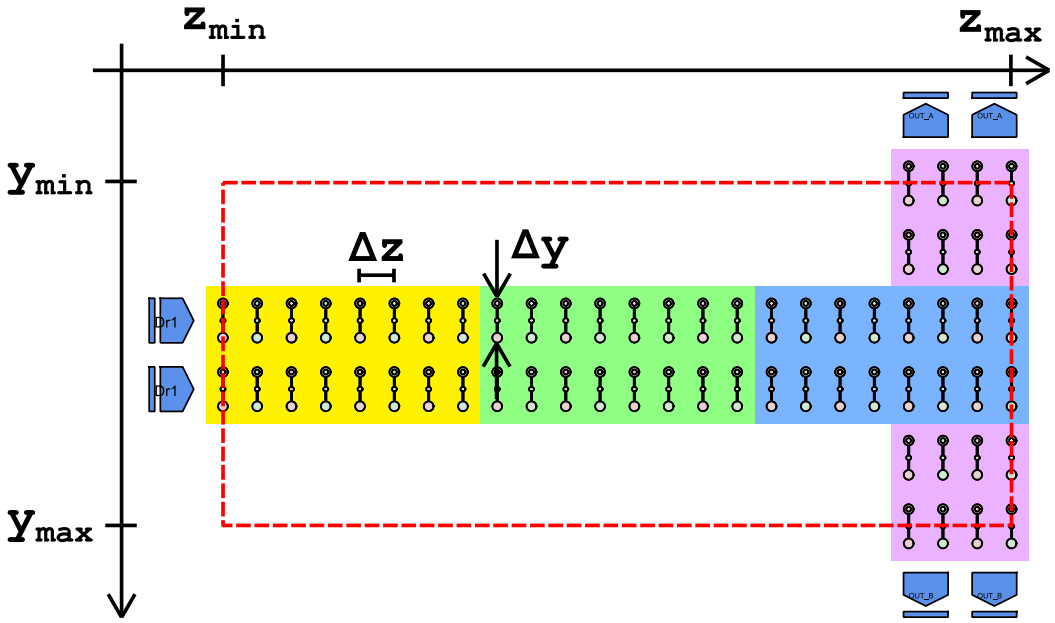


Figure 3.11: Evaluation of the area of a T-connection termination.

3.2 The *InOut_eval.m* function

The last feature of the Characterization Tool is the *InOut_eval.m* function, which can be launched only after the *characterization.m* function has been executed: its main principle is the exploitation of the .csv files as libraries, in order to extract specific data. For this reason, in order to launch it from launch.m the flags debugMode and LibEvaluation are set, respectively, to 0 and 1.
It takes as inputs:

- the simulation path;
- the name of the file;
- an array of inputs.

The input values, which must be inserted by the user as an array of numeric voltage values, represent the inputs of the circuit that is being analyzed. The function searches for the directories that contain the files related to the characterization of the specific circuit, then for each library file (i.e. each .csv file) it makes a comparison between the input values given by the user and the driver values in the circuit, then returns the corresponding output values (or, if the inputs are not exactly the same, the closest ones).
Figure 3.13 depicts a flowchart of the function. For each library file examined by the function, a subtraction is made between every element of the i-th column and the i-th element of the input array. The result is stored into a matrix that records these different, which tell how much the recorded values are far from the input ones. After the matrix is

created, it is scanned to check, for each column, where is the minimum value; after that, each row is scanned and the row that contains the highest number of minima is considered as the one that contains the values closer to the given inputs: the corresponding output value is saved into the struct 'Vout'. Figure 3.12 shows an example of the kind of data that are selected: the input array is represented below the table, while the yellow cells represent the values of the columns that are closer to the corresponding input values.

Vout_D	DrB_a	DrB_B	Dr0_a	Dr0_b	DrA_a	DrA_b
-0.344591	-0.924	-0.924	-0.924	-0.924	-0.924	-0.924
-0.344591	-0.677704	-0.677704	-0.924	-0.924	-0.677704	-0.677704
-0.344536	-0.924	-0.924	-0.924	-0.924	0.926471	0.926471
-0.344522	0.757645	0.757645	-0.924	-0.924	-0.677704	-0.677704
-0.344493	0.926471	0.926471	-0.924	-0.924	-0.924	-0.924
-0.34449	-0.677704	-0.677704	-0.924	-0.924	0.757645	0.757645
0.828754	0.926471	0.926471	-0.924	-0.924	0.926471	0.926471
0.828758	0.757645	0.757645	-0.924	-0.924	0.757645	0.757645
input:	0.9	0.7	-0.924	-0.924	0.7	0.3

Figure 3.12: Example of a table examined in the function InOut_eval.m.

This function is the reason why the whole characterization process is so powerful. As mentioned in section 1.6, it gives a specific output value corresponding to a specific set of inputs without having to simulate again the whole structure and without having to create again all the library files, saving a significant amount of simulation time.
When it comes to simulate more complex structures, composed of more than one simple device (for instance, a NAND gate in MolFCN logic is composed of a majority voter followed by an inverter), the output of a block can be considered as the input of another one: exploiting this cascade behaviour with which information propagates, the *InOut_eval.m* function can be used repeatedly for each simple block with, as input values, the output values of the previous block. The last one gives the output value of the whole structure.

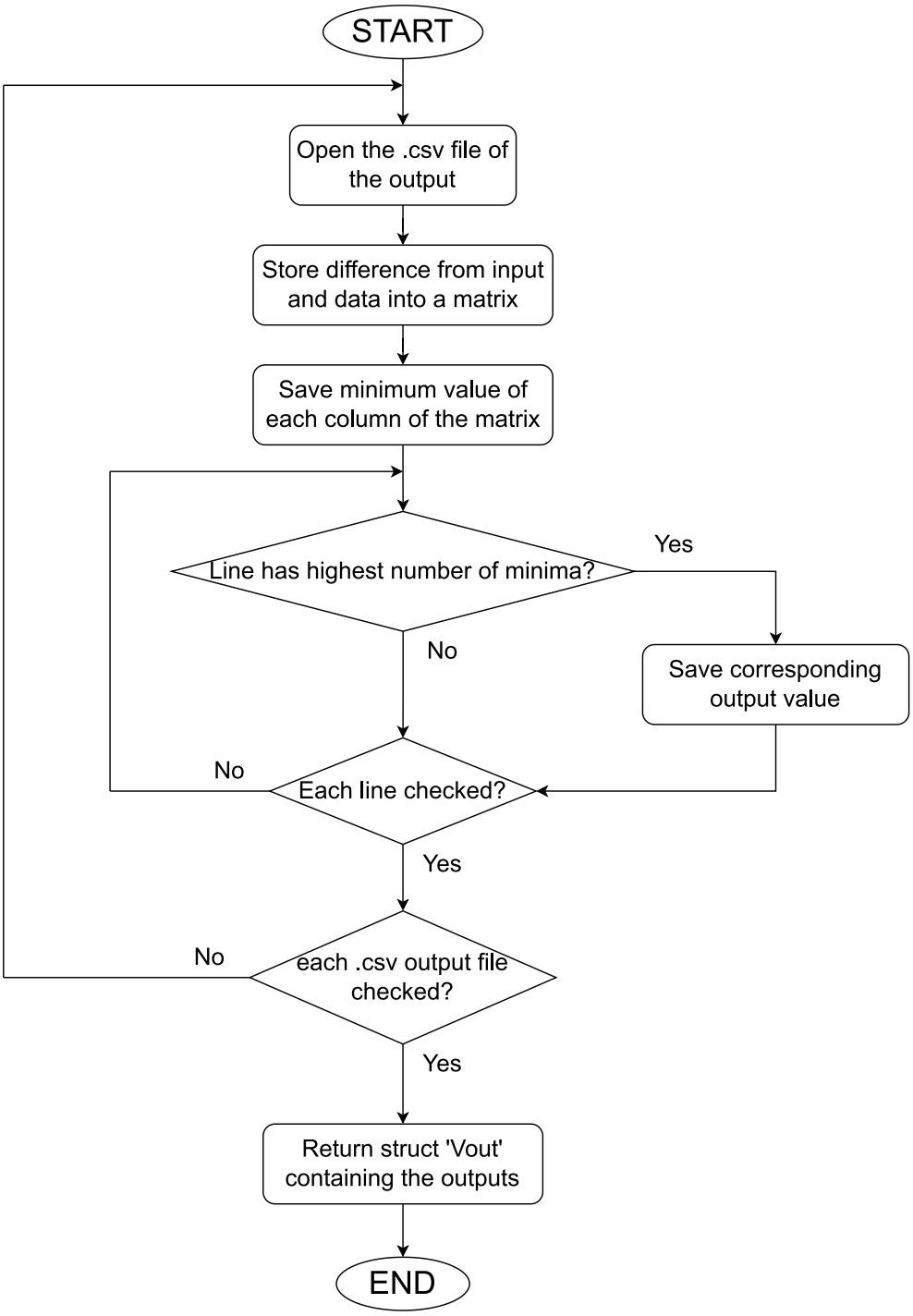


Figure 3.13: Flowchart of the InOuteval.m function.

Part III

Simulation of a XOR gate

Chapter 4

Characterization of simple blocks

Once the main features of the characterization tool have been described, it is possible to apply the method on some useful blocks that are going to be used later to build a more complex structure, starting from the simplest one (a bus wire) and introducing more advanced blocks, depending on which shapes and logic functions are needed.
There are two kinds of blocks that can be generated and used:

- **interconnections**, as wires, L-connections, branches, T-connections;
- **logic gates**, as inverters or majority voters.

In this chapter each of these blocks' characteristics will be illustrated; simulation parameters that the circuits have in common are maintained the same for every simulation and are depicted in 4.

```
clock_low = -2; clock_high = +2; clock_step = 5;
pSwitch =  linspace(clock_low, clock_high, clock_step);
pHold =   linspace(clock_high, clock_high, clock_step);
pRelease =  linspace(clock_high, clock_low, clock_step);
pReset =   linspace(clock_low, clock_low, clock_step);
pCycle = [pSwitch pHold pRelease pReset];
driverPara.doubleMolDriver = 1;
% number of sweep steps (define accuracy of the simulation)
driverPara.NsweepSteps = 4;
% number of clock regions in the layout
driverPara.NclockRegions = 4;
% number of times NclockRegions repeat in the layout
driverPara.phasesRepetition = 1;
terminationSettings.busLayout = 1;
```

Listing 4.1: Clock simulation parameters

The number of clock phases for every block is 4: a termination is added to each of the simulated circuits, as explained in section 2.4, which means adding another clock phase after the end of the layout. However, in order to achieve a clean visualization of the layouts, termination is not shown in the pictures that follow.

4.1 Interconnections

4.1.1 Wire

The simplest interconnection analized in this project is the bus wire, whose MagCAD representation is shown in Figure 4.1.

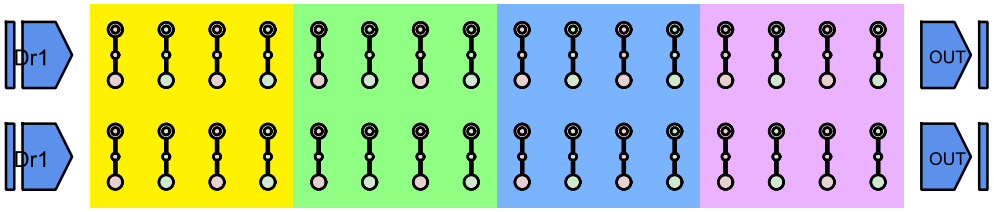


Figure 4.1: MagCAD representation of a bus wire structure.

This circuit guarantees transmission of data from input to output, propagating charges only in the horizontal direction with an angle $\alpha = 0°$.

4.1.2 L-connection

The L-connection block guarantees transmission of information in the vertical direction, either upwards (with an angle of $\alpha = 270°$, depicted in Figure 4.3) or downwards (with an angle of $\alpha = 90°$, depicted in Figure 4.2).

4.1.3 Branch connection

The branch interconnection block allows to duplicate the information in order to propagate it in two directions, where one is the horizontal one (angle $\alpha = 0°$) and the other one is either the vertical upwards one, with an angle $\alpha = 270°$ (Figure 4.4) or the vertical downwards one, with an angle $\alpha = 90°$ (Figure 4.5).

4.1.4 T-connection

The T-connection, similarly to the branch connections, is used to propagate information in two directions, but in this case both the directions are vertical, one upwards and one downwards, meaning that the two angles of the termination are respectively $\alpha = 270°$ and $\alpha = 90°$.

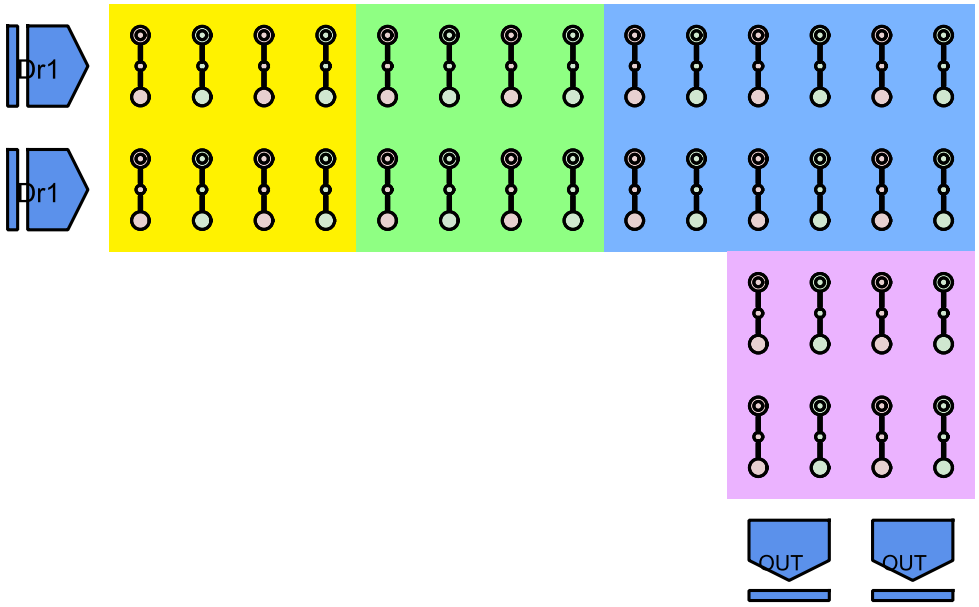


Figure 4.2: MagCAD representation of a L-connection with a downwards propagation.

4.2 Logic gates

4.2.1 Inverter

The inverter is a logic gate that simply inverts the sign of its input signal. Its 4-phases layout is shown in Figure 4.7.

4.2.2 Majority voter

Finally, the majority voter layout is depicted in Figure 4.8. Its truth table has been described in Table 1.1
Each of the mentioned circuits can be simulated on SCERPA, with the parameters listed in 4, as a first step of the process. The parameter $N_{sweepsteps}$ is fundamental for what concerns the accuracy of the simulation: the higher the granularity, the finer the values obtained. Simple circuits ought to be simulated with a $N_{sweepsteps}$ as high as possible, as far as the increased simulation time allows, in order to get more complete libraries; on the other hand, for bigger circuits it is not necessary to employ a high granularity since they do not need to be characterized and, most importantly, simulation time would increase dramatically, as it will be shown in the next chapters.
Furthermore, when it comes to simple blocks, rather than simulating the logic inputs '0' or '1' described in section 2.3 it is more convenient to test the inputs 'sweep' or not_sweep', since they allow to obtain a more complete variety of combinations of values between the

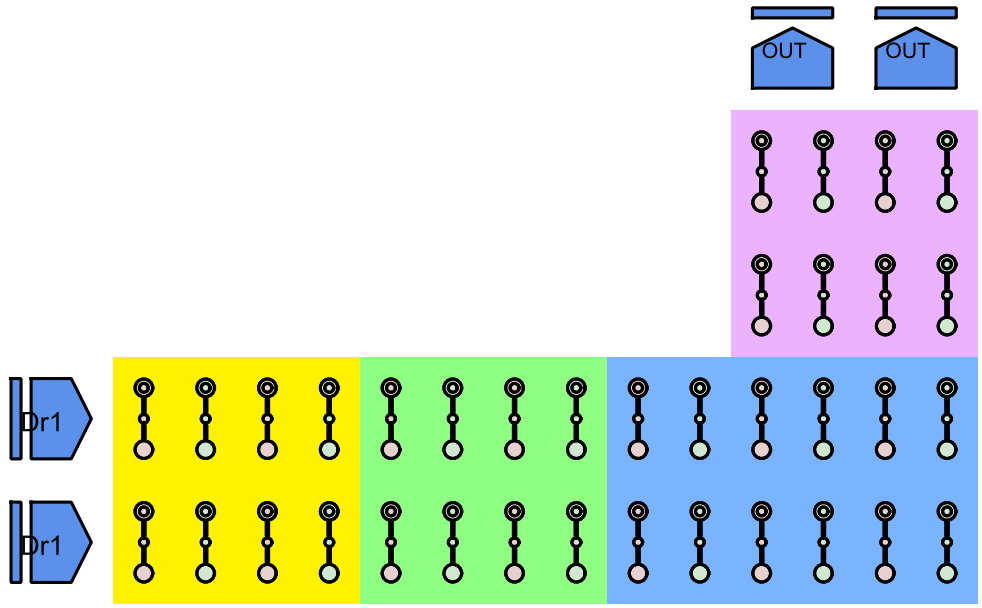


Figure 4.3: MagCAD representation of a L-connection with an upwards propagation.

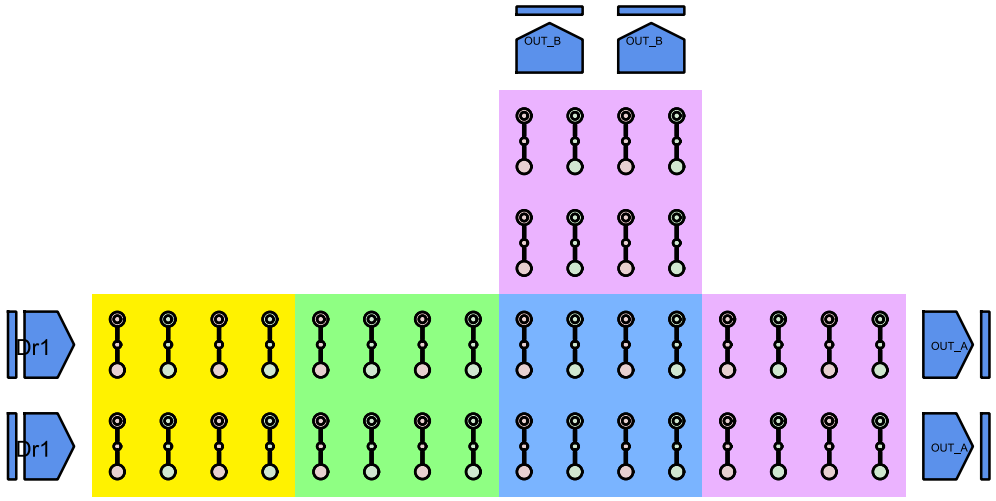


Figure 4.4: MagCAD representation of a Branch connection with an upward propagation.

logic '0' and the logic '1'. On the other hand, complex circuits can be simulated with just '0' or '1' as inputs since they do not need to be characterized and the corresponding logic values are most likely to be already present in the libraries of the simple blocks they are

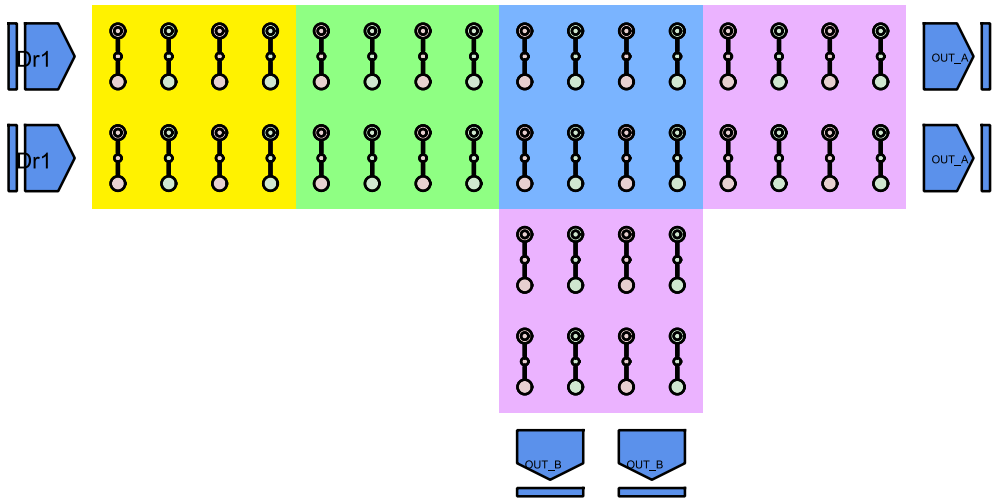


Figure 4.5: MagCAD representation of a Branch connection with a downward propagation.

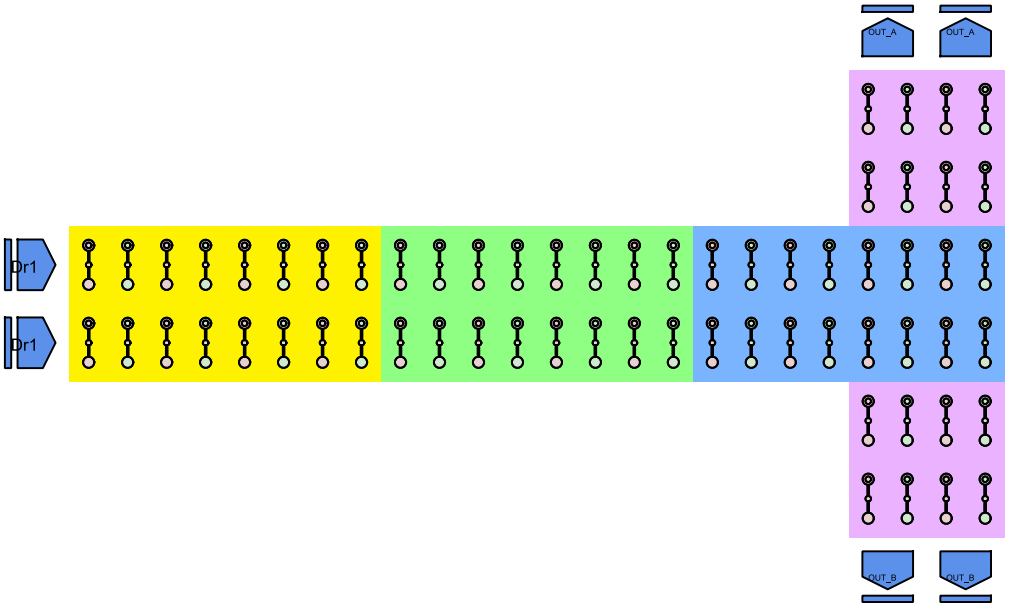


Figure 4.6: MagCAD representation of a T-connection.

made of.

After its SCERPA simulation, each circuit is characterized by creating the corresponding libraries in the charResults folder.

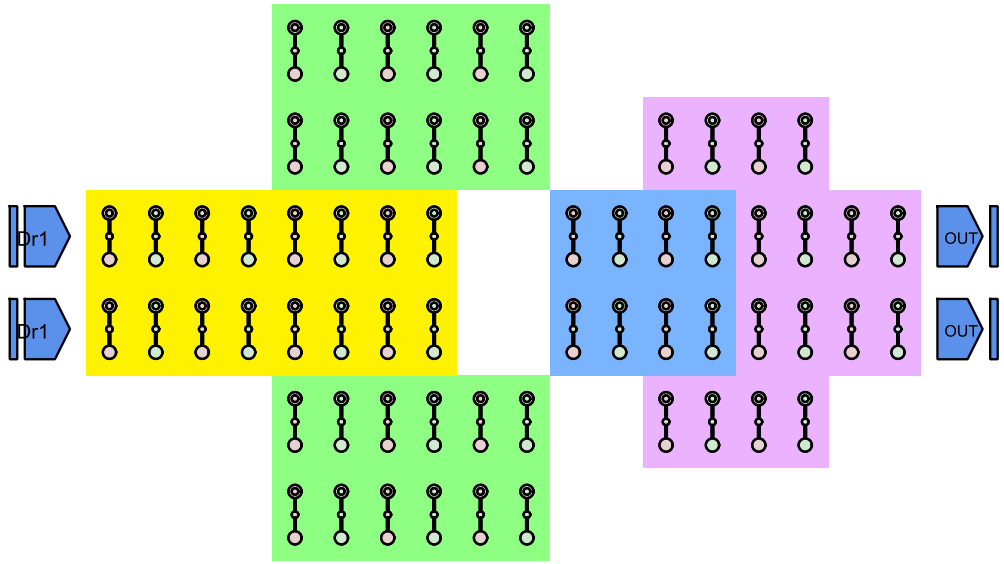


Figure 4.7: MagCAD representation of an inverter.

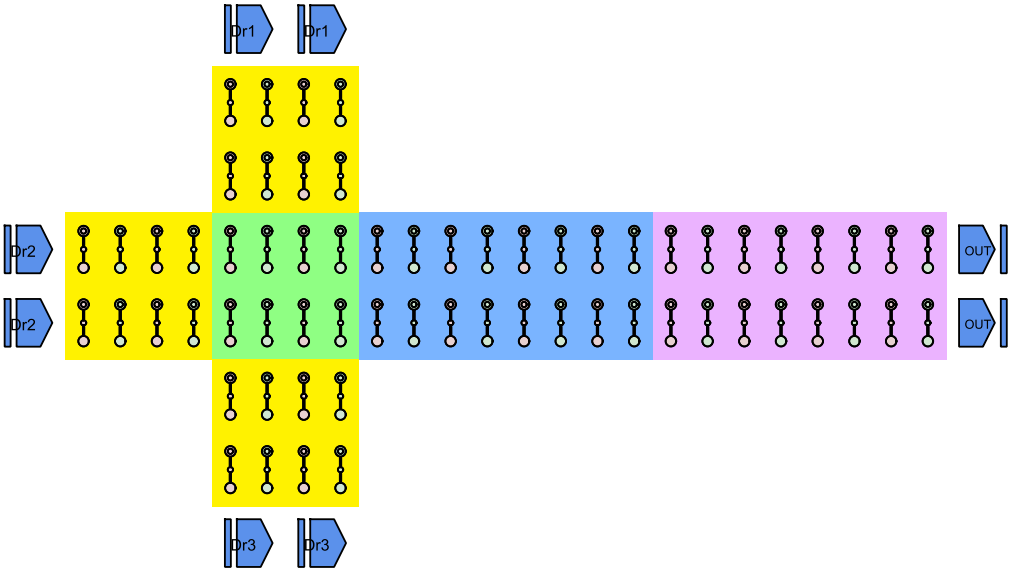


Figure 4.8: MagCAD representation of a majority voter.

4.3 Creation and validation of a NAND gate

The first composite circuit that can be built with the blocks introduced before is the NAND gate, which can be represented by an AND gate followed by a NOT gate, as in Figure 1.17. Since its logic structure involves a Majority Voter followed by an Inverter in a MolFCN design, the corresponding MagCAD representation that exploits the blocks introduced until now would be the one depicted in Figure 4.9.

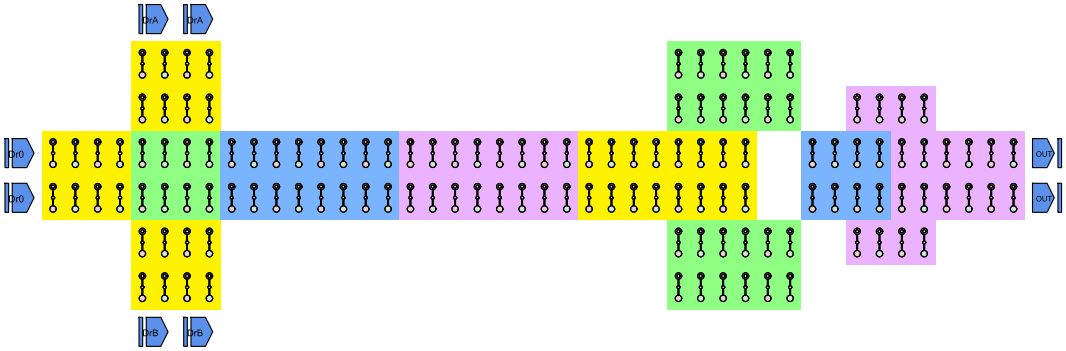


Figure 4.9: MagCAD representation of a NAND gate.

To perform a first validation of the process described until now, applied to the NAND, the aim is to:

1. simulate the NAND gate on SCERPA with the same inputs of its first block, which is the Majority Voter;

2. check for the values on the output molecules, that can be found in the tableNMol struct exactly as for the previous circuits;

3. browse the libraries of Majority Voter to retrieve the related outputs, then pass them as inputs to the Inverter by browsing its libraries[1]: the resulting output of the Inverter, in this configuration, should correspond (with a certain error margin) to the output of the NAND.

The SCERPA simulation of the NAND in Figure 4.9 can be executed with $N_{sweepsteps}$=1 and phasesRepetition=2, since equal phases are repeated 2 times in the circuit.
The fixed logic '0' value is assigned to Dr0, while DrA and DrB are the variable ones, to which the 4 combinations of logic '0' and logic '1' are assigned. In each simulation only a combination of inputs can be tested, so 4 simulations are needed.
The Majority Voter and the Inverter, then, are both simulated separately with SCERPA; the former should be, ideally, simulated with every possible combination of 'sweep' and

[1]The browsing operation is performed by InOut_eval.m (section 3.2).

'not_sweep' applied to DrA and DrB (in order to obtain the most complete characterization of the circuit), but for demonstrative aims just a few combinations are considered. After that, for each combination of inputs some driver values are taken from the Additional_Information.txt file of the NAND simulation and given as inputs of InOut_eval.m, which in this case is called 2 times (one for the Majority Voter and one for the Inverter, in cascade), as shown in 4.3.

```
% Majority Voter takes the same inputs of the NAND
Vout_MV = InOut_eval(BBcharPath,'majority_voter',
                     [zero_driver,zero_driver]);
% Inverter receives the output values
% of the Majority Voter
Vout_inv = InOut_eval(BBcharPath,'inverter',
                     [Vout_MV,Vout_MV]);
```

Listing 4.2: Evaluation of the outputs of the NAND with InOut_eval.m.

Finally, the results are compared in Table 4.1.

Table 4.1: Table comparing the results of the SCERPA simulation alone with the results of the characterization process.

Dr0	DrA	DrB	Vout_A (SCERPA)	Vout_A (characterization)	error (%)
0	0	0	0.3549	0.3626	2.12%
0	0	1	-0.764	-0.760	0.52%
0	1	0	-0.764	-0.760	0.52%
0	1	1	0.3549	0.3626	2.12

Chapter 5

Construction and simulation of the XOR gate

5.1 Building the XOR structure

The simulation of the NAND gate gives a first hint of the meaning of the project, but it is necessary to build a more complex structure in order to appreciate its true potential advantages. A further example could be the one of a XOR gate.
One of the ways in which a XOR gate can be obtained is by connecting 4 NAND gates together, as shown in Figure 5.1, in order to exploit the reusability of the NAND gate simulated before.

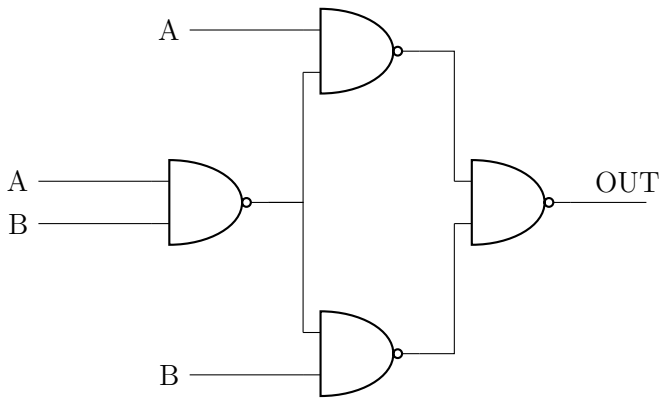


Figure 5.1: Logic representation of a XOR gate made exclusively of NAND gates.

To represent this configuration in a MolFCN design, it is necessary to connect the 4 gates to the drivers and to the outputs by adding the interconnections described in chapter 4, in a structure as the one depicted in Figure 5.2.
The basic principle is the same followed in section 4.3: the first step consists of simulating the main circuit on SCERPA, then simulating and characterizing the blocks and then

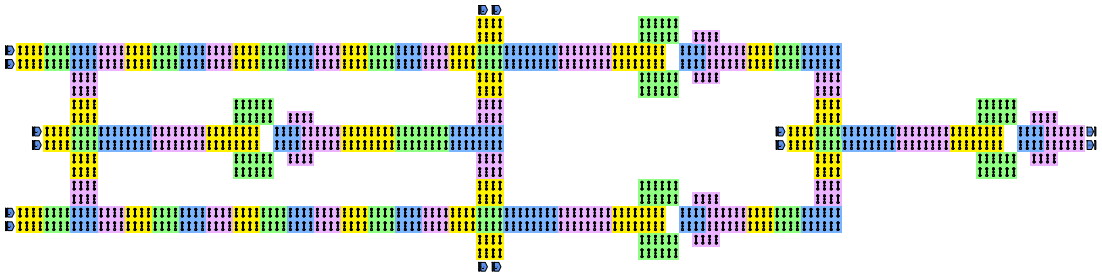


Figure 5.2: MagCAD representation of a XOR gate.

browsing the libraries obtained to study the difference between the respective results.

5.2 Simulating without the termination

The simulations mentioned until now have performed with the implementation of a termination "appendix", added to the end of the circuit, that increased stability. However, since bigger circuits can be particularly time consuming, a faster approach would be not to consider such termination: the outputs are carried in the exact same way, but their values are likely to be smaller in absolute value and less stable. This tradeoff allows to achieve slightly faster responses, without losing too much accuracy.
As a matter of fact, the error represented by the difference between the results obtained with and without termination (**term_error**) is expected to be similar regardless of the complexity of the structure: it is sufficient to evaluate it for a bus wire in order to predict approximately the actual output value also for a XOR.
The procedure to simulate the structure and characterize it has already been pictured in the previous sections; 5.2 shows the inputs given to the InOut_eval.m function to calculate the outputs, while Table 5.1 shows the value of **term_error** in a simulated bus wire, along with the difference in terms of simulation time overhead Δt, evaluated as:

$$\Delta t = time_{with} - time_{without} \tag{5.1}$$

where $time_{with}$ and $time_{without}$ indicate the simulation time required by the SCERPA simulation of the circuit with and without the termination, respectively.

```
% inputs of the Majority Voter
Vout_MV = InOut_eval(BBcharPath,'majority_voter',
          [-0.94,-0.94,0.9264,0.9264,-0.94,-0.94]);
% output of the Majority Voter becomes input of the Inverter
Vout_inv = InOut_eval(BBcharPath,'inverter',
          [Vout_MV.Vout_B,Vout_MV.Vout_D]);
```

As it appears from Table 5.1, adding a termination in the case of a bus wire implies an improvement in terms of simulation time estimated around $50 \div 60\%$; on the other hand, the error between the output values appears to be relatively low for $Vout_B$ and $Vout_D$ and higher for $Vout_A$ and $Vout_C$.

Table 5.1: Comparison between the output values and the timing performances with and without adding the termination, for what concerns a bus wire.

(a) Table related to the output $Vout_A$.

Dr1	$Vout_A^{with}$	$Vout_A^{without}$	term_error (%)	Δt (%)
0	0.354	0.095	73.2	65.2
1	-0.764	-0.491	35.7	53.2

(b) Table related to the output $Vout_B$.

Dr1	$Vout_B^{with}$	$Vout_B^{without}$	term_error (%)	Δt (%)
0	-0.765	-0.65	15	65.2
1	0.354	0.347	1.97	53.2

(c) Table related to the output $Vout_C$.

Dr1	$Vout_C^{with}$	$Vout_C^{without}$	term_error (%)	Δt (%)
0	0.831	0.49	41	65.2
1	-0.346	-0.113	67	53.2

(d) Table related to the output $Vout_D$.

Dr1	$Vout_D^{with}$	$Vout_D^{without}$	term_error (%)	Δt (%)
0	-0.355	-0.325	8	65.2
1	0.825	0.73	11	53.2

These values are expected to be similar in the case of a XOR gate.

5.3 Simulation of the XOR gate

The XOR gate described in Figure 5.2 can be simulated on SCERPA, starting from the .qll file that describes its structure generated on MagCAD.

Before going into detail on the parameters of the simulation, it is important to highlight a specification related to the timing constraints of the layout. Figure 5.3 shows a zoomed overview of the central part of the circuit, where the first NAND gate has longer clock phases with respect to the wires that are located below and above it. This design choice is mandatory to guarantee that, after the signal leaves the last phase of both the branch connections, it gets simultaneously to the next NAND gates on the right: with this kind of layout, between the branches and the next NAND gates (the ones reached by the T-connection) there are 3 repetitions of clock phases, both in the central path and in the above and below ones.

That is the reason why the bus wires have been designed with a reduced length if compared to the dimensions of the NAND layout. Actually, the design choices relevant to this issue were two:

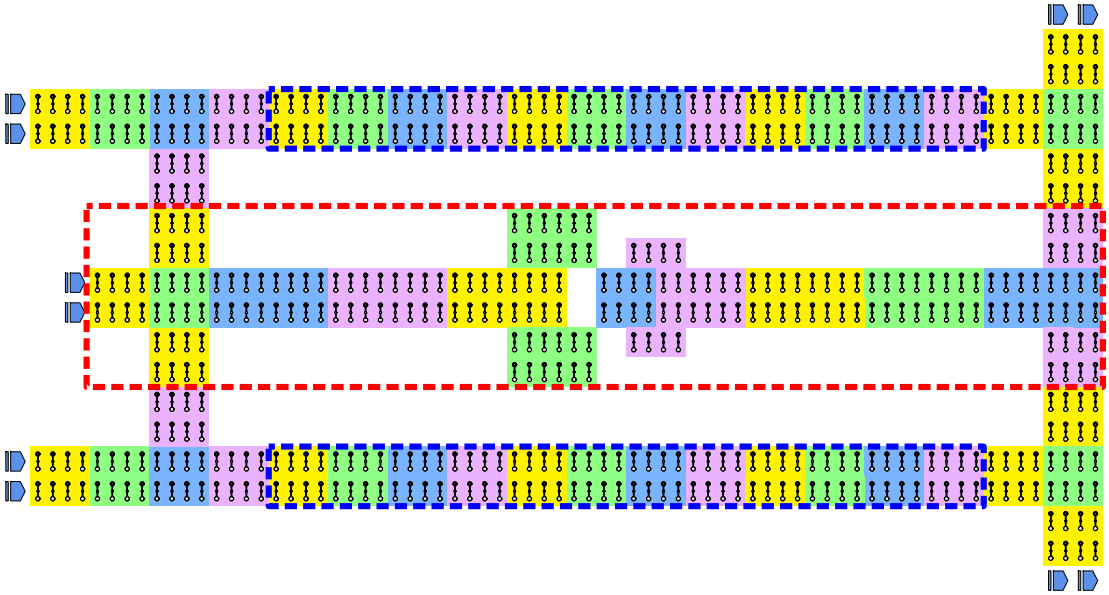


Figure 5.3: Zoomed region of the XOR layout. Both the red and the blue highlighted regions are made up by 3 phases repetitions.

- Reducing as much as possible the length of the bus wires;
- Stretching as much as possible the length of the NAND gate.

This necessity is generated by the constraint of delaying the signal on the wires: propagation there is faster than in the central part, due to the branch connection (before the NAND) and the T-connection (after the NAND) that favour a vertical propagation, thus slowing down the horizontal one that is typical of straight wires.
Figure 5.4 depicts how the clock phases repetitions are distributed along the circuit, highlighting the last phase of every repetition: it takes the input signal 9 repetitions to get to the end of the circuit.

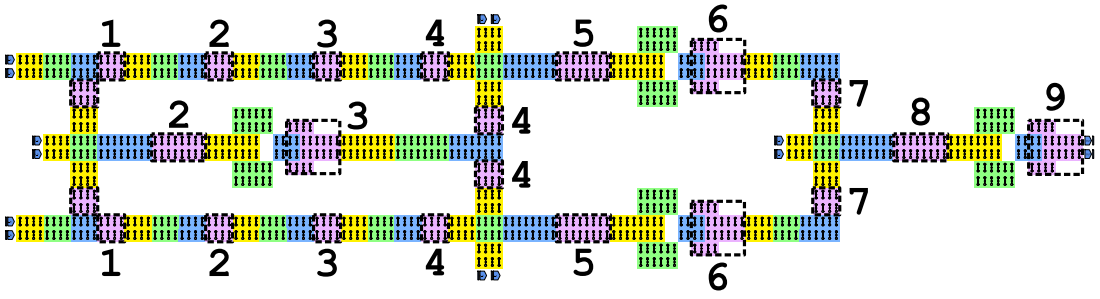


Figure 5.4: Distribution of the phases repetitions of the circuit.

After these specifications about the design choices, the simulation of the XOR can be carried out: the parameters used in it are exposed in 5.3. The parameter phasesRepetition

is set to 9, due to what explained in Figure 5.4, and enableTermination is set to 0 since the termination is not considered in this case.

```
driverPara.Ninputs = 3;
% inputs are DrA and DrB, while Dr0 is fixed to logic '0'
driverPara.driverNames = {'DrA','DrB','Dr0'};
driverPara.driverModes = {'0','0','0'};
driverPara.NsweepSteps = 1;
driverPara.NclockRegions = 4;
driverPara.phasesRepetition = 9;
terminationSettings.enableTermination = 0;
terminationSettings.busLayout = 1;
```

Listing 5.1: Parameters set to the SCERPA simulation of the XOR gate.

Although in 5.3 only the ['0','0','0'] combination of logic inputs is shown, all the 4 combinations must be tested on DrA and DrB, as in the case of the NAND gate, meaning that 4 simulations are needed in total.

5.4 Characterization of the XOR gate

Once the XOR gate has been entirely simulated on SCERPA, it is possible to make a comparison between the results of the simulation and those of the characterization process applied to the blocks assembled together (as in Figure 5.2).
Each block, before being characterized, has to be simulated on SCERPA as well. For the circuits with just one input either a 'sweep' or 'not_sweep' logic value is sufficient, since it gives a complete variety of values; for the Majority Voter, instead, the most optimised choice is to keep the Dr0 input to logic '0' and then simulate DrA and DrB with all the possible combinations of 'sweep' and 'not_sweep', in order to save all the possible values. The granularity is set to $N_{sweepsteps} = 4$ for every circuit.
Similarly to the case of the NAND gate, at the end of the characterization process the function InOut_eval.m is called for every block of the circuit in order, considering as input the output of the previous block(s) or directly the drivers according to the propagation of the signal.

Chapter 6

Verification of the process

After the outputs have been evaluated in the two ways, it is possible to make some comparisons between them in order to validate the proposed process and comment its results.

6.1 Effect of simulating more combinations in the same simulation

As mentioned in subsection 3.1.4, the possibility of simulating more combinations in the same simulation brings the important advantages of reducing the overall time overhead required by SCERPA and building complete libraries.
The reason behind the time difference with respect to simulating the combinations separately is that, for each simulation, the last array of which the rows of ValuesDr are composed[1] is simulated again. But since this array of values is always the same regardless of the quantity of input combinations, it is more convenient to simulate it just once.
Table 6.1 takes into consideration the two simulated circuits in which the effect is more evident, since they have more possible combinations of outputs: the Majority Voter and the XOR. On one side the table shows the sum of the time overheads of all the SCERPA simulations taken with a single combination of inputs, while on the other side it shows the time overhead when considering all the combinations into the same simulation.

Table 6.1: Comparison between the time overheads of the two approaches.

Circuit	One comb. at a time (s)	All comb. in one (s)	Difference (%)
Majority Voter	3397	1146	33.7
XOR gate	567140	241540	42.5

[1]the aforementioned *filler*, which takes into account the delay of the signal between the first cell and the last cell and depends only on the layout

As emerges from Table 6.1, the time overhead necessary to simulate all the necessary combinations at once is less than half of the time needed for simulate them all separately: this proves that the former approach is the more convenient one, since it also allows to obtain more complete libraries.

6.2 Comparison of the output values

Table 6.2 shows the output values of the XOR gate simulated alone on SCERPA (indicated with the superscript S) compared with the ones obtained through the characterization process (indicated with the superscript *char*), with the different combinations of inputs; Dr0 is not inserted in the table, since its logic value is fixed to '0'.

Table 6.2: Comparison between output values of the SCERPA simulation (without termination added) and output values of the characterization process.

DrA	DrB	$V^{S}_{out,A}$	$V^{char}_{out,A}$	$V^{S}_{out,B}$	$V^{char}_{out,B}$	$V^{S}_{out,C}$	$V^{char}_{out,C}$	$V^{S}_{out,D}$	$V^{char}_{out,D}$
0	0	0.089	0.362	-0.643	-0.821	0.487	0.822	-0.312	-0.329
1	0	-0.491	-0.76	0.352	0.345	-0.112	-0.356	0.734	0.858
0	1	-0.491	-0.76	0.352	0.345	-0.112	-0.356	0.734	0.858
1	1	0.089	0.362	-0.643	-0.821	0.487	0.822	-0.312	-0.329

It has to be remarked that the values in Table 6.2 refer to a simulation where the termination block has not been added, as specified in section 5.2. Since the **term_error** can be considered approximately equal for every kind of circuit considered in this study, the values of the XOR with an added termination can be predicted by applying the error difference of the case of the bus wire (Table 5.1) to the outputs evaluated without the termination.
The result is shown in Table 6.3, which finally shows the difference between the SCERPA simulation and the characterization both considering the presence of the termination.

Table 6.3: Comparison between output values of the SCERPA simulation (with termination added) and output values of the characterization process.

DrA	DrB	$V^{S}_{out,A}$	$V^{char}_{out,A}$	$V^{S}_{out,B}$	$V^{char}_{out,B}$	$V^{S}_{out,C}$	$V^{char}_{out,C}$	$V^{S}_{out,D}$	$V^{char}_{out,D}$
0	0	0.348	0.362	-0.758	-0.821	0.828	0.822	-0.342	-0.329
1	0	-0.764	-0.76	0.359	0.345	-0.345	-0.356	0.829	0.858
0	1	-0.764	-0.76	0.359	0.345	-0.345	-0.356	0.829	0.858
1	1	0.348	0.362	-0.758	-0.821	0.828	0.822	-0.342	-0.329

Table 6.3 shows that, with all the due approximations, independently on the design choice of adding a termination to the layout, the extrapolation of data from libraries guarantees an error with respect to the simulation data that corresponds, even in the worst case, to a value below the 92.3% of the actual value.

6.3 Simulation time overhead

Table 6.4 shows the total simulation time overhead required by SCERPA, t_S, for each different block, in the case where $N_{sweepsteps}$ is set to 4.
For what concerns the Majority Voter, the time overhead depicted in the table refers to the a single simulation where 4 combinations of inputs have been tested one after another. The total time required by the sum of all the simulations, t_S^{TOT}, is indicated in the last row of the table.

Table 6.4: Comparison between the average time overhead of the SCERPA simulation of the different blocks.

Circuit	t_S (s)
Bus wire	243
L-connection	207
Branch connection	373
T-connection	454
Inverter	449
Majority Voter	3397
t_S^{TOT}	5123

On the other hand, the time overhead required by the functions Characterization.m and InOut_eval.m to be executed (defined respectively as t_{Lib} and t_{InOut}) corresponds approximately to 3s each: this means that the amount of time necessary to obtain the outputs of a block (corresponding to the given inputs) after it has been simulated corresponds approximately to 6s.
Table 6.5 compares the average simulation time required by the SCERPA simulation of the "monolithic" XOR with the average time required by the Characterization process of all the components of the circuit[2], defined as t_{char}, for one combination of inputs.
For the XOR structure depicted in Figure 5.2 the time needed to characterize it all is evaluated with Equation 6.3, since there are 8 different kinds of blocks to be characterized and a total of 19 blocks.

$$t_{char} = 8 \times t_{Lib} + 19 \times t_{InOut} \simeq 81s \tag{6.1}$$

As a consequence, once the blocks have been simulated on SCERPA, it takes about 81 seconds to evaluate the output values of the XOR gate through the Characterization process.

[2]Here the Characterization process has to be intended as the execution of both characterization.m, to create the libraries, and InOut_eval.m, to browse them.

Table 6.5: Comparison between the time overhead required by the XOR SCERPA simulation and the average one required by the characterization of its different blocks (in the worst case).

XOR SCERPA overhead	t_{char}	Difference
2d19h5m	81s	0.03%

Chapter 7

Conclusions and future perspectives

The SCERPA simulation of all the kinds of blocks of the circuit is necessary for the purpose of the application of the Characterization process, despite representing the majority of the time overhead of a process that aims at overperforming SCERPA itself. However, since the blocks need to be simulated just once, every time the user needs to test a new combination of inputs it is only necessary to create the libraries from simulation results: while simulating again the same structure of the beginning would have the same duration of the first time, employing the Characterization alone represents a significantly faster method. As a matter of fact, the value obtained from Table 6.5 is related to the worst case in which every circuit has to be characterized for the first time.
The advantage of converting a complex circuit into a netlist-like system is that it enhances the reusability of the components, because a library-based system offers the possibility of simulating a multitude of input combinations without paying the exaggerate time overhead related to the SCERPA simulation of the whole structure every time.
Table 7.1 summarizes the advantages brought by the method.

Table 7.1: Numerical advantages brought by the Characterization tool in terms of average accuracy (with respect to SCERPA values) and time overhead (duration of the process compared to the SCERPA one).

Accuracy (% average)	Accuracy (% worst case)	time overhead (% worst case)
96.63	92.3	0.03%

The concepts and the results applied to the XOR in this project can be considered for sure as a starting point for the research of future potential developments. Exploiting the reusability of the components is a precious feature that leads to the construction of even more complex structures, for example multiplexers, full-adders, Ripple-Carry Adders and other digital circuits that can be built by assembling simple gates as the ones described until now, including the XOR itself.

Some further developments related to the physical realization of the circuit could be represented by the study of a Safe Operating Area (SOA) or the evaluation of its power consumption.

Bibliography

[1] Yuri Ardesi, Ruiyu Wang, Giovanna Turvani, Gianluca Piccinini, and Mariagrazia Graziano. Scerpa: A self-consistent algorithm for the evaluation of the information propagation in molecular field-coupled nanocomputing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2749–2760, 2020. doi: 10.1109/TCAD.2019.2960360.

[2] Yuri Ardesi, Giovanna Turvani, Mariagrazia Graziano, and Gianluca Piccinini. Scerpa simulation of clocked molecular field-coupling nanocomputing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(3):558–567, 2021. doi: 10.1109/TVLSI.2020.3045198.

[3] Umberto Garlando, Fabrizio Riente, Deborah Vergallo, Mariagrazia Graziano, and Maurizio Zamboni. Topolinano & magcad: A complete framework for design and simulation of digital circuits based on emerging technologies. In *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 153–156, 2018. doi: 10.1109/SMACD.2018.8434919.

[4] Umberto Garlando, Fabrizio Riente, and Mariagrazia Graziano. Funcode: Effective device-to-system analysis of field-coupled nanocomputing circuit designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(3):467–478, 2021. doi: 10.1109/TCAD.2020.3001389.

[5] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, 2006. doi: 10.1109/N-SSC.2006.4785860.

[6] A. O. Orlov, I. Amlani, G. H. Bernstein, C. S. Lent, and G. L. Snider. Realization of a functional cell for quantum-dot cellular automata. *Science*, 277(11):928–930, 1997.

[7] Azzurra Pulimeno, Mariagrazia Graziano, and Gianluca Piccinini. Molecule interaction for qca computation. In *2012 12th IEEE International Conference on Nanotechnology (IEEE-NANO)*, pages 1–5, 2012. doi: 10.1109/NANO.2012.6322051.

[8] Azzurra Pulimeno, Mariagrazia Graziano, Alessandro Sanginario, Valentina Cauda, Danilo Demarchi, and Gianluca Piccinini. Bis-ferrocene molecular qca wire: Ab initio

simulations of fabrication driven fault tolerance. *IEEE Transactions on Nanotechnology*, 12(4):498–507, 2013. doi: 10.1109/TNANO.2013.2261824.

[9] Azzurra Pulimeno, Mariagrazia Graziano, Ruiyu Wang, Danilo Demarchi, and Gianluca Piccinini. Charge distribution in a molecular qca wire based on bis-ferrocene molecules. In *2013 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 42–43, 2013. doi: 10.1109/NanoArch.2013.6623041.

[10] M. Roser and H. Ritchie. Transistor count. 2020. URL `https://ourworldindata.org/technological-change`.

[11] Ruiyu Wang, Michele Chilla, Alessio Palucci, Mariagrazia Graziano, and Gianlucca Piccinini. An effective algorithm for clocked field-coupled nanocomputing paradigm. In *2016 IEEE Nanotechnology Materials and Devices Conference (NMDC)*, pages 1–2, 2016. doi: 10.1109/NMDC.2016.7777166.

[12] H. Wong. On the cmos device downsizing, more moore, more than moore, and more-than-moore for more moore. In *2021 IEEE 32nd International Conference on Microelectronics (MIEL)*, pages 9–15, 2021. doi: 10.1109/MIEL52794.2021.9569101.