POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

EDA tools for emerging nanotechnologies: from layout to VHDL

Relatori:
Prof. Maurizio Zamboni
PhD Fabrizio Riente

Candidato:
Riccardo Chiola

July 2019

Table of contents

List of figures

Chapter 1

Introduction

As the scaling speed of CMOS technology projected by Moore's law is slowing down due to physical limitations, new technologies, called Beyond CMOS, are undergoing scrutiny to find candidates to allow further progress when the limits will be finally reached. Among these emerging devices, field-coupled technologies are considered promising thanks to their low power consumption and the possibility of integrating logic and memory elements. Based on this principle, many implementations exists, in particular, three have been considered during this work: in-plane Nano Magnetic Logic (iNML), perpendicular Nano Magnetic Logic (pNML) and Molecular QCA.
Since these technologies are at an early stage of the development, the availability of CAD software is still very limited due to the absence of commercial tools. For this reason, at the VLSI laboratory at the Politecnico di Torino, a software framework has been developed, providing a flexible set of tools capable of helping in the design, simulation and verification of circuits based on these emerging nanotechnologies.
The framework is composed of two software: ToPoliNano and MagCAD. ToPoliNano, which gives the name to the framework itself, is able to perform simulations of layouts and to generate them by parsing a VHDL netlist, while MagCAD's function is to provide an environment in which the user can graphically design custom circuits and generate their associated VHDL netlists (figure 1.1).

The overall purpose of this thesis has been the expansion of the features of MagCAD: this has been achieved by working on separate objectives, affecting different elements of the software.
The first part of the thesis is related to the development of a new, general algorithm for the detection of functional blocks within the design starting from basic element connectivity. Starting from custom designs based on simple nanomagnets, the algorithm is able to identify basic gates according to the signal flow directionality

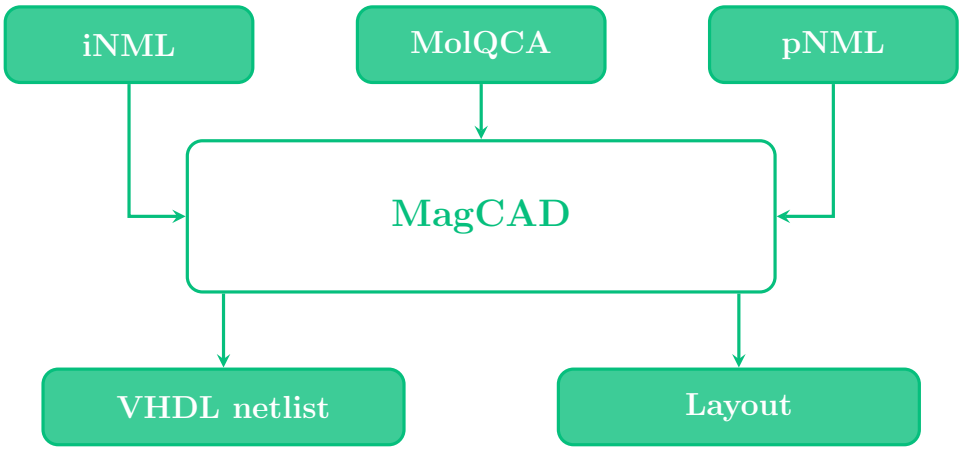


Figure 1.1: MagCAD structure

defined by input/output pins. The detection of the functionality is a fundamental step toward the generation of the final VHDL netlist describing the circuit.
The algorithm has been verified through the implementation of an automated test environment. A database of custom circuits having different complexity has been defined and the capability of determining the propagation direction in case of ambiguous paths has been verified.
The second part of the thesis was focused on the development of an algorithm able to report critical paths within hierarchical pNML circuits.
This is crucial when dealing with long interconnection wires that starts from an higher level of the hierarchy and terminate in some inner block. The VHDL generation algorithm has been extended to include this functionality. Similarly, the automated test environment has been extended to the pNML technology. A circuit database has been developed covering all critical cases.
The last part has been focused in the implementation of the MagCAD plugin for Molecular QCA. In particular, the basic molecules and their physical properties have been defined. This features make it possible to design custom molecular circuits based on BisFerrocene that can be simulated exploiting the SCERPA simulator developed by the VLSI group.
In addition to this, other minor features were added, such as the ability to organize pins in vectors and the ability to change the size of individual iNML magnets, with the purpose of making the tool more versatile and the user experience more pleasant.

Chapter 2

Technological Background

2.1 QCA Logic

A cellular automaton (CA) is a system defined as a grid of cells which can be in one of a finite number of states at a discrete time and the rules that determine how the system evolves as time passes. The rules that determine the state of each cell consider only the cell previous state and the states of the neighboring cells. The first physical implementation of the cellular automaton theory is called Quantum dot Cellular Automata (QCA) because it uses as cells an array of quantum dots connected by tunnel coupling. With 4 quantum dots placed at the corners of a square, electrostatic repulsion pushes electrons to occupy opposite corners, creating 2 states that can be seen as a logic 0 and 1.

(a) Logic 0 (b) Logic 1

Figure 2.1: QCA states

The system can't evolve on its own, because the interaction between cells is not strong enough to force a cell outside of a stable state, so an external force must be applied. This external force is the clock of the QCA, which forces a part of the cells in an unstable state, so that when it is removed the cells align themselves based on their stable neighbors. To regulate the flow of the information, the circuit is then divided in zones that receive the clock signal with a different phase. Usually 3 or 4

phases are used, and if the clock is properly confined inside its own zone it's possible to make logic signal propagate in the desired way.

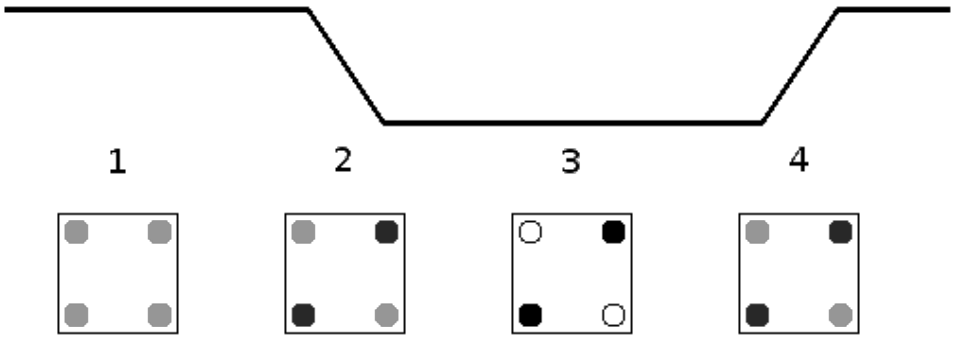
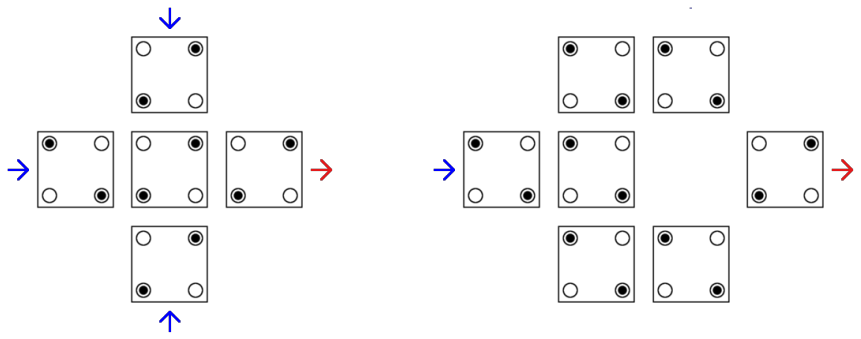


Figure 2.2: QCA clock

Having defined inputs and outputs in stable and unstable cells, logic functions can be obtained by placing a cell with 3 neighbors as input. This cell is called a Majority Voter and its output assume the value 1 only if at least 2 of the inputs are 1. By forcing one of the inputs at 0 a AND gate can be obtained, and similarly, an OR gate is a Majority Voter with an input fixed at 1. The NOT function is obtained by making cells interact diagonally, forcing an inversion of the input.



(a) QCA majority voter (b) QCA inverter

Figure 2.3: QCA logic gates

It's important to note that every QCA cell is naturally a memory element, and every transition between clock zones acts like a buffer, making QCA circuits intrinsically pipelined.

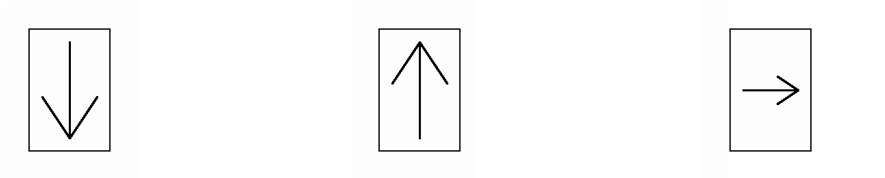
2.2 Nanomagnetic Logic

Nanomagnetic logic (NML) is an implementation of the QCA model where the cells are nanomagnets whose magnetization has 2 stable states. There are 2 variants:

- in-plane Nano Magnetic Logic (iNML) [1]
- perpendicular Nano Magnetic Logic (pNML) [2–4]

2.2.1 iNML

In the iNML technology, the basic elements are single domain magnets with a rectangular shape and typical dimension of 90nm x 60nm x 20nm. The magnets are anisotropic and their magnetization is only stable in the directions parallel and antiparallel of the long side of the magnet.



(a) Logic 0 (b) Logic 1 (c) Unstable state

Figure 2.4: iNML states

As with the quantum dot technology, iNML also need a clock signal to destabilize the magnets and allow signal propagation. Since the elements are magnets, the clock takes the form of an external magnetic field which is generated by currents flowing through wires placed under the plane of the magnets and is confined by means

of a ferrite yoke. An example of the working of a 3 phases clock is shown in figure 2.5.

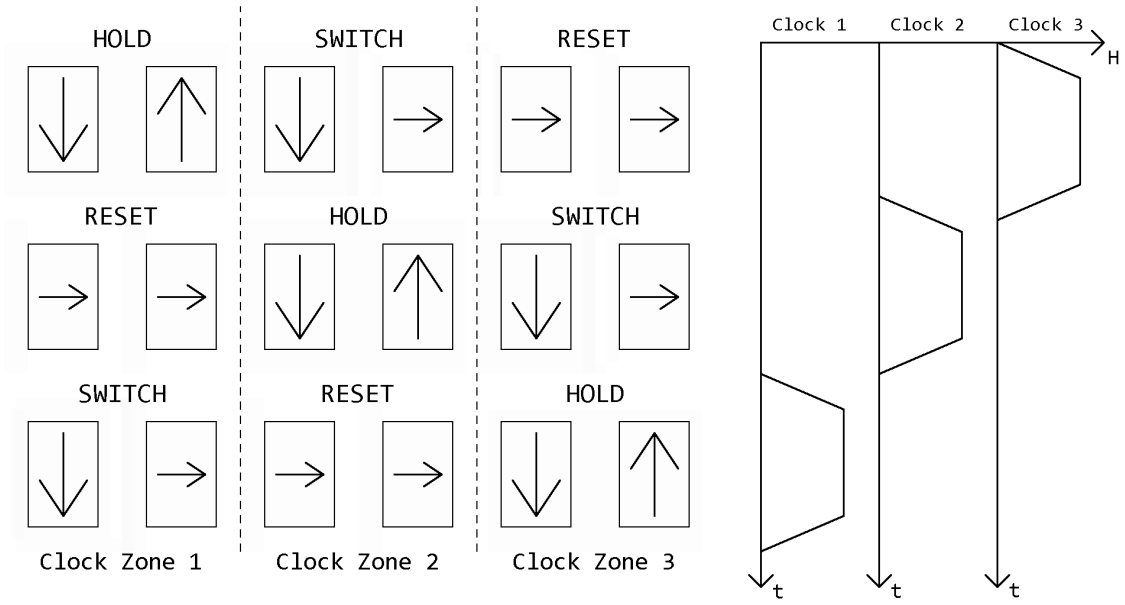
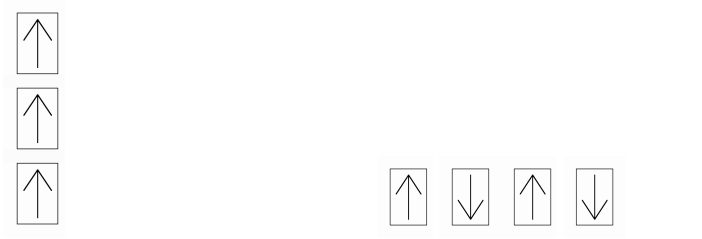


Figure 2.5: iNML clock

Due to the magnets rectangular shape, cascading them horizontally or vertically is not indifferent: when placed with their short sides adjacent, the interaction is weaker, so they need to be placed closer and less of them can be placed inside the same clock zone. As a result, the preferential direction of wires is the direction perpendicular to the magnets polarization, which means that the coupling between the magnets is antiferromagnetic and the signal gets naturally inverted (see figure 2.6.b). For that reason, the number of magnets inside a clock zone is usually chosen to be even, typically 4 or 6, resulting in an even number of inversions which equates to no inversion at all.

AND and OR logic gates are not made using majority voters because it's possible to shape magnets in special ways to obtain elements that favor 0 or 1 when the inputs are different, as shown in figure 2.7.

The iNML technology has limited clock speed, but it has very low switching



(a) Ferromagnetic coupling (b) Antiferromagnetic coupling

Figure 2.6: iNML coupling

(a) AND gate (b) OR gate

Figure 2.7: iNML logic gates

power consumption and no leakage currents, in addition to the possibility of fabrication with current processes.

2.2.2 pNML

The basic cell of pNML technology is made of a stack of Co/Pt and his magnetic anisotropy is perpendicular to the plane of the cell. Thus, the stable states of the magnetization are out-of-plane.

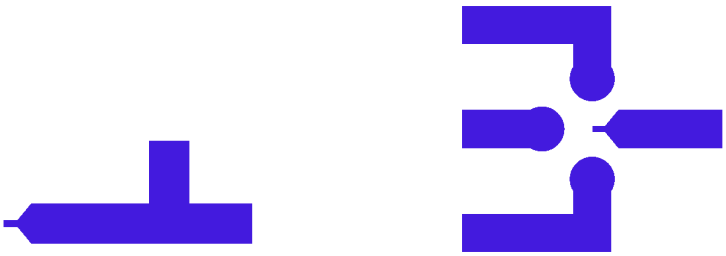
To propagate the signal, the magnets are made with an area more sensitive to the external magnetic fields, called nucleation center, where we want the magnet to have an input. In pNML there are no clock zones: the clock signal is a periodic perpendicular magnetic field administrated to the entire circuit and its role is to provide the additional energy required to overcome the nucleation threshold and trigger the switch, which propagates from the nucleation center along the magnetic domain.

Wires are not made with chain of magnets, but with a domain wall: a long magnetic

(a) Logic 0 (b) Logic 1

Figure 2.8: pNML states

nanowire with a single domain that propagates the signal. If the domain wall has branches, it propagates the signal to all of them, making it possible to drive multiple gates.
Majority voters are not present in pNML because the interaction between magnets is antiferromagnetic, there is instead a minority voter which can be used to create NAND and NOR by fixing one input to 0 or 1.



(a) Example of domain wall (b) Example of minority voter

Figure 2.9: pNML structures

The propagation of the signal through a domain wall can be stopped by using variations in the geometry of the magnet to create barriers that require an in-plane magnetic field to be overcome. This field has to be provided in addition to the clock and allows finer control of the flow of the signal through the wires.

Another interesting characteristic that makes the pNML technology interesting is the ability to create monolithic 3D structures to make compact circuits. The magnets can be placed in different layers and can communicate through magnetic

Figure 2.10: pNML notch

vias which allow vertical coupling between magnets. The vertical coupling, unlike its horizontal counterpart, is ferromagnetic, so special care has to be considered when creating voters with inputs on different layers.

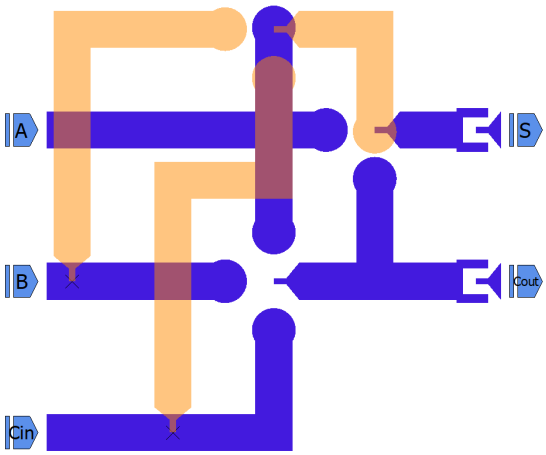


Figure 2.11: pNML 1-bit full adder

2.3 Molecular QCA

Molecular QCA cells are realized with pair of molecules with 3 quantum dots each to store the charges, 4 of the dots of the cell are used to encode the logical information, and the remaining 2 are used for the null state that enables the transition between states. The charges are confined in the cell and so no currents are involved in the switching [5].

The design of Molecular QCA circuits follows the same principles outlined in the QCA section: a clock is needed to enable the switching and the circuit has to be divided into clock zones with different clock phases to define the direction of the

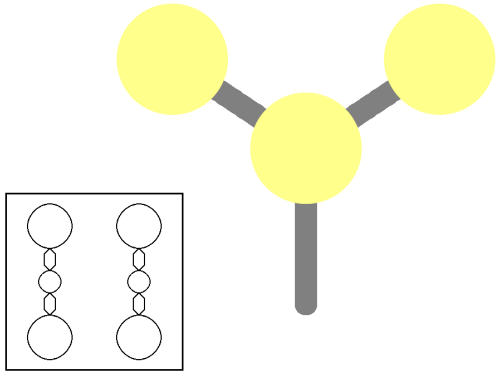
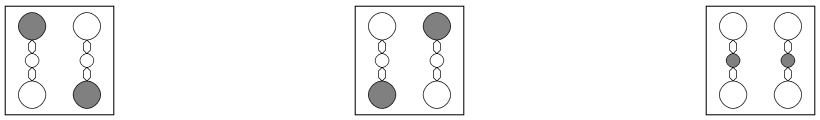


Figure 2.12: MolQCA cell and molecule model

signal propagation. The basic logic gates are again the majority voter, which can be used to obtain the AND and OR functions, and the inverter, which exploits oblique coupling to perform the NOT function.



(a) Logic 0 (b) Logic 1 (c) Unstable state

Figure 2.13: MolQCA states

This technology is characterized by an high switching frequency, which coupled with the low power consumption makes it particularly attractive among the emerging technologies.

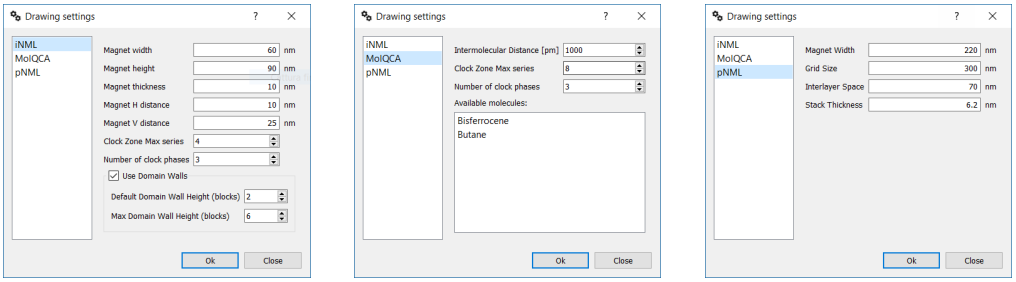
Chapter 3

MagCAD

MagCAD is a software used to design logic circuits based on QCA technologies [6,7]. The main goal of this work was to improve the functionality of MagCAD by adding new features and improve the existing ones, so before continuing it's important to understand what it can do and how it works.

3.1 Overview

MagCAD can design circuits based on multiple technologies. The available technologies are included under the form of plugins, in order to make the software universal and extensible. It currently supports iNML, pNML and Molecular QCA.



(a) iNML (b) MolQCA (c) pNML

Figure 3.1: MagCAD drawing settings

When MagCAD is started, it immediately presents to the user the Drawing Settings window, shown in figure 3.1, which is used to create a new drawing by selecting the technology and changing various physical parameters. After confirming the creation of the new drawing, the main window of MagCAD appears, as shown in figure 3.2. The creation of the layout is then performed graphically by selecting and placing the desired elements in the drawing area. In addition to the pins working as the

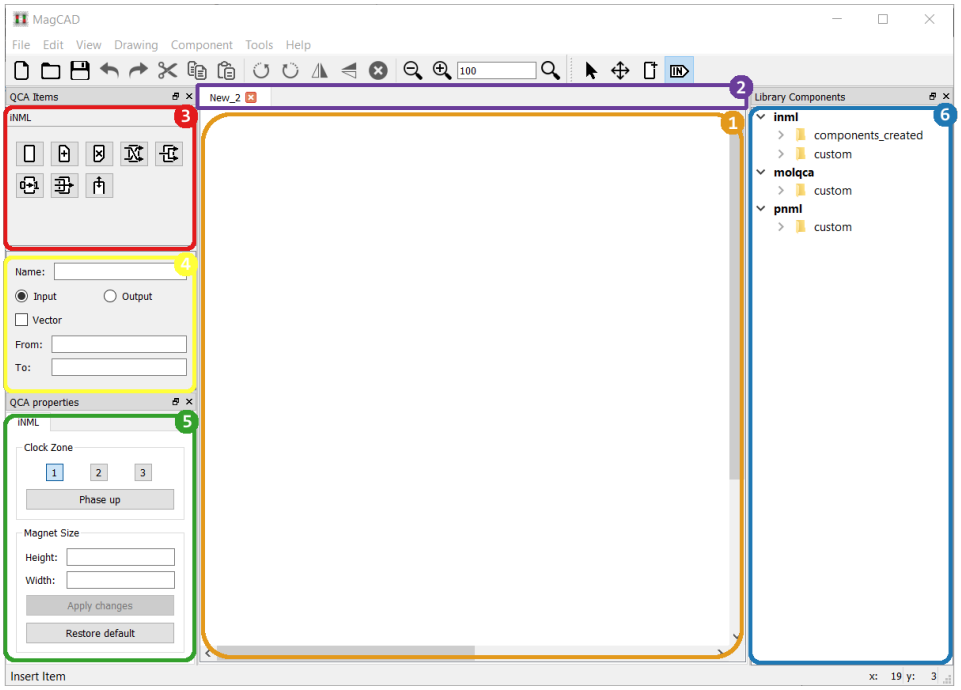


Figure 3.2: MagCAD interface

1) Drawing area; 2) Open tabs; 3) Item selection; 4) Pin creation properties; 5) Selected item properties; 6) Available components

input and output of the circuit, every technology has its own set of items that can be used, and it's possible for the circuit to be fully hierarchical, using previously saved circuits as components.
The layouts are saved in the .qll file format, which is an XML file listing all the settings for the technology used and all the items placed in the design. When choosing to export a layout as a component, a second file is created, in .qcc format, which is also an XML file containing the list of items, but without the settings and with the pins separated from the rest of the items because they are used to define the connection points of the component.
Finally, for the plugins that support it, which currently are both the NML technology ones, when exporting the component MagCAD can also generate a netlist of the layout and write it in the form of a VHDL description that can be used for simulations.

The generation of the VHDL description requires the program to correctly link

Figure 3.3: Circuit pins

A red arrow indicates an output pin, a blue arrow an input pin.

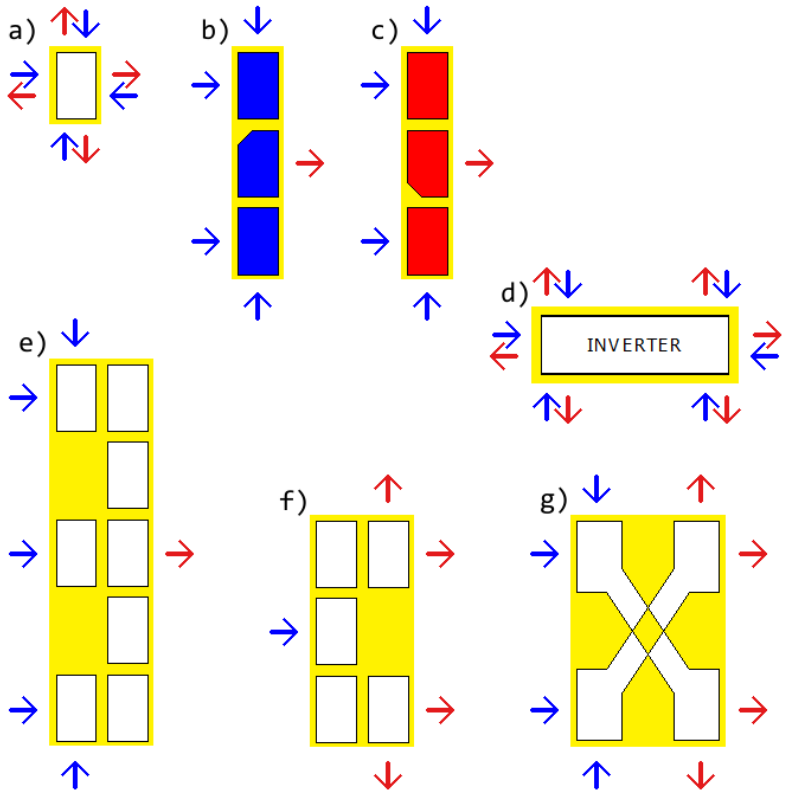


Figure 3.4: MagCAD iNML items

The coupler and the majority voter are legacy items that have been functionally replaced by the magnet.

a) Magnet; b) OR gate; c) AND gate; d) Inverter; e) Majority voter; f) Coupler; g) Crosswire

the placed items respecting the direction of the signals: this is done by giving every item input and output pins that define the position and the direction of the possible

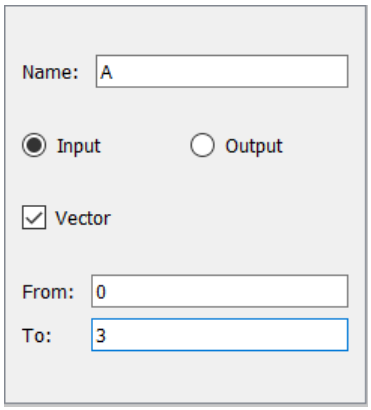
connections and then by going through the elements and linking every input pin of an item with the output pin of a neighboring item.
It's important to keep in mind that the input and output pins of the circuit are items, and as such they also have the aforementioned item pins, but the input pins only have an output pin and vice versa, as shown in figure 3.3. This is because the input pin needs to output the signal it receives from outside to the element that follows and similarly the output pin needs an input pin to receive the signal that is then relayed to the outside.
Figure 3.4 shows the items of the iNML technology and the position of their pins. With the exception of the magnet, which can serve multiple logic functions, when an element can connect an input/output in multiple directions, only one can be used at a time and any attempt to do otherwise results in an error.

3.2 Common features

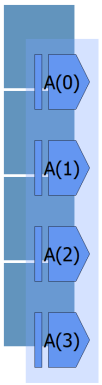
MagCAD itself has been the subject of some improvements of existing features during the course of this work: the introduction of vectors of pins and the streamlining of the definitions of the drawing settings.

3.2.1 Pin vectors

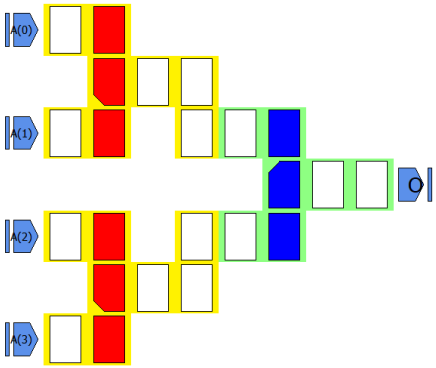
For MagCAD, every IO pin inserted in the design is a separate item with no relation to any other pin. For the designer, on the other hand, multiple pins can be the different bits of a single input, and it's desirable to have them represented as such when exporting the layout.
For this reason, MagCAD has been instructed to detect inputs named "*characters*(*number*)" and write them as a std_logic_vector "*characters*(*max_number* downto *min_number*)" in the VHDL files.
Furthermore, the pin insertion interface inside MagCAD has been modified to simplify the creation of pin vectors: checking the box "Vector" and entering numbers in the fields "From" and "To" generates in a single click an array of pins with the



(a) Pin insertion interface



(b) Placement of the pins in the drawing area



(c) Example circuit

```
1  entity iNML_custom_PinVectorsExample is
2  port(
3    O: out std_logic;
4    O_param: out param_data := (others => 0.0);
5    A: in std_logic_vector(3 downto 0);
6    A_param: in param_data_vector(3 downto 0) := (others
         => (others=>0.0));
7    CLK: in std_logic_vector(0 to 2));
8  end iNML_custom_PinVectorsExample;
```

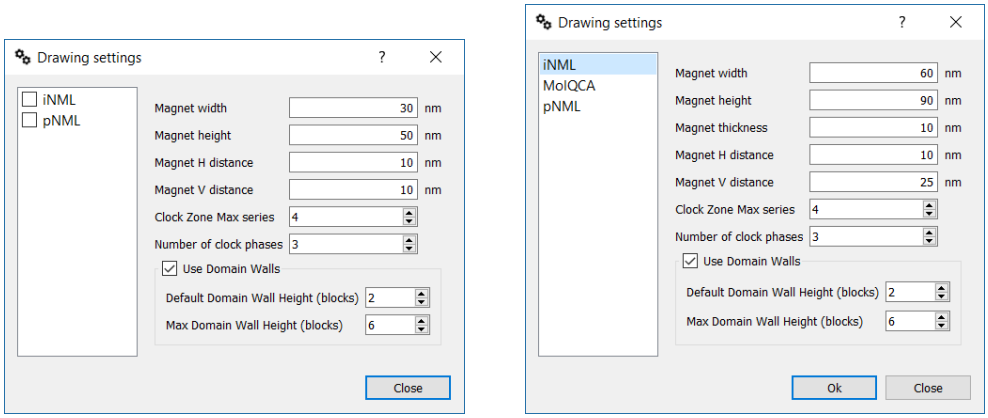
(d) Generated VHDL entity

Figure 3.6: Pin vector creation

correct names.

3.2.2 Drawing setting streamlining

To facilitate the creation of new drawings, the drawing settings window has been enhanced and made more natural to use by the addition of the "Ok" button and the removals of checkboxes in the technology selection panel.



(a) Before (b) After

Figure 3.7: Drawing settings comparison

The new "Ok" button has inherited the functions previously held by the "Close" button, i.e. the creation of a new drawing with the selected settings, while the "Close" button now closes the windows without doing anything.
The removal of the checkboxes has been accompanied by a change in the way the technology get chosen: now the technology used is the one selected, which means that the settings currently shown are the ones that get used when the new drawing gets created, as one would expect.
These changes have also made it possible to add a default selection to the technology panel, so that an inattentive user can't forget to select the technology and have to input the settings again after being reminded by the program that a drawing can't be created without a technology.

Chapter 4

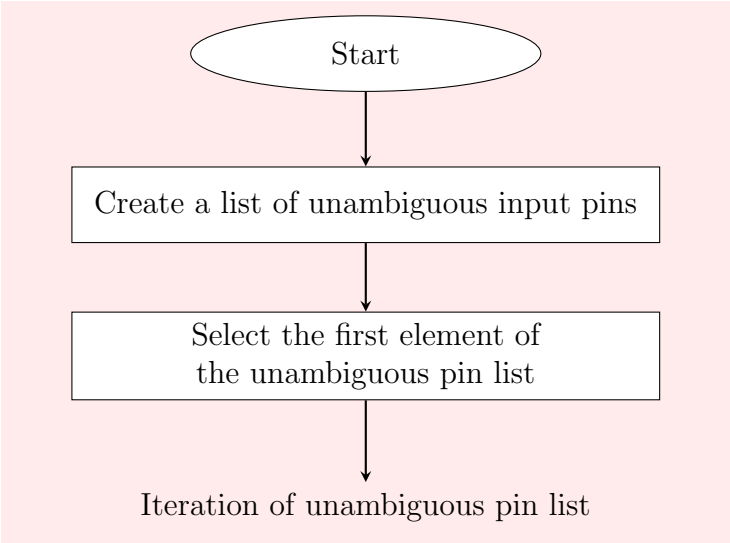
iNML plugin

4.1 Automatic functionality detection algorithm

Only for some elements the position of inputs and outputs is predetermined, the others get their direction defined as they get linked with their neighbors. Of particular interest are iNML simple magnets as none of their pins have a direction known from the start, and because the number of inputs and outputs is not known a priori: if provided with 1 input and 1 output they act as a wire; if connected with 1 input and 2 or 3 outputs they work as a coupler; finally, if fed 3 inputs, they work as a majority voter.
The goal of this work is thus to provide an algorithm to correctly perform the generation of the netlist of an iNML layout.
The basic idea is to start from the pins of elements with univocal direction, such as those at the border of the circuit and the pins of logic gates, and then follow the signal direction, linking elements along the way, until either the design has been fully connected or some ambiguous elements has been found.
Any time the algorithm reaches a magnet with more than 2 adjacent magnets, it has to put it aside and determine which of his neighbors are inputs and which are outputs to know in which way to propagate the signal.
To do so it uses a recursive function that takes a pair of elements, checks their clock zones and if they are different it uses that information to define the direction, otherwise, it moves one element forward and repeats the operation until it either reaches another ambiguous element or it discovers the direction.
After all the neighbors have been analyzed it's at last possible to use the number of inputs and outputs found to guess the function of the element and deduce the direction of any ambiguous signal to perform the links.

4.1.1 Algorithm details

Figure 4.1: Preliminary steps

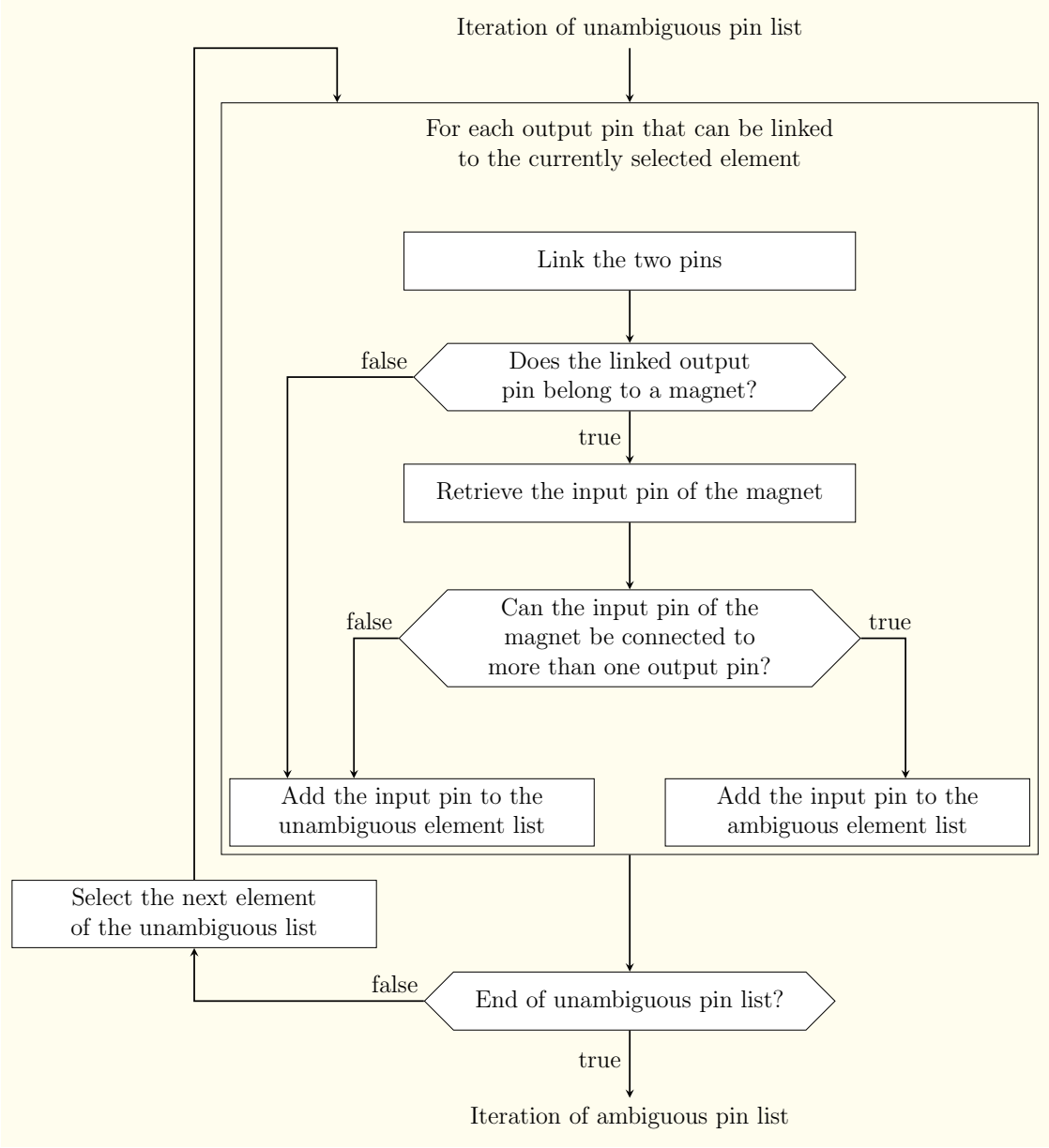


The first part of the algorithm is represented in the flowchart 4.1 and consists in the creation of the list of the unambiguous elements. This list serves as the starting point for the signal propagation. To create the list, all the input pins of the elements of the layout are checked and those that don't belong to a magnet or an inverter are inserted into the unambiguous pin list.

The second part is the iteration of the unambiguous pin list, shown in the flowchart 4.2. Every input pin of the list gets connected with a neighboring output and then the element to which the output belongs is analyzed: if it's not a magnet, the input pins of the element are added to the unambiguous pin list; if it is a magnet, its neighbors are also considered: if there is only one neighbor then it's possible to conclude that the magnet is used as a wire and the input is added to the unambiguous pin list, otherwise it's necessary to determine whether it's a coupler or a majority voter and it's added to the ambiguous pin list.

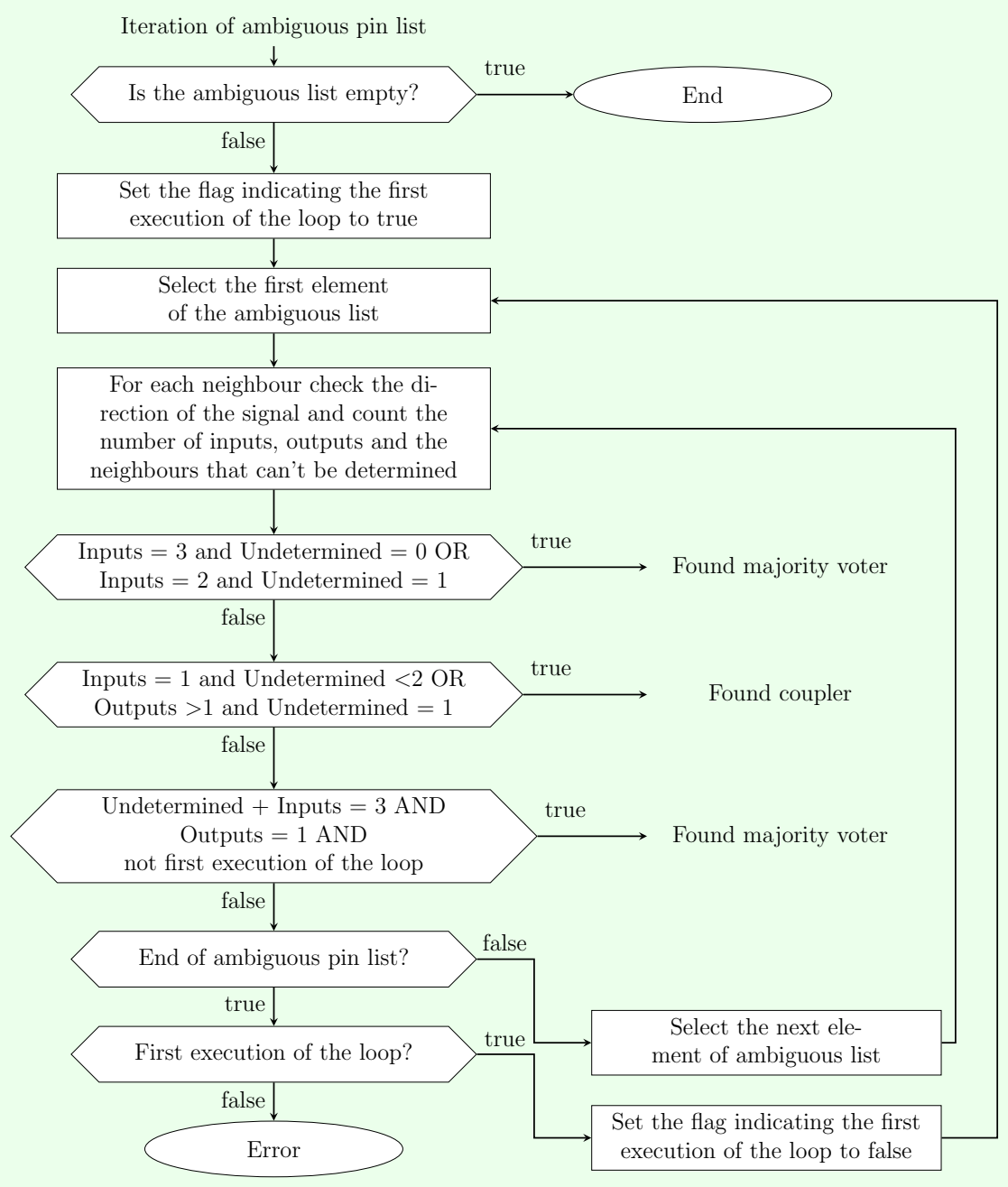
Once the end of the unambiguous pin list is reached, it's time to see if there are elements in the ambiguous pin list. This is shown in the flowchart 4.3. If the list is not empty, then it is also iterated to solve the ambiguities.

Figure 4.2: Iteration of unambiguous pin list



Every element in the list is a magnet that have more than 2 elements that it can connect to. For each of these elements, the algorithm tries to determine if it's an input or an output of the ambiguous element under scrutiny. If the direction can't be resolved, the connection is counted as undetermined. This happens when the

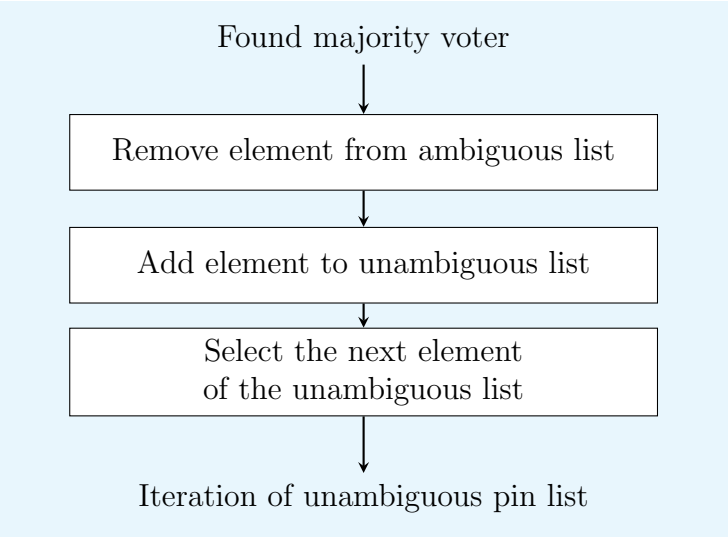
Figure 4.3: Iteration of ambiguous pin list



exploration of the circuit finds another ambiguous element. Undetermined connections do not necessarily prevent the algorithm from finding the correct function of the magnet: knowing that an even number of inputs is impossible (2 inputs are not

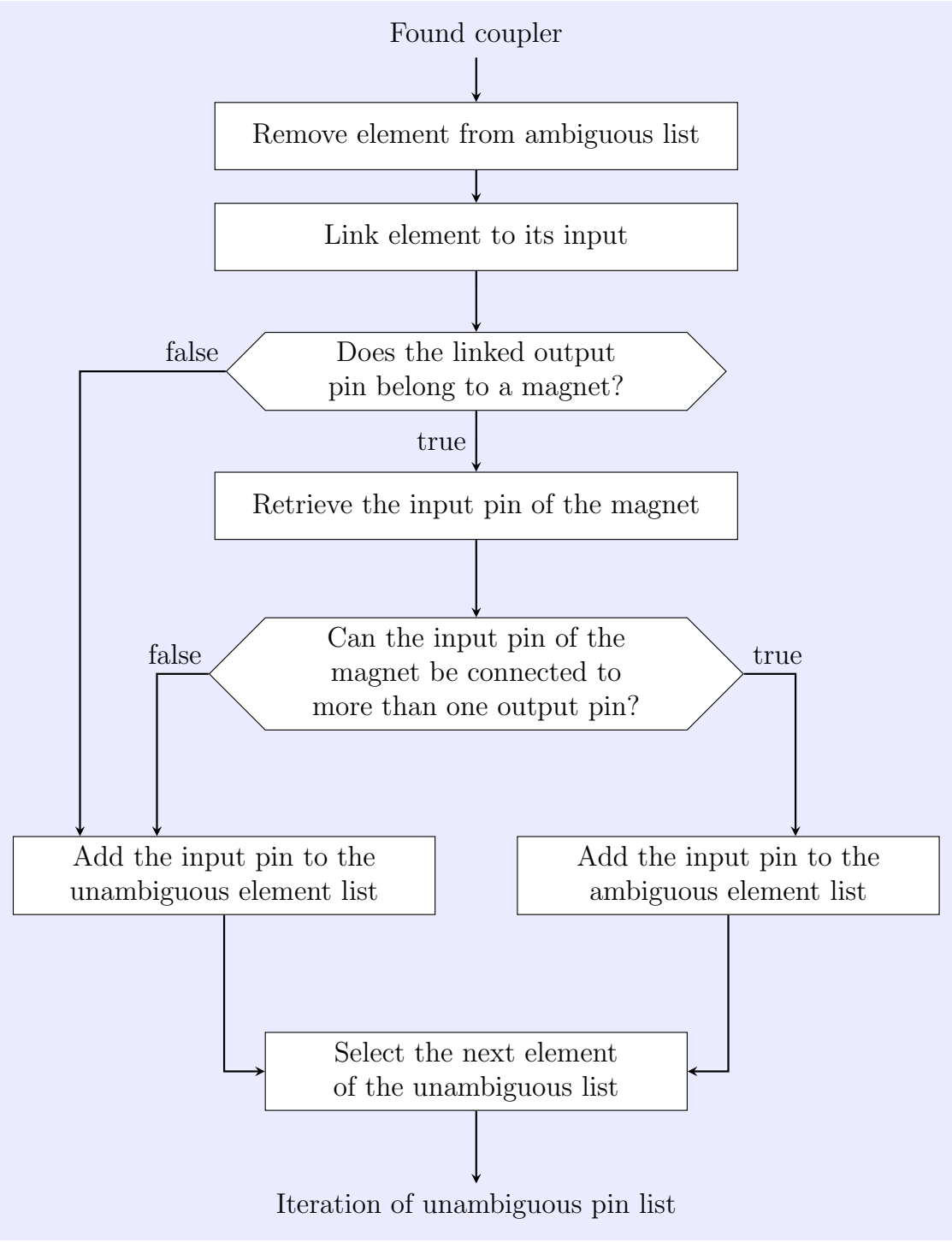
enough for a majority voter, 4 inputs would mean no output) allows the algorithm to guess whether the undetermined connection is an input or an output based on the others.
More precisely, the magnet is a majority voter if it has 3 inputs and 1 output, or if it has 2 inputs, 1 output and 1 undetermined connection. It's instead a coupler if it has 1 input and 0 or 1 undetermined connections or if it has more than one output and exactly 1 undetermined connection. In this second case it's necessary to have only 1 undetermined connection because if a magnet has 2 outputs and 2 undetermined connections we can't decide which of the undetermined connections is the input of the coupler, so we are still unable to connect it.
Finally, there is a third condition that is only used if the previous 2 have failed for all of the ambiguous magnets, that is considering anything with exactly 1 output and 3 other neighbors a majority voter. This is riskier, because a coupler can also meet these conditions, for this reason the algorithm first tries to solve one ambiguity with the other 2. If this fails and the end of the ambiguous list is reached, the list is iterated again with this third condition enabled.

Figure 4.4: Found majority voter



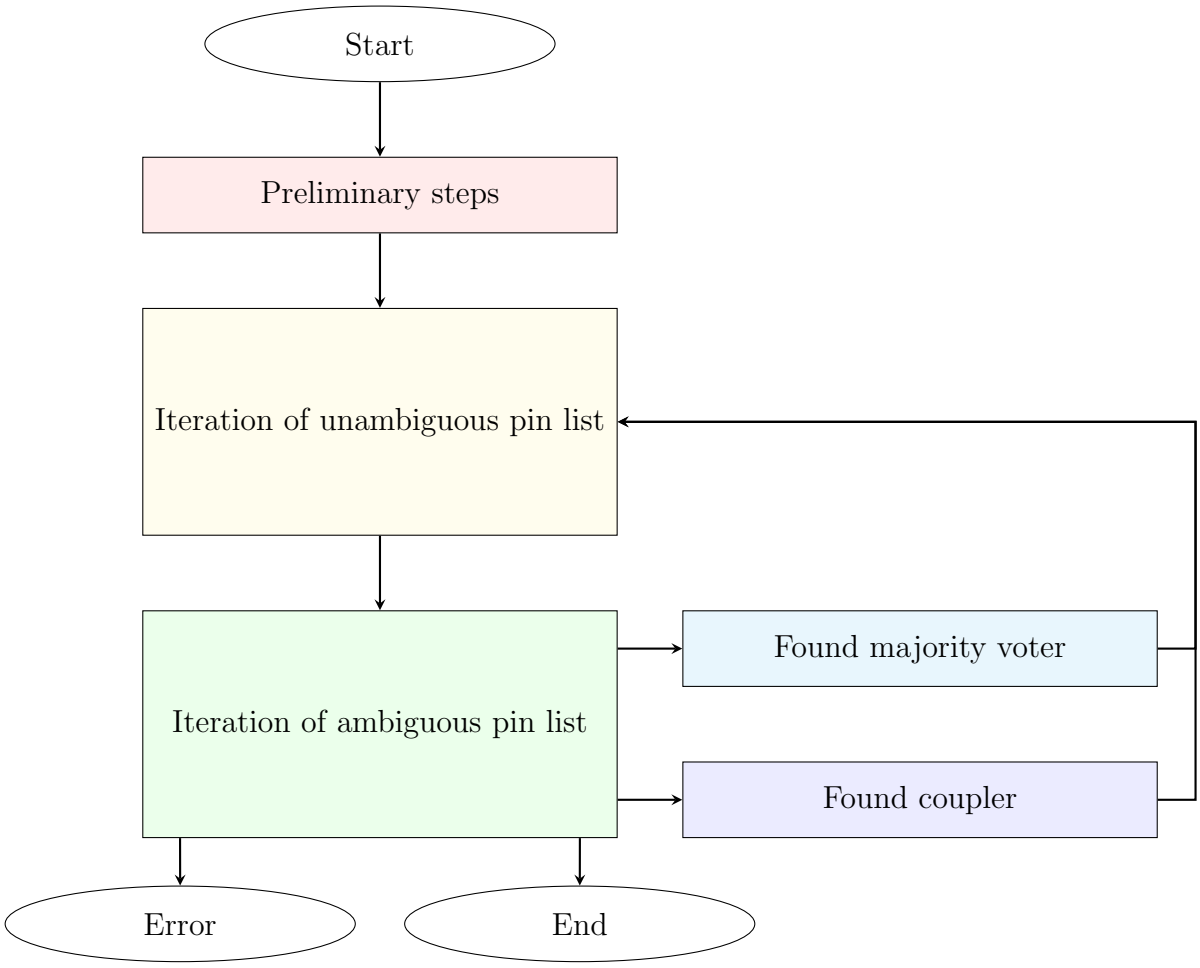
Once the function of any ambiguous element is found, the element is removed from the ambiguous list and the iteration is stopped. If the found element was a

Figure 4.5: Found coupler



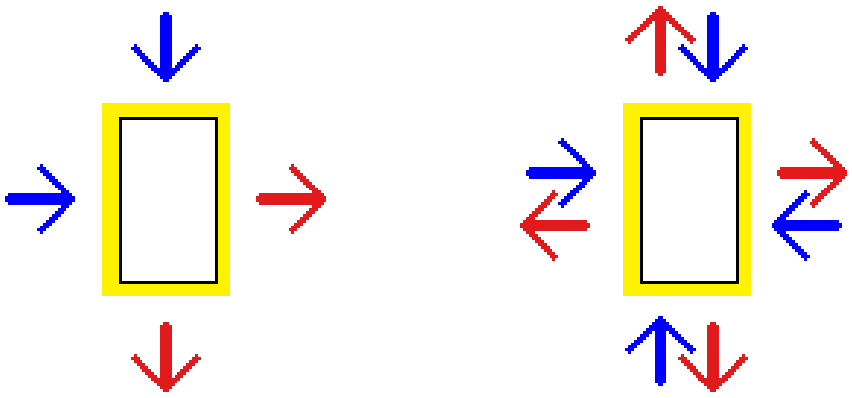
majority voter, the element is just added to the unambiguous list to be connected with its inputs, as shown in flowchart 4.4, if it was a coupler, the element is connected to its input and the input is added the the ambiguous or unambiguous list depending on the number of its possible connections, as shown in flowchart 4.5. Then the algorithm goes back to the unambiguous pin list to connect the pins that were possibly added, in order to make the next iteration of the ambiguous list easier.

Figure 4.6: Complete algorithm



4.1.2 Advantages

Before the development of the new connection algorithm, the magnet item had predetermined input pins on the left and top sides, and output pins on the right and bottom sides. This was necessary because the old algorithm linked input and output pins without performing additional checks, meaning that every item could only support predefined connections. The result of this limit was the necessity to flip pins horizontally or vertically in the case of wires that needed to propagate the signal from right to left or from the bottom to the top, and the necessity of having dedicated items to perform the roles of couplers and majority voters. This issues were made worse by the inability to flip multiple items at once, the symmetrical shape of the magnets, which made it impossible to determine at a glance whether they had been flipped or not, and the fixed shapes of the coupler and the majority voters, which made certain layouts outright impossible.
The ability to automatically determine the function of a magnet given to MagCAD by the new algorithm, has removed all these limits, making the design process faster and more versatile.



(a) Old magnet (b) New magnet

Figure 4.7: iNML magnet pins

4.1.3 Limits

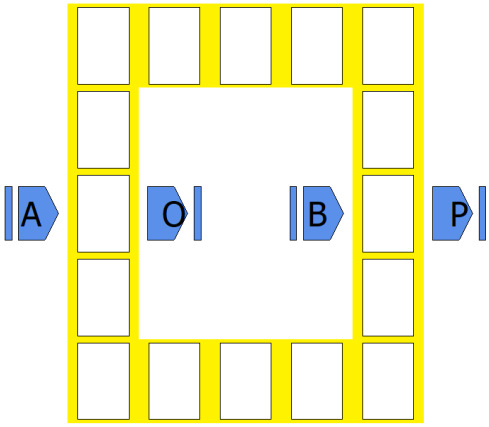


Figure 4.8: Symmetrical circuit that should not be solved

The reliance of the new algorithm on finding either an unambiguous element or a clock zone difference to determine the signal direction, may result in problems when having multiple ambiguous elements connected to each other.
The test performed have shown that the algorithm is able to connect properly a multitude of different layouts, in fact every working circuit provided was correctly solved. In addition to working circuits, layouts with errors where tested, and the algorithm could detect errors such as magnets with an even number of inputs or disconnected elements, as well as mismatches on pin connections. A kind of errors that the algorithm is not able to detect, can be found in symmetrical circuits such as the one shown in figure 4.8. In the layout shown, there are 2 ambiguous elements, which are the magnets directly to the left of the outputs. As they are both found to have 1 input, 1 output and 2 connections with an undetermined direction, one of them gets arbitrarily chosen as a majority voter, resulting in the other being a 3-way coupler, even though the layout itself shows no difference between the two magnets. The magnet chosen as a majority voter is not predictable, as it depends on the order the pins are stored in the lists MagCAD uses. The declaration of a magnet as a majority voter is caused by the way the algorithm have to guess the function when ambiguous elements are connected to other ambiguous elements, and due to the fact that unambiguous elements are always linked before ambiguous elements, once one

of the two ambiguous element gets declared a majority voter, all its supposed inputs get connected, resulting in the symmetry of the layout not being noticed and the circuit being solved.

4.1.4 Performance analysis

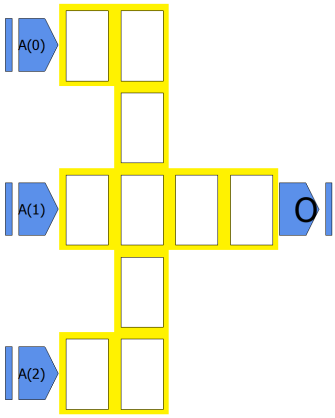
The algorithm makes extensive use of iteration, so it's important to know how its performance scales with the number of items in the layout.
The algorithm is composed of a main loop that continues until everything is connected or an error is detected. Inside this loop reside two more loops which iterate the lists of ambiguous and unambiguous pins.
The first loop of the algorithm is the iteration of the unambiguous pin list. As the index reached is saved between the iterations of the main loop, it's immediately evident that the number of iterations scales linearly with the number of elements of the list, which is directly proportional to the number of items of the layout, because every pin is processed at most once, and every item has a fixed number of pins. This means that the complexity of this loop is $O(n)$.
The second loop is the iteration of the ambiguous pin list. The iteration of this loop stops as soon as the function of an ambiguous element is determined, and always starts from the beginning of the list, in order to detect changes caused by the new connections made in the meantime. This means that in the worst case, where every ambiguous pin is in the list at once and the list has to be completely iterated every time to solve a pin, the number of iterations has a quadratic growth, giving the loop a complexity of $O(n^2)$.
On the other hand, the second loop goes over pins that can be solved immediately only once, so, in the best case, the complexity of the part of the algorithm tasked with the resolution of the ambiguity can be as low as $O(n)$.
As a consequence, the overall complexity ranges from $O(n)$ to $O(n^2)$ depending on the amount of ambiguous elements connected to each other inside the same clock zone. This is usually not a concern for circuits that respect the physical limitations for the number of magnets that can be cascaded inside a clock zone, but in general, to achieve the best performance, it's advised to make use of hierarchical structures

to reduce the number of items present in the layout.

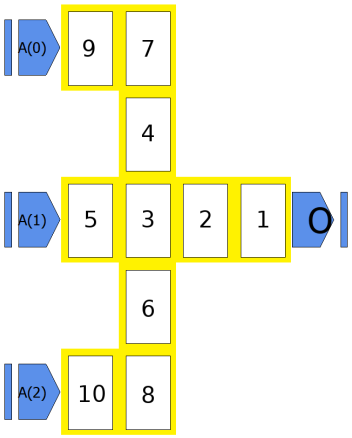
4.1.5 Examples

To better explain how the algorithm works some examples are provided. Magnets are given a number in order to be better addressed in the text, arrows are placed whenever two elements are linked and question marks are placed when the direction of the signal is been investigated.

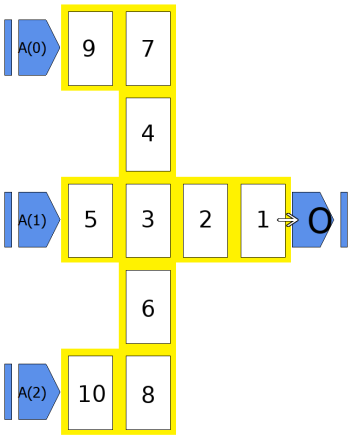
Example 1: Majority voter



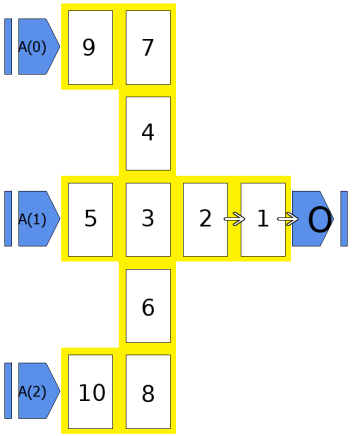
The first example is a majority voter, which will allow us to see how the algorithm treats ambiguous elements.



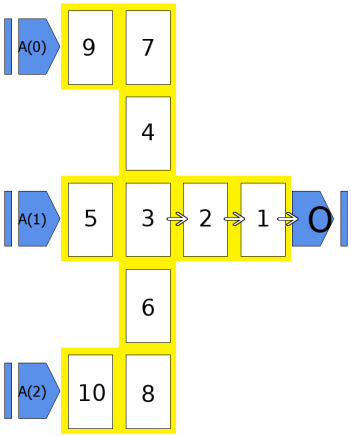
1) To start, the unambiguous pins are added to the unambiguous pin list. For this circuit, the only pin added is the input pin of the pin O.



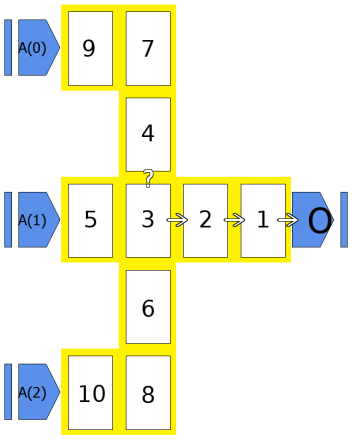
2) The unambiguous pin list is iterated and the first element, the input of the pin O, is linked with the magnet 1. The magnet 1 has only one other neighbor, so it's added to the unambiguous pin list.



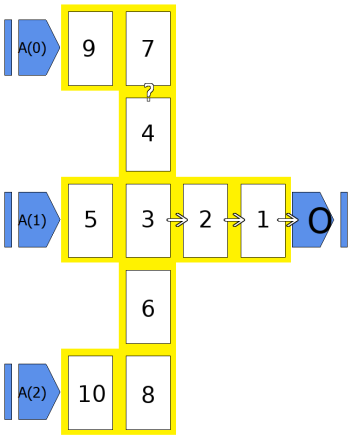
3) Magnet 1 is linked with magnet 2 and magnet 2 is added to the unambiguous pin list.



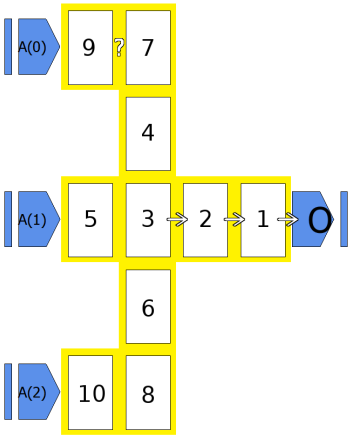
4) Magnet 2 is linked with magnet 3. Magnet 3 has more than one neighbor it's not connected to, so it's added to the ambiguous pin list. This is the end of the unambiguous pin list. As the ambiguous pin list is not empty, it will be iterated next.



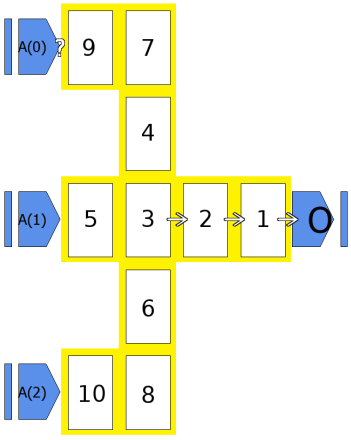
5) The first element is magnet 3. For each of its possible connections, the direction has to be determined. We start with magnet 4. Magnet 4 is in the same clock zone as magnet 3, so it's not possible to determine the direction of the signal at this step, but since it's a magnet used as a wire, the function moves along the wire and analyzes the pair magnet 7/magnet 4.



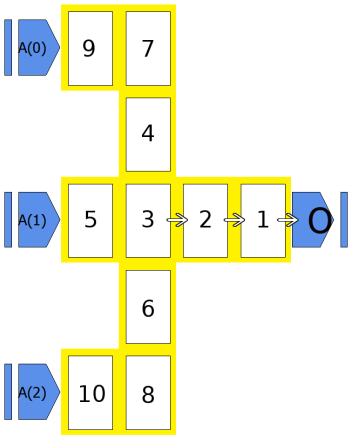
6) Magnet 7 and magnet 4 are also in the same clock zone, but like before it's possible to move along the wire and check the magnet 9/magnet 7



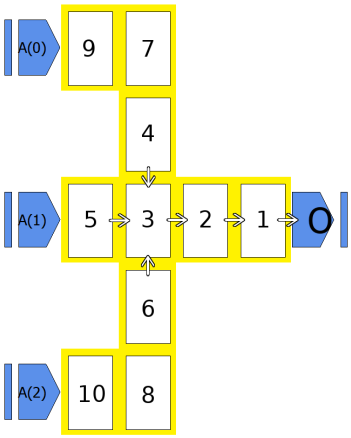
7) Magnet 9 and magnet 7 are again in the same clock zone, so the algorithm moves to magnet 9/pin A(0)



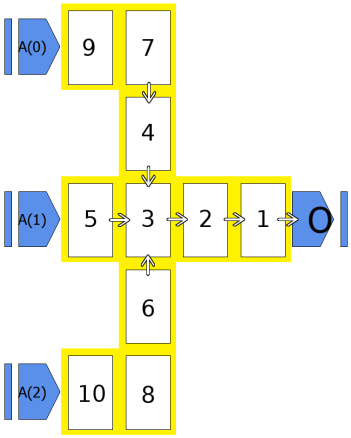
8) Since pin A(0) has a predetermined direction, we know that the signal goes from A(0) to magnet 9. This direction is the same for all the other pairs in the wire, meaning that magnet 4 is an input of magnet 3.



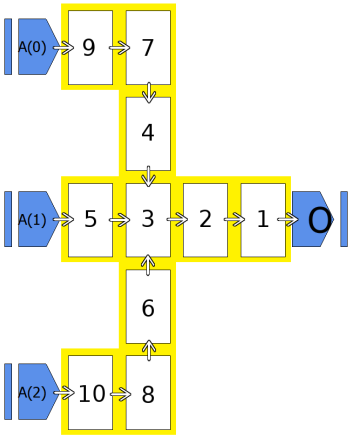
9) The same steps are executed to determine the direction of the connections 3-5 and 3-6, which are both inputs. This concludes the analysis of magnet 3. We have 1 already connected output and 3 inputs, so magnet 3 is a majority voter.



10) Magnet 3 is linked with magnets 4, 5 and 6, and they added to the unambiguous pin list. After that magnet 3 is completely connected and it's removed from the ambiguous pin list.

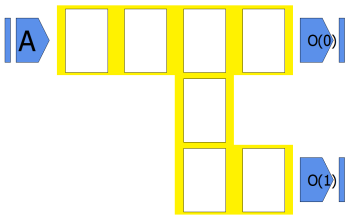


11) Since new pins were added to the unambiguous pin list, we go back to that and pick up where we left off, starting with linking magnet 4 to magnets 7 and adding magnet 7 to the unambiguous pin list.

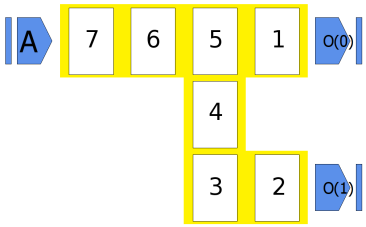


12) In the same way, the remaining elements are linked. After magnet 10 is linked with pin A(2) we reach again the end of the unambiguous pin list, but since the ambiguous pin list is empty, the algorithm has finished and everything is connected.

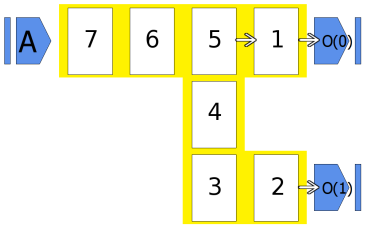
Example 2: Coupler



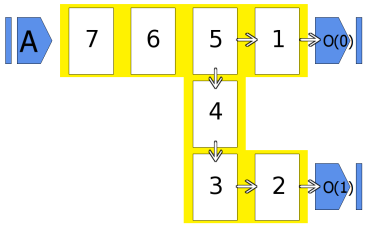
The second example is a coupler, which will be fully connected without being considered ambiguous.



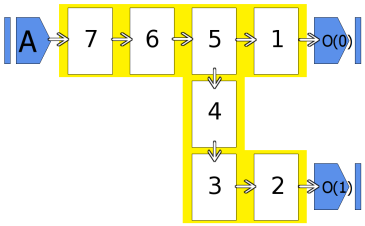
1) The first step is populating the unambiguous pin list. In this case we have the input pins of the pins O(0) and O(1).



2) As in the previous example, the list is iterated and O(0) is linked to magnet 1, O(1) to magnet 2 and magnet 1 to magnet 5. Magnet 5 can be connected to 2 magnets, so it's added to the ambiguous pin list.

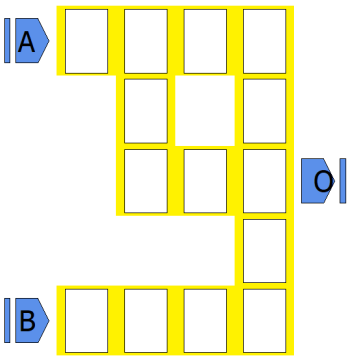


3) The iteration of the unambiguous list continues and 2 is linked with 3, 3 with 4 and 4 with 5. Now magnet 5 can only be connected to one other magnet, so it's not ambiguous anymore. It is thus removed from the ambiguous list and added to the unambiguous list.

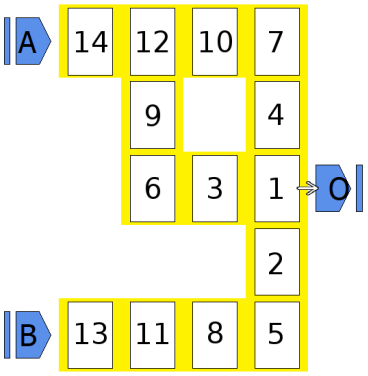


4) The algorithm proceeds linking 5 with 6, 6 with 7 and 7 with pin A, reaching the end of the unambiguous list. Since the ambiguous list is empty, the algorithm has finished.

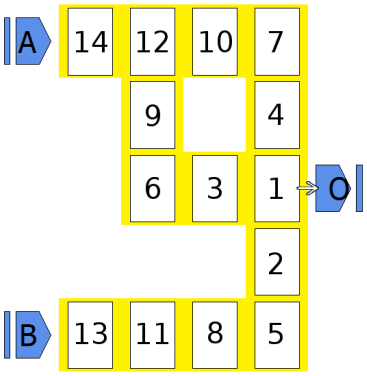
Example 3: More complex circuit



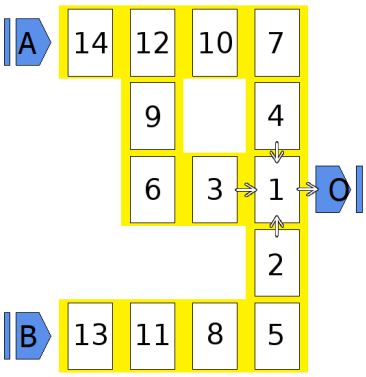
This example is the combination of the previous two and shows a case when there are connections whose direction can't be determined by the function.



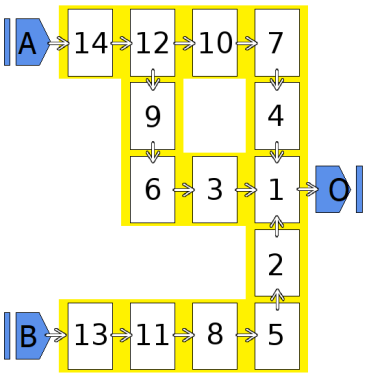
1) As in the previous examples, the unambiguous pin list is initialized with the input pin of the pin O and then pin O is linked to magnet 1, which is added to the ambiguous input list.



2) Now the algorithm tries to determine the direction of the connections between magnet 1 and its neighbors 2, 3 and 4. Magnet 2 is easily found as an input to magnet 1. Magnet 3 and 4, though, lead to magnet 12, which is ambiguous, leading to both connections labeled as undetermined.



3) Since a magnet with 1 output, 1 input and 2 undetermined connections can't be solved without a doubt, the algorithm repeats its analysis, but with an additional condition: that anything with 1 output and 3 possible inputs is a majority voter. This is exactly the situation magnet 1 is in, so magnet 2, 3 and 4 are considered inputs and linked to magnet 1.



4) After magnet 1 is solved the algorithm solves the the bottom part of the circuit immediately because it's a simple wire and upper part is solved as in example 2. After everything is connected, the algorithm has finished.

4.1.6 Pseudocode

This is the pseudocode of the main part of the algorithm. It takes the list of all the input and output pins of the layout and continues until it either finishes or it encounters an error (not shown in the pseudocode).

```
1  inputPins // List of input pins - given as input to the
       algorithm
2  outputPins  // List of output pins - given as input to the
       algorithm
3
4  unambiguousInputPins
5  ambiguousInputPins
6
7  for each pin P in inputPins do
8    if P is not a pin of a Magnet then
9      append P to unambiguousInputPins
10
11 pinIndex = 0  // Index of the next pin in unambiguousInputPins
       that will be linked

12 repeat
13   while pinIndex < size of unambiguousInputPins
14     currentInput = unambiguousInputPins[pinIndex]
15     for each pin otherPin in outputPins that can be connected
           to currentInput
16       link currentInput to otherPin
17       if otherPin is a pin of a Magnet then
18         if otherPin can be connected to only one cell
19           append otherPin to unambiguousInputPins
20         else if otherPin can be connected to more than one cell
21           append otherPin to ambiguousInputPins
22     increment pinIndex

23   for each currentInput in ambiguousInputPins
```

```
24    currentOutput = output pin of the cell parent of
          currentInput
25
26    nInputs = number of inputs already connected to currentInput
27    nOutputs = number of outputs already connected to
          currentOutput
28    nUndetermined = 0
29    for each pin P in outputPins that can be connected to
          currentInput
30      direction = direction of the signal between currentInput
            and P
31      if direction = output then
32        increment nOutputs
33      else if direction = input then
34        increment nInputs
35        lastOutputFound = P
36      else  // direction could not be determined
37        increment nUndetermined
38        lastUndeterminedFound = P
39    if (nInputs = 3 and nUndetermined = 0) or (nInputs = 2 and
          nUndetermined = 1) then

40      append currentInput to unambiguousInputPins
41      remove currentInput from ambiguousInputPins
42      exit loop

43    else if nInputs = 1 and nUndetermined <= 1 then

44      link currentInput to lastOutputFound
45      lastInputFound = input pin of the cell parent of
            lastOutputFound
46      if lastInputFound is a pin of a Magnet then
47        if lastInputFound can be connected to only one cell then
48          append lastInputFound to unambiguousInputPins
49        else if lastInputFound can be connected to more than
              one cell
```

```
50          append lastInputFound to ambiguousInputPins
51        remove currentInput from ambiguousInputPins
52        exit loop

53      else if nOutputs > 1 and nUndetermined = 1 then

54        link currentInput to lastUndeterminedFound
55        lastInputFound = input pin of the cell parent of
              lastUndeterminedFound
56        if lastInputFound is a pin of a Magnet then
57          if lastInputFound can be connected to only one cell then
58            append lastInputFound to unambiguousInputPins
59          else if lastInputFound can be connected to more than
                one cell
60            append lastInputFound to ambiguousInputPins
61        remove currentInput from ambiguousInputPins
62        exit loop

63      else if (nOutputs = 1 and nUndetermined + nInputs = 3) and
            the loop ended inconclusively the first time then

64        append currentInput to unambiguousInputPins
65        remove currentInput from ambiguousInputPins
66        exit loop

67  until all unambiguous input pins are connected and no ambiguous
      input pins are left
```

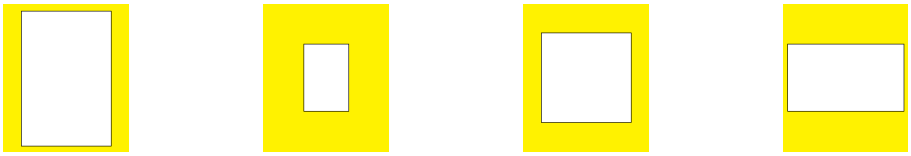
This is the pseudocode of the function that is used to determine the direction of the connections between two elements.

```
function checkSignalDirection(inputPin,outputPin)
// Function inputs
inputPin  // Input pin of the reference cell
outputPin // Output pin of a cell connectible to the reference
     cell

if inputPin is a pin of a Magnet then
  if clock phase of outputPin = (clock phase of inputPin - 1)
       mod number of clock phases then
    return output
  else if  clock phase of outputPin = (clock phase of inputPin
       + 1) mod number of clock phases then
    return input
  else if clock phase of outputPin = clock phase of inputPin
       then
    newInputPin = input pin of the cell parent of outputPin
    Count = count of the cells connectible to newInputPin
    if Count = 1 and number of cells already connected to
         newInputPin = 0 then
      newOutputPin = cell connectible to newInputPin
      return checkSignalDirection(newInputPin, newOutputPin)
    else if Count = 0 and number of cells already connected to
         newInputPin = 1 then
      return input
    else
      return undetermined
  else
    return undetermined
else
  return input
```

4.2 Magnet resizing

Another functionality that has been added to the iNML plugin is the ability to resize magnets. This is done by using the interface shown in figure 4.10, which is positioned in the bottom left corner of MagCAD window, and it works by adding to the selected item optional properties that override the default ones. When an item is selected its current dimensions are displayed in the height and width fields and then new ones can be entered and set by pressing "Apply changes". If the entered values are equal to the default ones or if the button "Restore default" is pressed, the optional properties are reset. The new size can't exceed the size of the grid, and a check is performed to prevent the copy from one drawing to another of items whose dimensions is incompatible with the drawing settings. The properties are saved in the .qll and .qcc files, but they don't affect MagCAD VHDL export, as the VHDL description is used for the behavioral testing of the circuit and not for physical simulations of the magnets.



(a) 60x90 nm (b) 30x45 nm (c) 60x60 nm (d) 80x45 nm

Figure 4.9: Examples of magnets resized

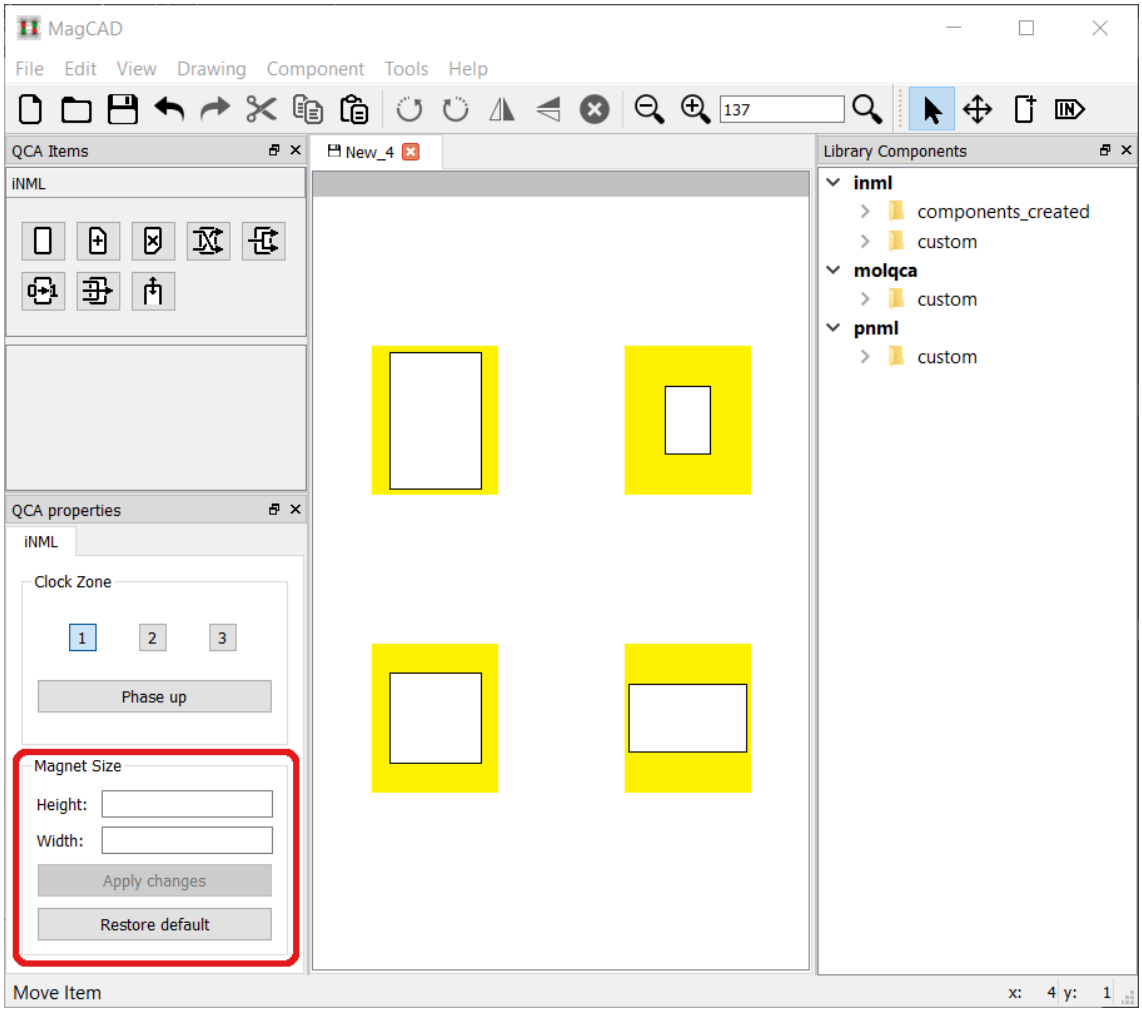


Figure 4.10: MagCAD magnet size interface

Chapter 5

pNML plugin

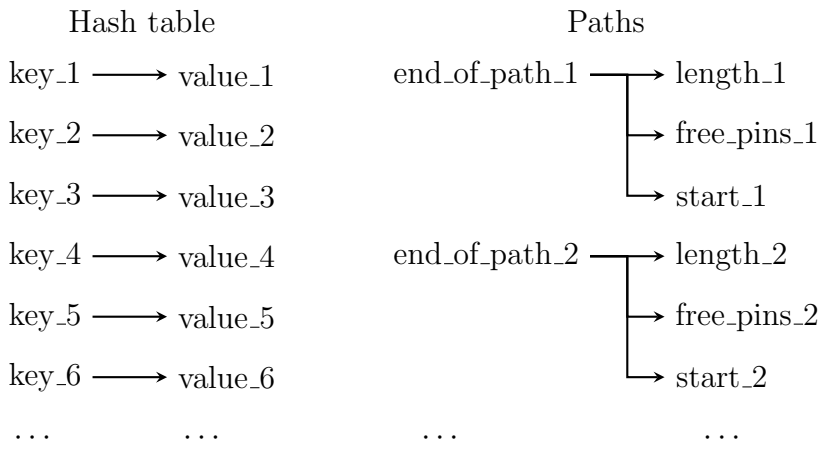
5.1 Critical path estimation for hierarchical circuits

The clock signal in pNML circuits works differently from that of other QCA implementations, and to guarantee the correct functioning of the circuit, its frequency must allow the full propagation of the magnetic domain across all domain walls before triggering a switch.
This means that the maximum frequency of operation depends on the length of the longest path between elements that interrupt the propagation (nucleation centers, including inverters, and notches). For this reason, when the elements are linked during the VHDL generation, it's useful to keep track of the various paths of the signal to find the longest one. As the clock period constraint depends on the sum of the nucleation time and the propagation time, two longest paths are considered, the longest path that starts with a nucleation center and the longest that does not. When using hierarchical layouts, the longest paths of the components has to be considered as well, as they may be the longest paths of the entire design. MagCAD already did that, but under the assumption that all the connections of the component with the outside pass through a nucleation center or a notch, meaning that the internal and the external paths are separated. This is not necessarily the case, a path can start or end inside a component and it's even possible for a signal to pass without interruptions through a component, creating a long domain wall that both starts and ends outside of it.
If the paths are not separated, they have to be summed together to discover the real longest path. To do so, when exporting a component, it's necessary to save not only the longest paths, but also any other path that can be continued outside

the component. More precisely, for every input pin, only the longest path that ends inside the circuit is saved, and only if it is longer than any path that start from that pin and exits the circuit, because this are the only paths that have the possibility to be the longest.

5.1.1 Computation of the paths in non-hierarchical layouts

When the netlist for the VHDL generation is created, connections between items are formed by linking together their pins. During this step the length of the paths are calculated.
Every path is saved in a data structure containing the length, the number of free pins and the start of the path. The free pins value is the number of connections yet to be used of the end point of path, and it's used to take into account branches. The list of paths is then saved into an hash table with the coordinates (in the format x.y.layer) of the end of the path as the key, as shown in figure 5.1.



(a) Generic hash table (b) Paths hash table

Figure 5.1: How paths are stored inside MagCAD

When two pins are linked, the input pin is analyzed. If the input pin is a pin of the circuit, nothing is done, as it is not considered a physical element of the circuit. If the input pin belongs to a nucleation center, inverter or notch item, then a new

path with length 0 and the item whose input pin is currently being considered as the starting point is created and added to the table under the coordinates of the input pin. In every other case, the output pin is considered: if the path table doesn't contain a path with the coordinates of the output pin, a new one with 0 length and the output pin as the starting point is created. Otherwise, the existing path is retrieved, its length is incremented by one and it is added to the table with the coordinates of the output pin. Then the free pins of the older path are decreased by one and if they become 0 the path is removed from the table, otherwise it is updated in the table, because it means that there is a continuation of this path not yet connected, which will need this path in the future.

5.1.2 Saving of the paths

Once every element is linked and all the paths are accounted for, it's time to find the longest ones for the clock frequency calculation, and the ones that need to be saved for usage as components.
The search for the longest paths is done in the obvious way: the path table is iterated and every time a path is found that is longer than the previously found longest path, the path is replaced. The resulting two paths are saved with the name "longest_path_nuc" and "longest_path_noNuc".
The search for the paths that continue outside the component is done in two steps. First of all, every path that ends with an element connected to a pin of the circuit has to be added unless that element is a notch or a pad. This is done by taking the list of all output pins of the circuit, and for each of them getting the coordinates of the element they are connected to and retrieving from the path table the one with the end point corresponding to that coordinates. This path is then added to the table of paths to save. The second step requires an iteration of the path table to search every path that has a pin indicated as the start element. Then, unless a path with the same starting point exist and is longer, or the current path has 0 length, the path is added to the table of paths to save with the starting point as the key. This difference in the usual convention of using the end point as the key is motivated by two considerations: 1) when saving the path it allows the program to quickly see

if a path with the same starting point already exists, instead of having to iterate the entire list to check the start point; 2) comparing the coordinates stored in the start field and the coordinates that make up the key is a simple way to see whether a path passes through the component or not: if they are the same, we can conclude that the path ends inside the component.
The paths are saved in a tab-separated values format in a hidden .csv file, stored in the same folder as the VHDL outputs. Only the key, the length and the start information is saved, as the number of free pins is only useful during the linking phase.

5.1.3 Computation of the paths in hierarchical layouts

The calculation of the path length for hierarchical layout is the same as those of non-hierarchical ones, with the added step that when the input or output pin that is being considered belongs to a component, the paths of the component have to be considered. More precisely, the file containing the paths of the component is opened and read, the coordinates of the pin are translated to the relative position inside the component, because otherwise they wouldn't match with the ones saved in the file, and then the type of the pin is considered.
If the pin was an input, all the paths that have that pin as the starting point have their start point compared with the key saved in the file. If they are equal it means that they end inside the component, so their length is added to the length of the path until this point and it is added to the path table with a key corresponding to the coordinates of the pin. If, on the other hand, their key list a different set of coordinates, it means that the end is another pin of the component, so the old path is kept as it is in order to avoid summing the length twice when it is connected with the other end later.
If the pin was an output, the path is retrieved by using the coordinate of the pin as the key and the length of the path internal to the component is always added to the new path. Then, if the starting point of the path is an item, it is set as the starting point of the new path, so that it is still possible to determine whether it was from a nucleation center or not. If instead it lists a pin, the start point of the new path is called "ConnectTo x.y.layer".

Finally, the path labeled "longest_path_nuc" is added to the table, and the path labeled "longest_path_noNuc" is added as well, if it doesn't have a pin as start.
After all the items are connected, the path table is iterated and all the paths which have the "ConnectTo" start sum their length to the path indicated by the coordinates, as to make a single long path with the correct start point.

5.1.4 Performance analysis

The algorithm used for the calculation of the critical path does not include any nested loop, and the number of operations required for its execution scales linearly with the number of elements.
According to the documentation of the Qt library, the container class "QHash", which is used for the storing of the paths, has worst case lookup and insertion complexity of $O(n)$, with an amortized behavior of $O(1)$ for repeated calls. This means that the average time required for each data insertion and retrieval can be approximated as constant, resulting in an overall performance of the entire algorithm that depends on the complexity of the main loop, so $O(n)$, making the algorithm well suited for working with layouts containing a large number of elements.

5.1.5 Examples

Two examples will be presented to provide a more practical explanation of how the paths are saved and used. For the sake of simplicity, no multilayer layouts will be used (so the layer coordinate will always be 0) and MagCAD grid will be shown to help with the reading of the lengths and of the coordinates.

Example of non-hierarchical layout

The first example is a non-hierarchical layout with 3 paths, which in the figure are named a, b and c:

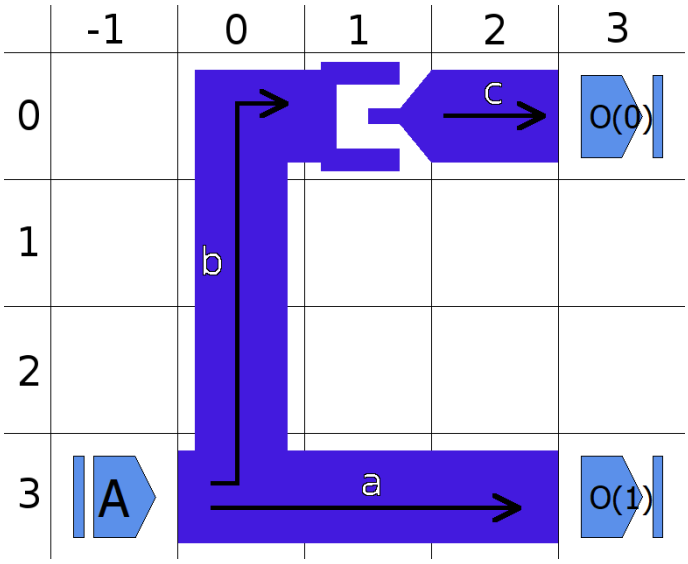


Figure 5.2: Example 1 layout

- a is a path that starts from pin A, which has coordinates (-1, 3), and ends with the magnet at (2, 3), which is connected to the pin O(1), meaning that this is a direct path that crosses the entire layout. This path is saved in the paths hash table with the key "2.3.0", which is its end point, and with length = 3 and start = "Pin -1.3.0".

- b also starts from pin A, but it ends with the magnet at (0, 0), which is connected to an inverter. This path is saved with key "0.0.0", length = 4 and start = "Pin -1.3.0".

- c starts with the inverter at position (1, 0) and ends with the magnet at (2, 0), which is connected to the pin O(0). It is saved with key "2.0.0", length = 1 and start = "Inverter". The coordinates of the inverter don't matter, because it's an internal item of the component.

Now the paths are saved to the .csv file.
The longest path starting from a nucleation center is path c, which will be written as

```
longest_path_nuc     1    Inverter
```

The longest path not starting from a nucleation center is path b, which will be written as

```
longest_path_noNuc   4    Pin -1.3.0
```

Then all the output pins are iterated: the first output pin is O(0), which is connected to the magnet at (2, 0), so the path with key "2.0.0", which is path c, is written to the file as

```
2.0.0    1    Inverter
```

The second output is O(1), which is connected to the magnet at (2, 3), so the path with key "2.3.0", which is path a, is written to the file as

```
2.3.0    3    Pin -1.3.0
```

Finally, all the path whose start contains "Pin" are considered as well: these paths are a and b, of those only b is written because a is shorter. Since path a ends inside the layout, it is written with the coordinates of the start as the key:

```
-1.3.0   4    Pin -1.3.0
```

The final file will then look like this:

Listing 5.1: Generated .csv file

```
-1.3.0   4    Pin -1.3.0
2.0.0    1    Inverter
2.3.0    3    Pin -1.3.0
longest_path_nuc     1    Inverter
longest_path_noNuc   4    Pin -1.3.0
```

Path b and c are written twice in the file, one time because they are path that start/end with a pin and the other time because they are the longest paths of the entire layout. It's possible for a path to be even written three times, if it starts and ends with a pin and is one of the longest ones of the layout: one time with the start as the key, one time with the end as the key and the last time with "longest_path_nucnoNuc" as the key. Removing them has been deemed unnecessary

at this point, as they do not affect the capability of the algorithm to correctly calculate the lengths of the paths.

Example of hierarchical layout

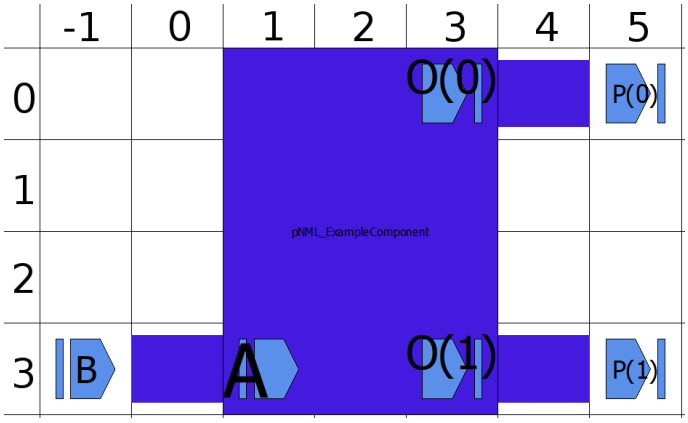


Figure 5.3: Example 2 layout

For the second example, we use the layout used in the first example as a component and see how MagCAD calculates the paths.

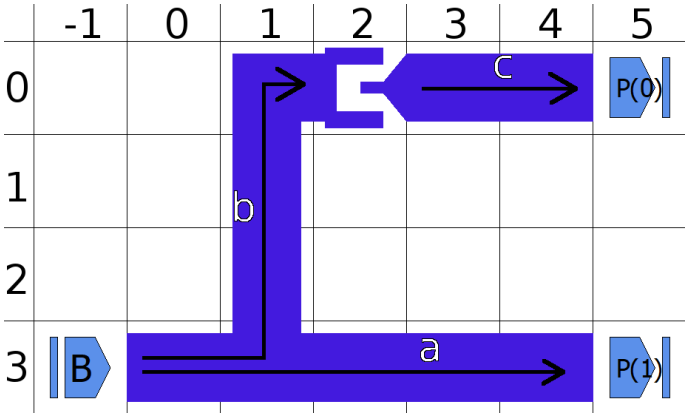


Figure 5.4: Example 2 as a flat layout

In figure 5.4, we see how the layout would look if it was flat instead of hierarchical. We expect to have the same paths of the first example, but longer.

Going back to the hierarchical layout, there are three paths present, one going from pin B to pin A of the component, one going from O(0) to P(0) and one going from O(1) to P(1). All three are of length 1, externally, but all three are extensions of the internal path of the component.
The linking process for the pNML technology works similarly to the iNML one, but instead of starting from the input pins of the items, it starts from the input pins of the circuit. In this case there is only pin B, so it starts from there.
Pin B is linked with the magnet in front of it and a path with length 1 is created, with the key "0.3.0", which corresponds to the coordinates of the magnet, then the magnet is linked to the pin A of the component. In doing so it detects that pin A belong to a component and loads the .csv file of said component, which is the one shown in the figure 5.1.5. Then it translates the coordinate of the magnet with the coordinates of the component: the magnet is at (0, 3) and the component is at (1, 0) (every item larger than one square is considered located in its top left square). This means that the pin we are looking for has coordinates (0, 3) - (1, 0) = (-1, 3). This is reformatted in "-1.3.0" (the layer undergoes the same treatment as the other coordinates, but since it's always 0 we are ignoring it). The paths of the component are searched for a path that contains "-1.3.0" in the start field and three are found. For each of them, the key is analyzed. The first one comes from the path

```
2.3.0    3    Pin  -1.3.0
```

This path has a key different than the coordinates in the start field, so it's not added to the path table. The same happens to the path

```
longest_path_noNuc   4    Pin  -1.3.0
```

because it also has a key different than the coordinates in the start field. Finally the path

```
-1.3.0   4    Pin  -1.3.0
```

is loaded. This path has a key "-1.3.0", which corresponds to the coordinates in the start field, meaning that it ends inside the component. So, a new path is created, its length is set as $1+4=5$ and it is inserted in the path table with the key "1.3.0", which is the position of the pin A in the current layout. Then the path labeled "longest_path_nuc" is also added to the table. The "longest_path_noNuc" is not

added because it has "Pin -1.3.0" as the start.
So currently the path table has 3 paths, "0.3.0" with length = 1, and "1.3.0" with length = 5 and "pNML_ExampleComponent_nuc" with length = 1.
The algorithm then proceeds to the pins P(0) and P(1). P(0) is linked with the magnet in front of it and the file is read again. As before, the coordinates are translated and result in (3, 0) - (1, 0) = (2, 0), which results in the string "2.0.0". In this case the string is directly used as the key to retrieve the path

```
2.0.0   1   Inverter
```

which is used as the base for the new path which is created. The length of the path is then incremented by 1 (making it 2), the free pins value is updated with the value of the magnet, which is 1, and the path is then added with the key of the coordinates of the magnet "4.0.0".
Finally, we have pin P(1), which is treated the same way as P(0). Its coordinates are translated to obtain (2, 3) which becomes the key "2.3.0" which is used to retrieve the following path.

```
2.3.0   3   Pin -1.3.0
```

This path is different, though, because it lists a pin as the starting point. So, in addition to the previous steps, the pin coordinates are summed with the component coordinates, resulting in (-1, 3) + (1, 0) = (0, 3), and so its start is changed to "ConnectTo 0.3.0".
The magnets are then connected to pins P(0) and P(1), ending the paths, and the linking process is finished.
At this point the path table is

```
4.3.0 4 ConnectTo 0.3.0
4.0.0 2 Inverter
0.3.0 1 Pin -1.3.0
1.3.0 5 Pin -1.3.0
pNML_ExampleComponent_nuc 1 Inverter
```

The final step is searching for all the paths with a "ConnectTo" and combining it with the proper path. In this case we have path "4.3.0" which has to be connected to path "0.3.0". This means that the length of the path "4.3.0" is summed to the

length of the path "0.3.0", resulting in a length of 5, and the start of the path "4.3.0" becomes the same as the start of the path "0.3.0", which is "Pin -1.3.0".
The final table is

```
4.3.0 5 Pin -1.3.0
4.0.0 2 Inverter
0.3.0 1 Pin -1.3.0
1.3.0 5 Pin -1.3.0
pNML_ExampleComponent_nuc 1 Inverter
```

The creation of the .csv file follows the same steps seen in the previous example and results in

Listing 5.2: Generated .csv file

```
4.3.0   5   Pin -1.3.0
4.0.0   2   Inverter
-1.3.0  5   Pin -1.3.0
longest_path_nuc    2   Inverter
longest_path_noNuc  5   Pin -1.3.0
```

This layout can then be used as component in another hierarchical layout, and so on.

Chapter 6

Molecular QCA

Originally MagCAD already had a plugin for the Molecular QCA (MolQCA) technology, but as it was very bare-bones and lacked many functionalities, one of the goal of this work was to flesh it out by adding everything required to create a complete layout.

6.1 Drawing settings

The choice of drawing settings to use is an important step that deeply affects the rest of the plugin, so its sections will be analyzed more in detail.

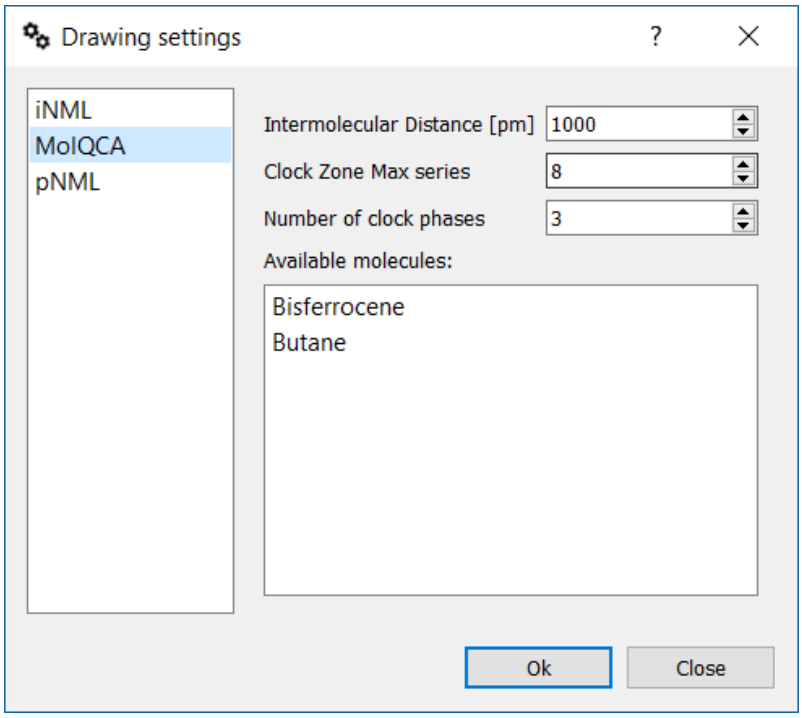


Figure 6.1: MolQCA drawing settings

6.1.1 Intermolecular Distance

The intermolecular distance setting affect the size of the cells that compose the grid of the drawing. The grid is square, so only one value of distance is required, as opposed to iNML, where there are different values for the height and the width, and the length of a side is twice that of the intermolecular distance, because as explained in the chapter about the technology, MolQCA cells are made by a pair of molecules. Initially a more physical approach was considered, where the grid would represent the crystal lattice of the substrate and the molecules would have different span in the drawing calculated from their height and width. In this case instead of the intermolecular distance, the distance between two binding point of the substrate would be provided, and the molecules span would be the result of the division between this distance and their dimensions. This resulted in rectangular elements, which could not be rotated freely.
As a result of these considerations, intermolecular distance was chosen as the parameter to use, and all the cells are 1x1 squares in the drawing.
The distances provided in the settings window go from 600 to 1500 pm, which are the values most commonly used, and the arrows increment or decrement the value by 100 pm at a time. If other values in the range are required, there is the possibility of entering them manually in the text box.

6.1.2 Clock Zone Max series

This is a setting present also in the iNML plugin, which refers to the maximum number of elements that can be placed in a single clock zone.
This setting only affect the clock zone that can be optionally drawn to aid with the definition of the phases of the items and does not affect the design. This value goes from 2 to 10.

6.1.3 Number of clock phases

This is a self-explanatory setting which serves the same purpose as the one found in the iNML plugin, defining the number of clock phases to be used in the drawing. The possible values are 3 and 4.

6.1.4 Available molecules

This is simply a list of the molecules available to use in the drawing.
Initially it was a proper setting that made possible to select individual molecules from the list to enable and disable them from use, but it was deemed unnecessary and now every molecule is always available.
Currently only 2 molecules are present: Bisferrocene and Butane.

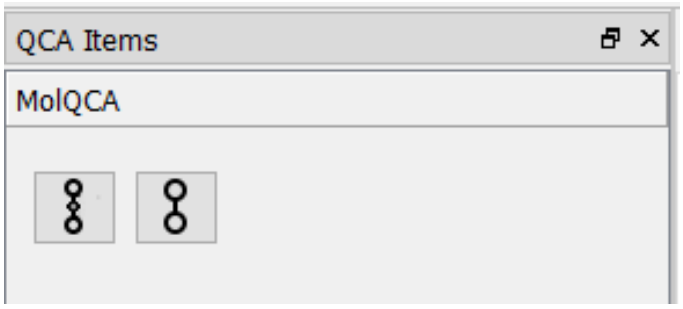
6.2 QCA Items

The only items available to use in the drawing are cells composed of pairs of molecules, with no pre-made logic structures present. In figure 6.2.a we can see the buttons used to create them, and in figure 6.2.b and 6.2.c are present the cells as they appear in the drawing itself.
While the span of the molecule items in the drawing is now fixed, the physical dimensions of the molecules are still used to scale the drawings: there is a file called "moleculedata.h", shown in the listing 6.1, which contains the heights of bisferrocene and butane, which are used to make the butane drawing smaller than the bisferrocene one. The data about the widths are still present, albeit commented, because they were used when the span of the items was variable.

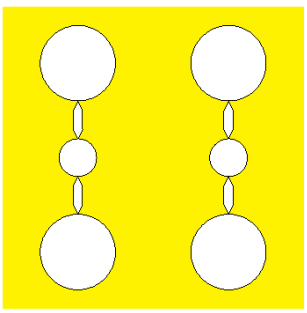
Listing 6.1: moleculedata.h

```
1 #ifndef MOLECULEDATA_H
2 #define MOLECULEDATA_H
3
```

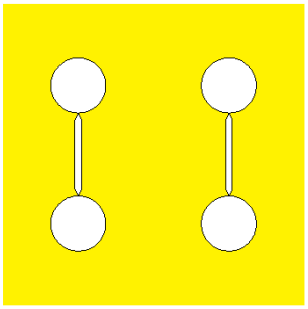
```
4 // Molecule size (unit is pm)
5 const int bisferroceneHeight = 1500;
6 //const int bisferroceneWidth = 800;
7 const int butaneHeight = 1100;
8 //const int butaneWidth = 500;
9
10 #endif // MOLECULEDATA_H
```



(a) MolQCA Items



(b) Bisferrocene



(c) Butane

Figure 6.2: Available items

6.3 Molecule manipulation

Despite the fact that molecules are usually used as a pair, it can still be required to manipulate them separately or even to have single molecules alone to create drivers. For this reason, it has been made possible to enable, rotate and shift the molecules individually. Figure 6.4 shows some of the possibilities. Additionally, it is still possible to rotate the entire cell by steps of 90°.

The ability to interact with individual molecules has been implemented by adding an extensive set of properties to every item, which can be modified using the interface shown in figure 6.3.
Molecule 1 and 2 refer to the molecules respectively on the left and the right when the cell is not rotated.
Movements of the molecule in the z axis are not visible due to the constraints of using a bidimensional drawing scene, but the ability to do so has been added so that it can be used by simulations that make use of the layout produced by MagCAD.

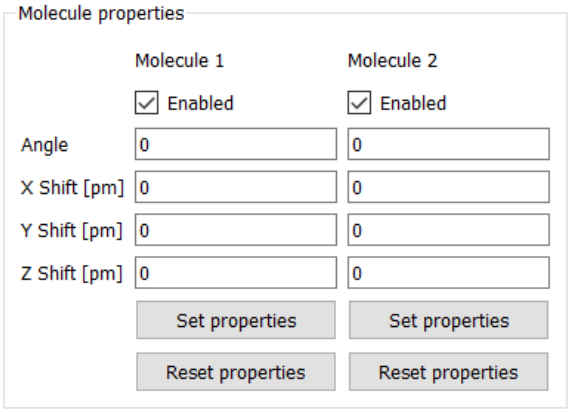


Figure 6.3: MolQCA properties interface

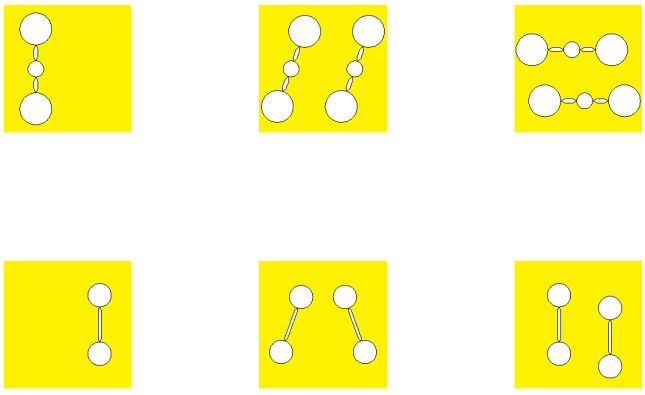


Figure 6.4: Example of single molecules in a cell and various rotations and shifts

6.4 Layout export

For layouts using the Molecular QCA technology, behavioral simulations based on the VHDL netlist, like those used to verify iNML and pNML circuits, would be of limited use. As a consequence, no VHDL library of the molecules has been developed, and as such the Molecular QCA plugin is not capable of exporting its layouts to VHDL.
Instead, the creation of the plugin was done with the goal of having the layout file saved by MagCAD directly simulated, exploiting the SCERPA simulator developed by the VLSI group. Since the simulator works with the physical properties of the molecules, it's possible to take full advantage of the abilities to rotate and shift molecules implemented, to create different geometries or to take into account process variations.

Chapter 7

Unit testing

Unit testing refers to tests performed on a specific part of a software to verify that its behavior is what's expected. To test the functionality of the algorithms developed for MagCAD, a project capable of automated unit tests has been developed. The tests are entirely functional, executing the desired function in a controlled environment and only checking the correctness of the results.

7.1 General

The test project is composed of a simple main function which calls the individual subtests that need to be run, as shown in the listing 7.2. The selection of tests to run is performed by editing an header file called "testtorun.h", shown in the listing 7.1, which through the use of "#define" directives set to 1 or 0 allows the switching on and off of individual tests. If all tests are to be run, there is the possibility to define "RUN_ALL" as 1, which overrides all the other defines.

Listing 7.1: testtorun.h

```
1 #ifndef TESTTORUN_H
2 #define TESTTORUN_H
3
4 #define RUN_ALL 0
5
6 #define TST_GENERATEVHDLTEST  0
7 #define TST_PNMLCRITICALPATH  1
8
```

```
#endif // TESTTORUN_H
```

Listing 7.2: test.cpp

```
int main(int argc, char *argv[])
{
    int dummyArgc = 0; // Used to launch "app"
    QApplication app(dummyArgc, nullptr);
    int status = 0;

#if RUN_ALL == 1 || TST_GENERATEVHDLTEST == 1
    // TEST generateVhdlTest //
    tst_generateVhdlTest generateVhdlTest;
    status |= QTest::qExec(&generateVhdlTest, dummyArgc,
        nullptr);
#endif

#if RUN_ALL == 1 || TST_PNMLCRITICALPATH == 1
    // TEST tst_pnmlCriticalPath //
    tst_pnmlCriticalPath pnmlCriticalPath;
    status |= QTest::qExec(&pnmlCriticalPath, dummyArgc,
        nullptr);
#endif

    return status;
}
```

Currently two tests are present, which check the VHDL generation algorithm of the iNML and pNML plugin. Each of these tests has a folder where the files used for testing are kept, which are called respectively "INML_Test" and "PNML_Test". Inside these folders they have the .qcc and .qll files they use as a input, a "QCC_List.csv" file which list the files that have to be loaded and the expected results, and the VHDL folder where the outputs of the generation are put. There is a way to comment lines in the .csv file: when reading it, both tests ignore any row that begins with the

character "#", resulting in an easy way to exclude files without deleting the entire row. The first line in the file is commented in this way and is used as a title, to label the data in each column.
As both test deal with the proper connection of the layout items, the unit whose behavior is tested is the function "generateVhdl" of the class "VhdlController", which has the task to generate the VHDL description of a component, described inside MagCAD using an object "QcaComponent". To obtain the required object, a class called "qcaFileReader" is used. This class mirrors the "DrawingReader" class used inside MagCAD to read the files .qll and .qcc, but differs from the original in the way it handles hierarchical layouts: instead of using MagCAD "ComponentsWidget", the element on the right side of MagCAD's main window which is also used to insert the components in the drawing, to retrieve the component data, "qcaFileReader" uses the component library and name to find the component's .qcc file inside the test folder and then calls itself to load it.
Both the .qcc and .qll files are used during the tests because while the .qcc file contains all the required informations about the items and the pins, it lacks any data about the settings, resulting in errors when those differ from their default value. As the .qcc and .qll files only differ for the extension, in the .csv file only the .qcc is saved, and the other is obtained by a simple replacement.

7.2 iNML

The iNML test was designed to verify that the algorithm to automatically detect the functionality of magnets worked as expected. To do so, the test uses the element count generated during the VHDL generation. The element count has an entry for every item of the layout, and generates a different entry for every function that the magnet can have:

- "magnet" is used when the magnet acts as a wire

- "magnet_coupler2" and "magnet_coupler3" are used for couplers with respectively 2 and 3 outputs

- "magnet_mv" is used for majority voters

The counts for the other elements are ignored, as they are not subject to ambiguities. During the VHDL generation, the element count is printed in the "definition_inml.vhd" file, but the debug build, which is the one used for testing, also outputs it to a .csv file in the same folder, which is easier to read.
If the VHDL generation fails, the test is also capable of reading the log of the generation to determine the kind of error that occurred, and some tests that are expected to fail are included in the .csv file, with the expected error string for comparison.
So, the content of the "QCC_List.csv" file is as follows: name of the .qcc file, expected number of "magnet", expected number of "magnet_coupler2", expected number of "magnet_coupler3", expected number of "magnet_mv", expected error.
Some example lines from the actual file, including the title, are shown below:

```
#FileName,magnet,magnet_coupler2,magnet_coupler3,magnet_mv,LogError
custom/BasicLoop1.qcc,33,1,0,1,
custom/InputDisconnectedAnd1.qcc,0,0,0,0,Input
    disconnected
components_created/rca4bit_hier.qcc,5660,0,0,0,
#custom/Test_1.qcc,55,1,0,2,
```

It's important to notice that the file name of the component has to include its library, because the reader assumes the same folder structure of MagCAD, which is .qll file in "drawings" and .qcc in "lib*library_name*" and can't find the components of hierarchical layouts if the folder structure differs. The libraries shown in the list are "custom", which is the default one for components created by MagCAD, and "components_created", which is the one used by components generated by ToPoliNano.
To sum up, the test operates as follows:

1. Read the "QCC_List.csv" file to retrieve the list of files to process and the expected results
   For each file:

2. Use "qcaFileReader" to read the .qcc file to generate the component and the .qll file to obtain the drawing settings

3. Use "generateVhdl" to produce the VHDL for the component

4. Read the element count or the error in the log

5. Compare the read results with the expected ones

7.3 pNML

The work on the pNML algorithm concerned the calculation of the longest paths of the circuit, so these are the results that are tested. The contents of the "QCC_List.csv" file for this test are: name of the .qcc file, expected length of longest path not starting from a nucleation center, expected length of the longest path starting from a nucleation center.
A few lines of example of the file are shown below:

```
#FileName,longest_path,longest_path_noNuc
custom/test_multilayer1.qcc,2,6
custom/rca32_3d.qcc,16,0
```

The length of the longest paths is saved in the hidden file used by MagCAD for the hierarchical layouts, so in this case there are no differences between the debug and the release builds, as far as the test is concerned. Since no work was done on the detection of errors during the generation of VHDL, this test does not read the log in case of error and simply fails the test if the VHDL generation doesn't produce the file with the paths length.
The steps executed are then the same of the iNML case:

1. Read the "QCC_List.csv" file to retrieve the list of files to process and the expected results
   For each file:
2. Use "qcaFileReader" to read the .qcc file to generate the component and the .qll file to obtain the drawing settings
3. Use "generateVhdl" to produce the VHDL for the component
4. Read the length of the longest paths
5. Compare the read results with the expected ones

Chapter 8

Conclusions

As the research in the field of emerging technologies continues, the importance of a flexible design tool like MagCAD should not be underestimated, and the work that has been done has successfully improved and expanded its functionalities.
The newly designed algorithm for the iNML technology has removed several limitations imposed by the previous way of performing the VHDL generation, and while its worst case performance of $O(n^2)$ is not optimal, it is capable of performing well even when working with layouts containing a large number of elements, as long as the physical limits of magnets are taken into account in the design.
The improvement of the pNML critical path detection algorithm is instead focused on increasing the versatility of custom components, allowing the use of hierarchical structures that previously would have resulted in a mistaken clock frequency calculation. The VHDL generation algorithm complexity has not been degraded by this addition, resulting in an adaptable system able of dealing effectively with any kind of circuit.
Finally, the development of a plugin for the Molecular QCA technology has been a valuable addition to the set of supported technologies, and its inability to generate a VHDL netlist is overcome by the SCERPA simulator developed by the VLSI group.
A possible direction for the further development of MagCAD could then be the expansion of the Molecular QCA plugin with the addition of new molecules or even with the ability to use user defined molecules, exploiting the generality with which the current items have been implemented.

Bibliography

[1] A. Di Mauro, M. Geuna, F. Viggiano, and E. Plozner. Vhdl model for domain magnet logic devices, design and simulation of a generic n-bit mac. *Low Power Electronic Systems course, Prof. M.Zamboni*, 2014-2015.

[2] F. Cairo, G. Turvani, F. Riente, M. Vacca, S. Breitkreutz v. Gamm, M. Becherer, M. Graziano, and M. Zamboni. Out-of-plane nml modeling and architectural exploration. *IEEE 15th International Conference on Nanotechnology (IEEE-NANO)*, 2015.

[3] G. Turvani, F. Riente, E. Plozner, M. Vacca, M. Graziano, and S. Breitkreutz v. Gamm. A pnml compact model enabling the exploration of 3d architectures. *IEEE Transactions on Nanotechnology*, 2017.

[4] E. Plozner. P-nml architecture: modelling and analysis. Master's thesis, Politecnico di Torino, 2016.

[5] Y. Ardesi, A. Pulimeno, M. Graziano, F. Riente, and G. Piccinini. Effectiveness of molecules for quantum cellular automata as computing devices. *Journal of Low Power Electronics and Applications*, 2018.

[6] F. Riente, U. Garlando, G. Turvani, M. Vacca, M. R. Roch, and M. Graziano. Magcad: A tool for the design of 3d magnetic circuits. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 3:65–73, 2017.

[7] S. Pennavaria. Design and implementation of a graphic editor for nanotechnologies. Master's thesis, Politecnico di Torino, 2015.