INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Computação Quântica: arquitecturas e simulação de operação de dispositivos

Geração automática de *Layout* QCA para circuitos combinatórios

Tiago Teresa Teodósio

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Setembro de 2007

Acknowledgments

Thanks to Carla Duarte for her support. Thanks to Prof. Paulo Flores for some discussions about data structures and logic synthesis algorithms. Thanks to Prof. Susana Freitas and Prof. Paulo Freitas for some enlightenments about the feasibility and fabrication of nano-magnets, towards Magnetic QCA. Thanks to Pedro Tomás for the thesis template. Thanks to Ricardo Chaves for english corrections. Thanks to Prof. Leonel Sousa for his guidance and help.

Abstract

The presented thesis is about a new technology, QCA (Quantum-dot Cellular Automata), a promising successor for CMOS transistor technology. As this technology allows the implementation of logic circuits using quantum devices (quantum dots or single domain nano magnets) instead of the traditional devices (eg. transistors, diodes and resistors), a new set of tools must be developed to assist the design process. That is the case of the QCADesigner for handmade layout and physical simulation. There are also tools for Majority Logic optimization, which is the logic unit adequate to design QCA logic circuits. However, no tool for QCA layout generation was still available. This thesis proposes and develops the QCA-LG software tool to generate QCA circuit layouts. With this work, the QCA technology design flow can be completely performed in an automatic way, so that the transformation of a high level hardware description into layout is possible, although under some restrictions. The main purpose of the presented QCA-LG tool is to produce basic QCA layout suitable for optimization by hand, although in the future optimized layout can be automatically generated. At the moment, the produced layout can be a starting point for proceeding with the optimization of the circuit's layout. Some layouts automatically generated by QCA-LG are presented in this thesis and compared with optimized handmade equivalent circuits. The main causes of inefficiency of the QCA-LG tool are identified, namely in what concerns circuit area. Some conclusions reached in this work suggest the possibility of QCA-LG being a serious candidate to the automatic generation of QCA circuits.

Keywords

QCA, Quantum Cellular Automata, Automatic Layout Generation, Place and Route.

Resumo

Esta tese tem como objecto de estudo uma tecnologia nova, o QCA (Autómato Celular Quântico), uma forte candidata a suceder à tecnologia CMOS. Dado que esta tecnologia permite a implementação de circuitos lógicos com dispositivos quânticos (*quantum dots* ou elementos magnéticos com comportamento de mono domínio) em vez de dispositivos tradicionais (transistores, diodos ou resistências), é necessário criar um novo conjunto de ferramentas para auxiliar o projecto de sistemas completos. Existem algumas ferramentas já disponíveis, nomeadamente o QCADesigner para desenho manual de circuitos e para a sua simulação física. Existem também ferramentas para a síntese de Lógica Maioritária. Não existem, no entanto, ferramentas para a geração de *layout* de circuitos QCA. Nesta tese é proposta e apresentada uma ferramenta para geração automática de *layout* QCA, designada QCA-LG, que se insere no fluxo de projecto de circuitos lógicos QCA. No presente, os *layouts* gerados pelo QCA-LG não são optimizados, no entanto, espera-se, no futuro, poder realizar a geração automática de *layouts* optimizados. Actualmente, os *layouts* gerados pelo QCA-LG podem servir de ponto de partida para serem optimizados manualmente com o QCADesigner. Alguns resultados obtidos no desenrolar desta tese são apresentados e comparados com circuitos equivalentes desenhados e optimizados manualmente. Os principais problemas de eficiência da ferramenta QCA-LG foram identificados, nomeadamente o desperdício de área de circuito. Algumas das conclusões deste estudo indicam que o QCA-LG é um sério canditado para a geração automática de circuitos QCA.

Palavras Chave

QCA, Autómato Celular Quântico, Geração Automática de Layout, Colocação e interligação.

Contents

List of Figures

List of Tables

Acronym table

Acronym	Meaning
AIG	And Inverter Graph
BFS	Breadth First Search
BLIF	Berkley Logic Interchange Format
DFS	Depth First Search
CMOS	Complementary Metal Oxide Semiconductor
FIFO	First In First Out
FILO	First In Last Out
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
MQCA	Magnetic Quantum Cellular Automata
MRAM	Magnetic RAM
PLA	Programmable Logic Array
QCA-LG	QCA Layout Generator
QCA	Quantum Cellular Automata or Quantum-dot Cellular Automata
RAM	Random Access Memory
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

1 Introduction

Contents

The subject of study in this thesis is a new and promising technology, the Quantum-dot Cellular Automata. Although Cellular Automata structures have been simulated using computer software programs for a long time, the proposal of a physical realization [1] only came in 1993 from Craig S. Lent, P. Douglas Tougaw, Wolfgang Porod and Gary H. Bernstein at the University of Notre Dame [2].

The implementation of this technology may be supported in different physical devices. The basic logic gates are the majority vote, where the output takes the value of the majority of the three inputs, and the logic inversion. The basic building block in this technology is the QCA cell, that is used to obtain not only logic gates but also interconnection wires.

Given the "state of the art" of QCA technology, an automatic layout generation tool, named QCA-LG, was implemented, and is presented as the main contribution of this thesis. In the presented thesis several possible implementations of QCA systems are discussed, to proof the applicability of the concepts, and also to show the relevance of this work. Other studies in QCA related fields, such as majority logic synthesis and quantum systems simulation, are also referred in this thesis to identify the place of QCA-LG tool in the design flow of QCA systems.

1.1 Motivation

In a mean term scenario, the present CMOS technology will fail to support the growth rates needed by the semiconductor industry Therefore alternative technologies are under development. One of the main problems in CMOS is quantum tunneling through insulators, causing leakage currents that increase power consumption. Some new technologies arising, such as QCA and Single Electron Transistor (SET), have their working principles based on quantum effects, so these are no longer a problem, but a required feature instead.

QCA is a computation paradigm that can be implemented by several quantum physical systems, and at least the following have been proposed:

- metal-island quantum-dots and tunneling effect junctions [3];
- semiconductor quantum-dots and tunneling effect junctions [4];
- nano magnetic particles with single magnetic moment domain behaviour [5];
- molecular quantum-dots [6].

In Table 1.1 shows the theoretical performance limits for the referred possible implementations.

The semiconductor quantum-dot implementation is the most promising in a near future, both in terms of operation speed and high density integration. While molecular implementations would have much better performance, they do not seem easily feasible. On the other hand, magnetic QCA circuits have been fabricated and their correct behaviour was experimentally observed, but the performance bounds are much lower than for the other possible implementations.

Table 1.1: The theoretical operation speed limits for some QCA implementations.

Implementation	Operation frequencies
Molecular	THz
Semiconductor	THz
Metal island	GHz
Magnetic	MHz

Given the very high frequencies expected for molecular and semiconductor implementations, the resulting QCA systems may present much higher throughput than the current CMOS digital processors. Nevertheless, regarding latency, as ti will be seen later, QCA systems may present some drawbacks, given the extremely pipelined architectures of QCA systems.

As QCA seems very promising, the need to enhance the design flow for this technology is drawing attention. Every detail must be covered to allow a smooth transition from CMOS to QCA, and to ensure the compatibility between them. Based on the proposed QCA design flow and the already existing tools, it was identified the need for an automatic layout generation tool for QCA. It should be remembered that this problem has already been solved for digital CMOS circuits, given the efficient Place and Route commercially available tools. In this thesis a simplistic version of an automatic layout generation tool for QCA was proposed and developed.

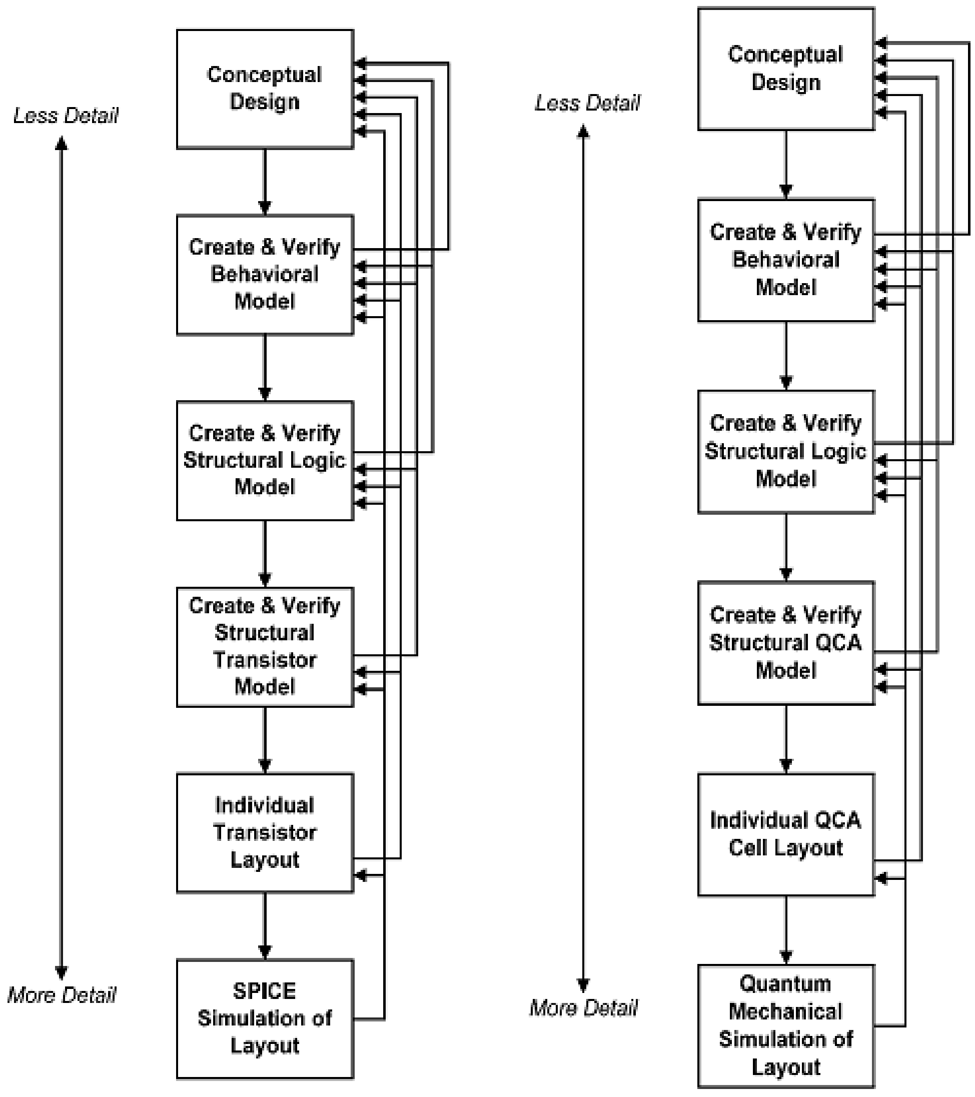
1.2 QCADesigner

QCADesigner [7] is a layout editor and simulation tool for QCA technology developed at the University of Calgary, by ATIPS Laboratory, and at the University of British Columbia, by Microsystems and Nanotechnology Group. With this software tool, layout can be drawn manually and physical simulation can be performed. Moreover the layouts can be stored in and imported in a specific file format.

This tool was extensively used during the elaboration of this thesis, firstly to validate the elemental blocks of the structures generated by the QCA-LG, and through the rest of the work, to visualize and simulate the automatically generated circuits produced by the QCA-LG.

1.3 Objectives

The main goal of this thesis is to (roughly) complete the design flow of QCA systems. The design flows of CMOS and QCA technology are presented side by side in Figure 1.1, were the common and different parts can be observed. By considering the design flow for QCA technology, as proposed by Steven C. Henderson, Eric W. Johnson, Jason R. Janulis and P. Douglas Tougaw in [8], only the last three phases of this flow (logic mapping, technology mapping and technology simulation) are different from the design flow of the current CMOS technology. Therefore, new software tools to assist these new steps are under development, being QCA-LG one of them.


Less Detail
More Detail
Conceptual Design
Create & Verify Behavioral Model
Create & Verify Structural Logic Model
Create & Verify Structural Transistor Model
Individual Transistor Layout
SPICE Simulation of Layout
Less Detail
More Detail
Conceptual Design
Create & Verify Behavioral Model
Create & Verify Structural Logic Model
Create & Verify Structural QCA Model
Individual QCA Cell Layout
Quantum Mechanical Simulation of Layout

(a) CMOS design flow. **(b)** QCA design flow.

Figure 1.1: Design flow of CMOS and QCA technologies, side by side.

Also referring to the work of Steven C. Henderson et al. [8], Very High Speed Integrated Circuit Hardware Description Language (VHDL) libraries have been described to model inherent features and basic logic blocks of QCA, such as majority gates, “in wire” memory and coplanar crossover. The representation on Hardware Description Language (HDL) enables an hierarchical organization of the circuit, as well as its logical and timing simulation.

Already existing tools can be used to synthesize logic circuits from HDL descriptions, by giving priority to optimal majority gate use in QCA. Different methods have been proposed to automatically map logic systems in QCA circuits, optimized for the preferable use of majority function, [9, 10]. The mapping can also be done with some general synthesis tools available (eg. Synopsys[11] or MVSIS[12]), but the results are not optimal, as these tools are oriented to current CMOS technology mainly supported on NAND gates.

Although SPICE models of electron tunnel junctions [13] and entire QCA [14] cells exist, they only allow to account for interactions between adjacent neighbour cells, and the current SPICE simulators can’t handle the models of QCA cells efficiently. Thus the QCADesigner simulation engines were preferred, once they perform the calculations based on the electrical field in each cell within a given (parameterizable) radius effect, in other words, these simulation engines perform a quantum physical simulation.

The objective of this thesis is to fill the gap between the logic synthesis and the physical simulation of the circuit layout, by providing an automatic layout generation tool for QCA circuits.

1.4 Main contributions

During the initial analysing of the state of the art in QCA, the need for automatic layout generation tools was perceived, and as a consequence, the main goal of this project become into providing a software tool to generate a QCA layout of a given logic circuit.

The development of such a tool was considered and carried out, and the resulting tool is named QCA-LG, were LG stands for Layout Generator. An article, T. Teodósio, L. Sousa, “QCA-LG: A tool for the automatic layout generation of QCA combinational circuits”, IEEE Norchip 2007, Denmark, was submitted and accepted for publication, [15, 16]. A website was created to present this software to the public, and it is currently available at *http://web.ist.utl.pt/ttt/qca-lg/*, where a little tutorial with some running examples can be found.

1.5 Dissertation outline

The relevant QCA background for this research is presented as the state of the art in Chapter 2, where the theoretical basis of Quantum Cellular Automata and its possible implementation are discussed.

A detailed presentation of the software tool QCA-LG is presented in Chapter 3, featuring the conceptual basis in which the tool is grounded. Furthermore, implementation details and utilization examples are given in Chapter 4. Some layouts produced with the QCA-LG are presented in Chapter 4.3, and these results are compared with equivalent handmade optimized layouts.

To conclude, in Chapter 5, future directions are pointed to improve the QCA-LG software tool, starting from the inefficiencies of the QCA-LG in its actual stage of development.

2

State of the art

Contents

Quantum-dot Cellular Automata is a technology featuring computer operations at high speed and low power consumption. It was first proposed in 1993 by Craig S. Lent, P. Douglas Tougaw, Wolfgang Porod and Gary H. Bernsteain [1], at the University of Notre Dame [2]. A comprehensive overview about this subject can be found in [17].

A more abstract concept than Quantum-dot Cellular Automata, is Quantum Cellular Automata, a computational paradigm independent of the physical implementation. This is the theory for supporting computation with quantum devices. Quantum-dot Cellular Automata, and Magnetic Quantum Cellular Automata, are two possible implementations of the Quantum Cellular Automata general concept.

Quantum-dots may be supported in many different technologies, such as metal islands, semiconductor physical dots, semiconductor electrically confined dots or even redox centers in molecules. However, regardless of the implementation, there are two simplistic requirements a system must meet in order to support QCA computation: the implementation of a QCA cell must be possible and its single behaviour must be the expected, and the cells have to be arranged in such a way that interactions between them allow to perform useful logic operations.

A single cell is expected to present a bistable behaviour, and also third a NULL state. Additionally, a given cell must be able to interact with its neighbour cells in such a way that it can influence their state, or get influenced by their state. This bistable behaviour and cell to cell coupling are illustrated in Figure 2.1.

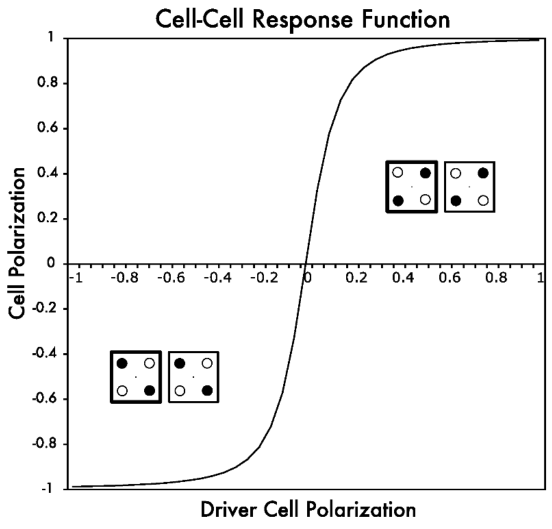


Figure 2.1: The bistable nature of a QCA cell is denoted by the abrupt polarization shift, when the cell is subjected to a smooth external influence coming from another QCA cell.

Regarding a single cell, in both cases, there are two different states, corresponding to the low energy states, and a third state with much higher energy. All other states are incorrect and lead to errors in computation, so it must be ensured they do not occur by imposing limits to the feature size and operation temperature.

The logic values '1' and '0' are encoded in the two lower energy states, ground states: by convention, the logic value '0' corresponds to the polarization value -1, and the logic '1' corresponds to the polarization value +1. The third energy state is used to control the switching between the other states, it is called the NULL state and corresponds to polarization = 0, and can only be maintained with the external influence of a "clock" signal. When the NULL state is forced by the clock, the cell accumulates energy, and when the cell is allowed to return to its ground state ('0' or '1' according to its neighbours), it releases the energy accumulated. The power needed to perform the polarization changes in cells (that support logic operations) is supplied by the clock signal any time it forces the cell to the NULL state.

A simple illustration of the energy states is shown in Figure 2.2. Please note that this plot is not accurate, the only purpose is to illustrate the concept. Note also that valid polarization values range only from -1 to +1, although the plot range is a little wider.

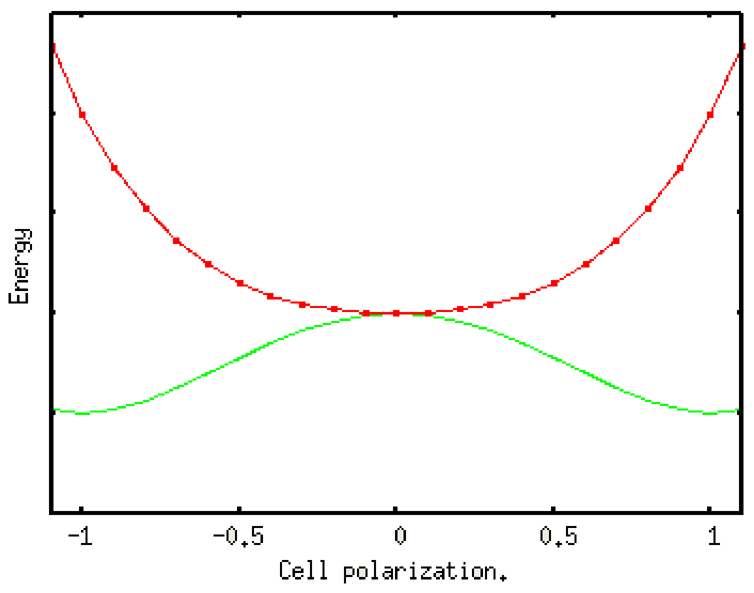
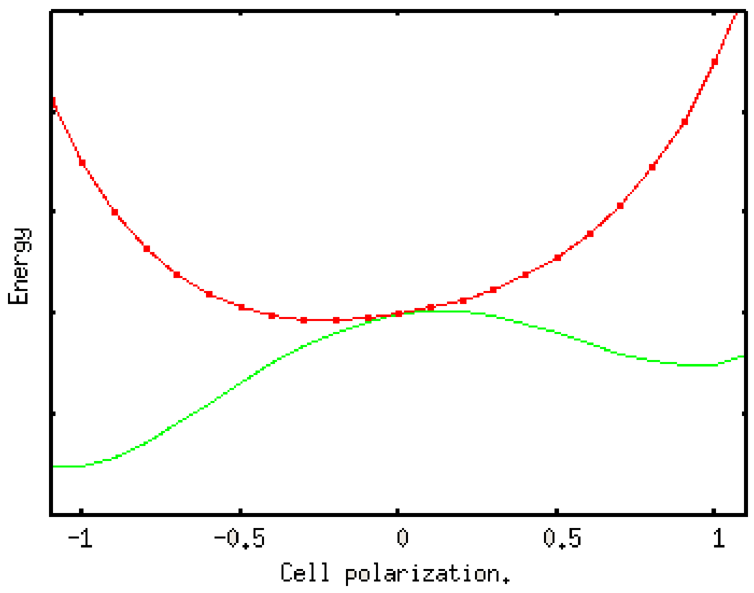
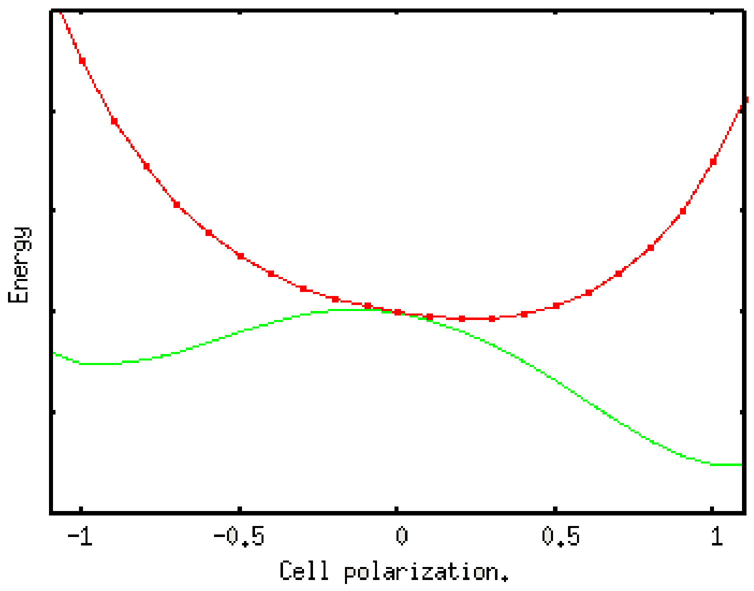


Figure 2.2: Energy states configuration of a QCA cell alone, without the influence of the clock signal (green line without markers), and with that influence (red line with markers).

In the presence of other polarized cells in the neighbourhood, the energy states shown in Figure 2.2 suffer changes and become unbalanced, as presented in Figure 2.3. There is a key issue here: if all the cell were submitted to the same clock, every clock cycle all the cells in the circuit would become in the NULL state, and thus, no useful operation could performed. The solution to this problem is to have a four clock system, were each clock signal has a different phase, being separated by a quarter period delay. This issue is explained in more detail in Section 2.4.



(a) The neighbour cell has polarization = -1 (logic ’0’)



(b) The neighbour cell has polarization = +1 (logic ’1’)

Figure 2.3: Energy states configuration of a QCA cell in the presence of other polarized cell, without the influence of the clock signal (green line without markers), and with that influence (red line with markers).

2.1 Physical structures

The physical structures designed to have the described properties are shortly introduced here. One of the proposed implementations of the Quantum Cellular Automata is the Quantum-dot Cellular Automata. Quantum-dot Cellular Automata isn't a physical implementation yet, it is rather a lower level abstraction, since there are several ways to build the quantum-dots and connect them. In this case the cells are made of four quantum-dots placed in the corners of a square, populated with only two electrically identical charges. Given the electrostatic interactions (repulsion) between the charges, these will tend to occupy diagonally opposed quantum-dots. There are only two stable configurations, as there are only two diagonals in a square, and these two stable configurations are the two lower energy states referred above: they encode the binary values '0' and '1' (see Figure 2.4). The NULL state configuration depends on the physical implementation; for example in some cases two extra quantum-dots are placed in the center of the cell to be occupied during this state.

(a) '0' **(b)** '1'

Figure 2.4: Representation of the two logic values of a QCA cell with four quantum-dots. Black filled circles represent occupied quantum-dots while white filled circles represent unoccupied quantum-dots. a) Logic value '0', b) Logic value '1'.

Quantum-dots can be any charge containers, with discrete electrical energy states (there may be more than two states, but only two are used), sometimes called artificial atoms. Some molecules have well defined energy states, and therefore, are suitable for supporting the operation of QCA systems. Small metal pieces can also behave as quantum-dots, if the energy states an electron can occupy are distinguishable, instead of the usual energy band. This means that the difference between two consecutive energy states must be well above the thermal noise energy (k_bT, being k_b the Boltzmann constant and T the absolute temperature).

By considering the "particle in a box" approach, the maximum dimensions of a quantum-dot can be estimated as follows. The possible values for the wavelengths of a particle, λ, in a container with size L are shown in Figure 2.5. Equation 2.1 reflects the fact that, half integer number of the wavelength of the particle must fit into the length of the quantum-dot (L).

$$n\frac{\lambda}{2} = L, \tag{2.1}$$

The first "de Broglie" relation, shown in Equation 2.2, allows the calculation L, of the quantum-dot corresponding to a given linear momentum (p), of the particle inside (n, is the number of nodes

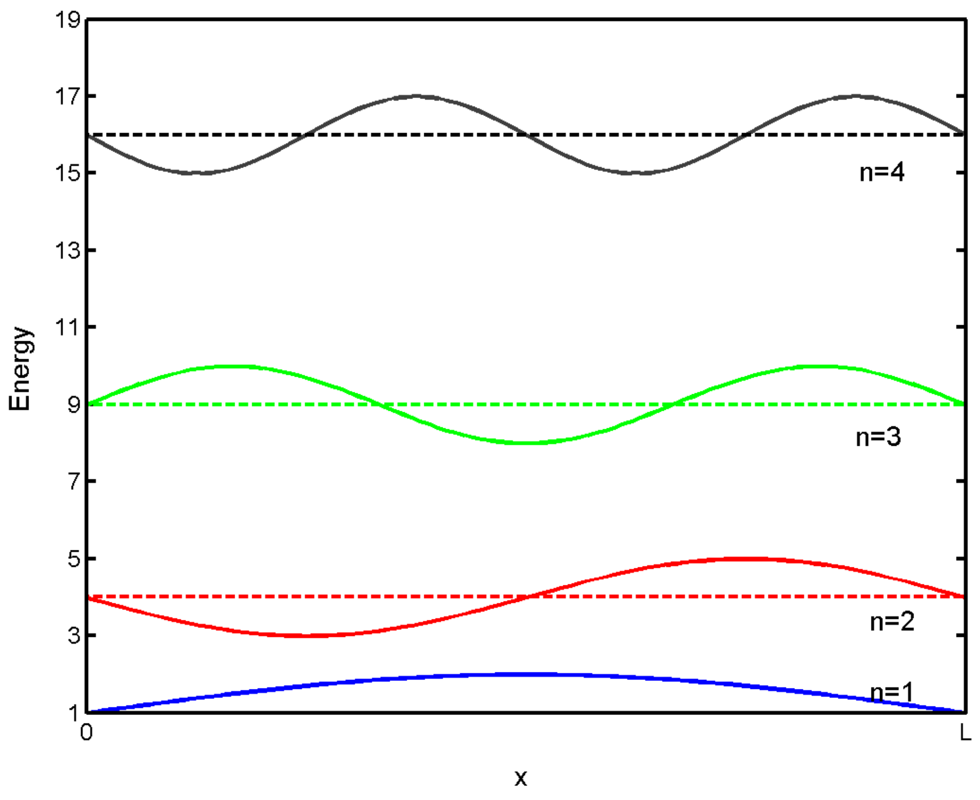


Figure 2.5: Possible wave functions for a "particle in a box" of size L; n is the number of nodes of the wave function.

of the wave function, and h is the Plank's constant).

$$p = \frac{h}{\lambda}, \tag{2.2}$$

Substituting λ calculated from Equation 2.1 in Equation 2.2, Equation 2.3 is obtained.

$$p = \frac{hn}{2L}, \tag{2.3}$$

Given the classical formula relating energy to momentum (Equation 2.4), the energy gap between particles associated with the two larger possible wavelengths can be found as stated in Equation 2.5, where the mass of an electron is considered by making $m = m_e$.

$$E = \frac{p^2}{2m} = \frac{h^2 n^2}{8 m_e L^2}, \tag{2.4}$$

$$E_{n=2} - E_{n=1} = \frac{4h^2}{8 m_e L^2} - \frac{h^2}{8 m_e L^2} = \frac{3}{8} \frac{h^2}{m_e L^2}, \tag{2.5}$$

There is in Equation 2.6 the necessary condition for the correct behaviour of a single quantum-dot, in energy terms.

$$E_{n=2} - E_{n=1} \gg k_b T, \tag{2.6}$$

Finally, Equation 2.7 leads to an estimate value for the limit size of a quantum-dot at room operation ($T = 300\ \mathrm{K}$).

$$k_b T \ll \frac{3}{8}\frac{h^2}{m_e L^2} \Rightarrow L \ll \sqrt{\frac{3}{8}\frac{h^2}{m_e k_b T}} = 6.6\ \mathrm{nm}, \tag{2.7}$$

To achieve correct room temperature operation of QCA individual cells, not only the gap between energy levels in a single quantum-dot must be greater than the thermal noise, but also the difference between the two lower energy states and the third higher energy state in a QCA cell must be clearly greater than the thermal noise.

The conditions imposed before for the energy are necessary but not sufficient to ensure QCA systems proper function, as until now the requirements are only focused the correct behaviour of quantum-dots and single QCA cells. The conditions for the correct interaction between cells must also be ensured. If we consider, for instance, a cell wire composed by a finite number of cells, all lined up forming a linear array. Moreover in the initial condition, all the cells have the same polarization, so the system is in ground state. Suppose then the cell in one extremity of the wire is externally forced to change its polarization, and suddenly a “kink” in polarizations appear between the first and the second cell of the array. This “kink” must then propagate through the array until it reaches the other extremity. Then the array would have returned to its ground state, having all cells the same polarization as in the beginning. The propagation of the “kink” is similar to a soliton, according to P. Douglas Tougaw and Craig S. Lent in [18].

In Figure 2.6 a sequence of states is presented, to illustrate the propagation of a “kink” in the cell polarization of a in QCA cell array. The electrostatic energy of a system composed by two cells (cell a and cell b, with respective polarizations p_a and p_b), side by side, is given by the Equation 2.8. The total energy of the two cells is calculated by the sum of the electrostatic energy between each of the four quantum-dots of cell a, (with charge q_i^a and location r_i^a) and each of the four quantum-dots of cell b, (with charge q_j^b and location r_i^b); both i and j range from 1 to 4, as there are 4 quantum-dots in each cell.

$$E^{a,b} = \frac{1}{4\pi\epsilon}\sum_{i=1}^{4}\sum_{j=1}^{4}\frac{q_i^a q_j^b}{|r_i^a - r_j^b|}, \tag{2.8}$$

$$E_{kink} = E^{a,b}_{p_a \neq p_b} - E^{a,b}_{p_a = p_b}, \tag{2.9}$$

This energy has a minimum value when the two cell have the same polarization, and has the maximum value when they have opposite polarizations. The difference between these maximum and minimum values is called the “kink” energy, and it is represented in Equation 2.9.

Please note, once the “kink” energy is defined as a difference between energies, the calculations of these energies may neglect some common contributions. As these calculations are done presuming each cell is always at its ground state, which means it exhibits a bistable behaviour,

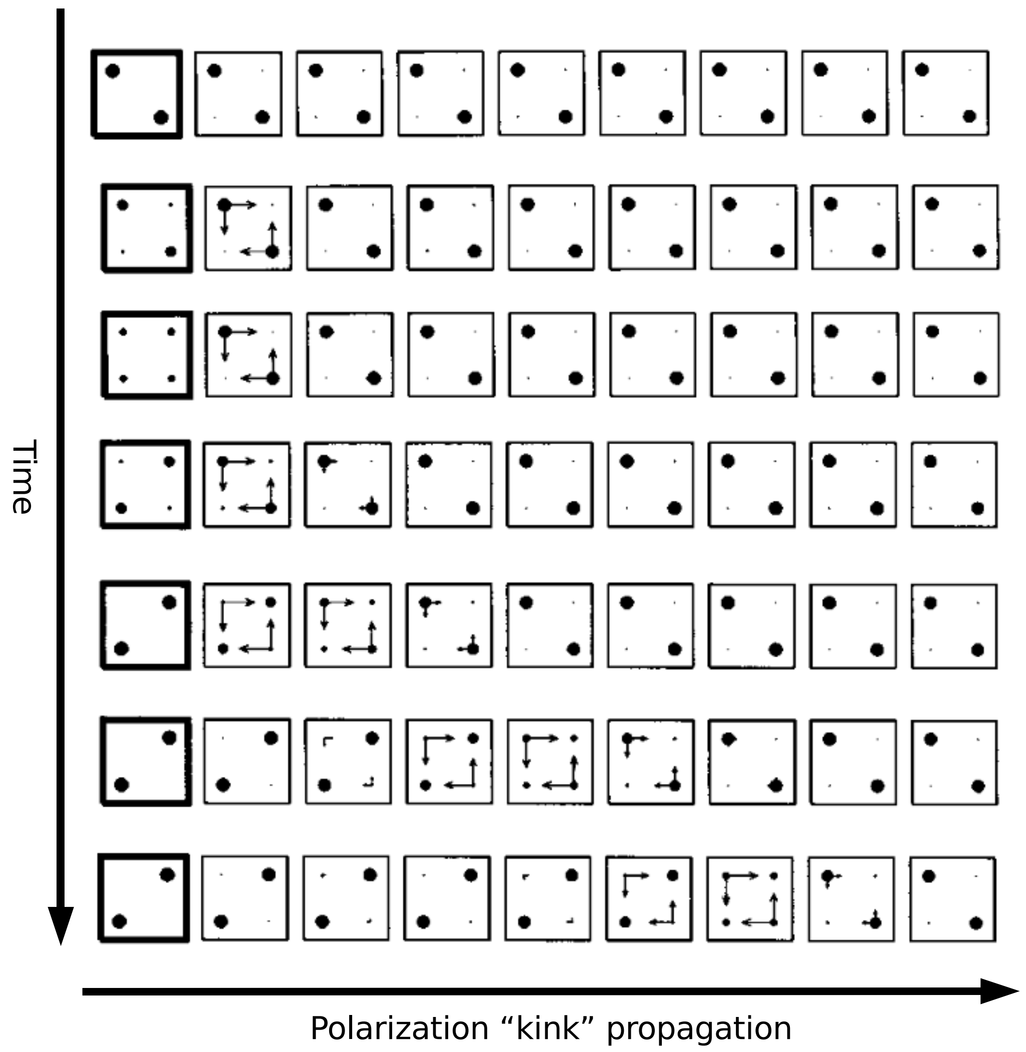


Figure 2.6: Propagation of a polarization “kink” along an array of QCA cells. The several snapshots of the array cells’ state show how the polarization of the cells evolves in time.

there is no need to account for the interactions between the quantum-dots inside a single cell, because this has the same value for each cell either there is a "kink" or not.

The values of the "kink" energy estimated by Konrad Walus and Graham A. Julien in [17], for a different kinds of QCA systems, are presented in Table 2.1.

Table 2.1: The estimated values for the "kink" energy between two adjacent cells and the consequent limit for clock zone length obtained from Equation 2.10.

Cell Type	Cell Size	Kink Energy	Max. cells per clock zone
Molecular QCA (ϵ_R=1)	$<$ 2 nm	$>$ 0.3 eV	1.0959 * 10^5
Self-Assembled	5 nm	9.13 meV	1.4236
Lithographically Defined	10 nm	4.56 meV	1.1929
Lithographically Defined	20 nm	2.28 meV	1.0922

The relation between the "kink" energy and the maximum length (N QCA cells) of a wire within a clock zone, to ensure a "kink" free operation, was presented by Vankamamidi, Ottavi and Lombardi in [19], and is shown in Equation 2.10.

$$N \leq e^{\frac{E_{kink}}{k_b T}}. \tag{2.10}$$

Given the fabrication feature size needed to ensure a correct room temperature operation of quantum-dot base devices, assuming that Moore's law will remain valid in the near future, and considering that in the present days 45 nanometers feature size is a reality, approximately in 6 years the feature size will be around 3 nanometers, and then QCA may compete to be commercial viable.

Besides to the proper operation of quantum-dots, the "kink" energy must be high enough to allow the coherent operation of large or medium sized arrays of QCA cells under the same clock. The lower boundary for the size of clock regions is three cells in height and width, because this is the area needed to fit a majority gate (as will be shown in Section 2.3), which must be in a clock zone of their own.

The operating temperature does also limit the size of the magnetic QCA elements, but in a different way. The main limitations regarding the behaviour of nanomagnets, as suitable elements for QCA operation, are the boundaries for the single domain properties, which are the thermal noise energy and the size beyond which multiple magnetic domais form to minimize the system's energy. In Figure 2.7 it is presented the typical plot of the coersive intrinsic magnetic field for a magnetic element (eg. permalloy) of sizes near the 100 nm. On the left side of the graphic, the particle loses all coercivity when the thermal noise energy is enough to make the magnetic moment of the particle change randomly, exhibiting a super-paramegnetic behaviour. On the other

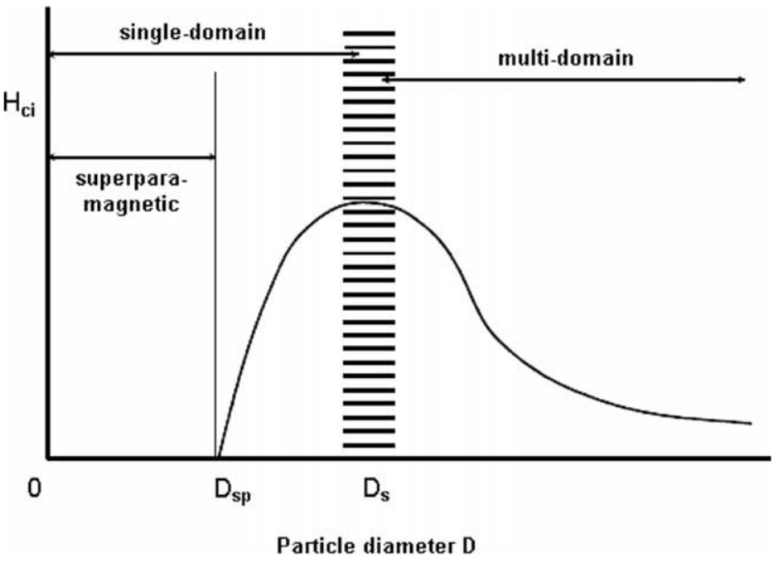


Figure 2.7: The coercive intrinsic magnetic field of a particle in function of its diameter.

hand, on right side the particle's magnetization breaks into multiple magnetic domains to minimize the magnetic energy.

2.2 Basic logic elements

The logic elements, the QCA cells, have common properties despite the implementation chosen, while the physical active elements in QCA technology can be quantum dots, tunnel junctions, nano magnets or molecules. Figure 2.8 shows the symbol representation of a QCA cell.

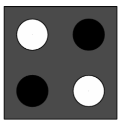


Figure 2.8: Symbolic representation of a QCA cell.

In the absence of any external influence there are two stable energy states, and these two states can be used to encode binary values “0” and “1”. When an external influence (electrical field or magnetic field) is present, the lowest energy configuration may change, and the cell can be forced into a new neutral state, which means the cell activity can be controlled. Some implementation proposals will now be detailed.

2.2.1 Quantum dot implementation

The configuration of charges in quantum dots represent logic values as explained in Section 2.1, and although there are several possible quantum-dot implementations, as described before, the functional principles are common.

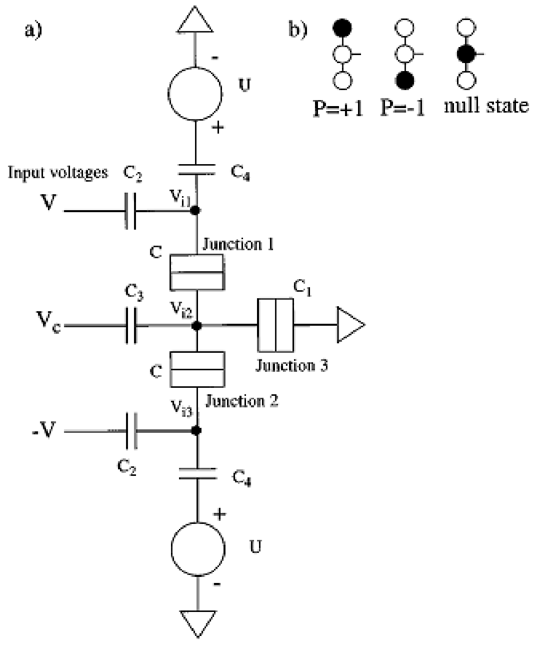


Figure 2.9: a) Schematic of a QCA half cell, b) Description of the polarization states.

The electrical schematic of a half cell with three quantum dots (V_{i1}, V_{i2} and V_{i3}) and two tunneling junctions ($Junction1$ and $Junction2$) is presented in Figure 2.9; the complete QCA cell constituted by two half cell side by side. The three quantum-dot system is populated by only one free charge (electron), which can move only through the tunnel effect junctions. The input signal is the voltage V, which is applied as $+V$ and $-V$ to the left plate of capacitors C_2. The electrical charge on nodes V_{i1} or V_{i3} will be attracted to (or depleted from) the right C_2 capacitors plate according to voltage V. The middle node V_{i2} allows to control the operation of the half cell through V_c, as it is possible to attract the charge from the other quantum-dots to the middle one (setting the NULL state), and also to repel the charge previously attracted to either one of the other two quantum-dots, V_{i1} or V_{i3}, setting a well defined state, $P = +1$ or $P = -1$, respectively. The whole half cell is settled at a given electrical potential U above the substrate, and the tunneling junction $Junction3$ is used to retrieve a unique free charge from the substrate as U is set accordingly. In general terms, the operation of such a half cell consists of the following steps:

- Set U to initialize the system, pulling a single free charge from the substrate into the central quantum-dot.
- Apply a clock signal to V_c in order to make the system synchronous, with defined periods of NULL state polarization alternating with well defined polarization state ($P = +1$ or $P = -1$)
- For the input half cells - set the input voltage V to define the polarization state in the next period of time.

- For the other half cells not externally available - the electrical field from the charges in the neighbouring half cells will be the input instead of voltage V.

In Figure 2.10 are the possible charge configurations and the respective polarization of QCA cell. In this case the full QCA cell is constituted of six quantum-dot, being the two middle ones used to

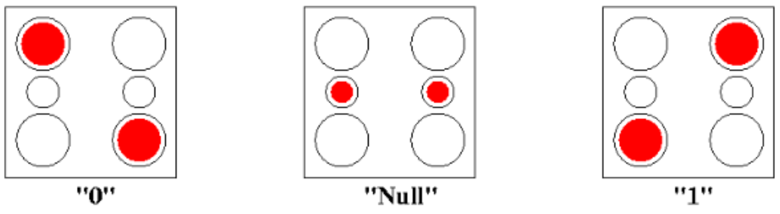


Figure 2.10: The three possible charge configurations in a six dot QCA cell and the respective polarization (below).

force the third (higher energy) state, used to drive the cell to a NULL state. Other constructions were proposed, such as the one presented in Figure 2.11.

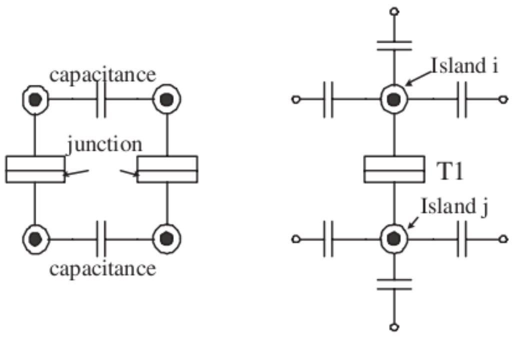


Figure 2.11: QCA cell (on the left), and the respective hafl cell (on the right).

In the six quantum-dot per cell implementation the basic principles remain, but the operation slightly changes. The two dots in the middle are forced to receive the two charges while the cell remains in the NULL state, being this the third state, the one with higher energy. Assuming the charges are electrons, then a positive electrical potential imposed to the middle dots would attract the negatively charged particles. On the other hand, a negative potential would repel the electrons to the top or bottom dots, forcing the cell to define its polarization. Thus the clock signal can be capacitively coupled to the middle dots, in order to impose the desired potential and make the cell pulse. The cells affected by this clock signal will be in the NULL state whenever it is “high”, and will have a well defined polarization (’0’ or ’1’) whenever the clock is “low”. During the raising transition of the clock signal, the cells loose their polarization progressively, and during the falling transition, the cells are compelled to reach a defined polarization. The clock must have raising and falling times large enough to permit the charges to settle in the lowest energy configuration, the ground state.

Consider the following scenario, due to some undesired influence, the polarization of a cell in NULL state isn't exactly zero, as if the cell's polarization was slightly biased. Then when the clock starts to fall, taking the cell from NULL state to a well defined '0' or '1' state, the charges may have not enough time to reach the lowest energy dot, and could stay trapped in some kind of local energy minimum. This would lead to incorrect cell behaviour, so the dynamics of the cell have to be taken into account to avoid this kind of problems.

Regarding the energy dissipation during transitions, the smoother the transition, the more efficient the operation is. The adiabatic operation of a cell is possible, and it leads to minimal power dissipation. This concept is also referred to as reversible computation.

2.2.2 Magnetic implementation

In the magnetic implementation, elements are shaped in such a way that two lower magnetic energy states are created. These energy states correspond to having the magnetization aligned with the preferential axis, called the “easy axis”.

The logic values '0' and '1' are encoded in the direction of the magnetization, being the possible directions anti parallel they are clearly distinguishable. The third energy state, with higher magnetic energy, occurs when the magnetization is perpendicular to the “easy axis”, and thus, being aligned with the called “hard axis”.

To have the magnetization aligned with the “hard axis” is only possible by applying an external magnetic field strong enough to overcome the anisotropy energy term. The binary behaviour of nano magnetic elements is used, for example, in Magnetic RAM (MRAM) devices.

The clock of a Magnetic Quantum Cellular Automata (MQCA) cell is applied as the amplitude of the magnetic field along the “hard axis”, which drives the cell into the NULL polarization state and back into a well defined state, periodically. When the clock signal has value zero, the cell exhibits its bistable behaviour and is suitable to hold a binary value.

In Figure 2.12, a majority gate made of elongated nano magnets is shown. The inputs are applied from the top, bottom and left side nano magnetic elements, and the output will be retrieved from the right most nano magnetic element. The magnetic poles are shown as bright and dark regions on top of the element shapes delimited by the black borders, on the two right most images, The poles are in the extremities of the nano magnets, thus the magnetization must be along the nano magnets from one pole to the other. The two magnetization configurations presented are completely opposite, as all magnetizations switch their orientation but not their direction; both evidence the expected behaviour of the nano magnets forming the QCA majority gate.

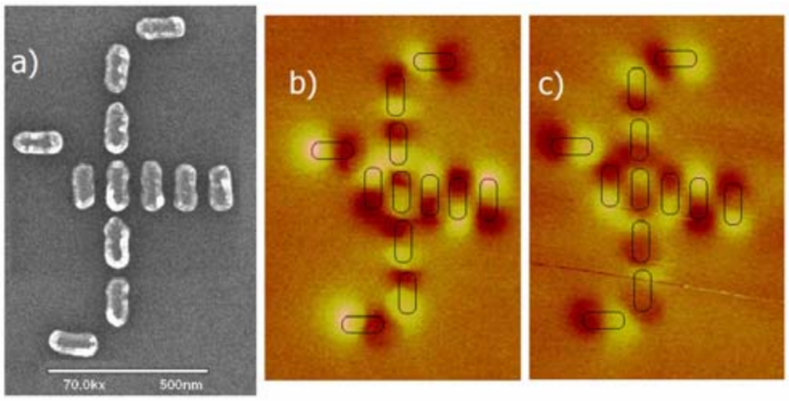


Figure 2.12: QCA magnetic majority gate image on the left, and magnetic polarization images on the center and right show the correct behaviour of the nano magnets.

2.2.3 Molecular implementation

The molecular implementation is perhaps the most complex, and involves chemistry knowledge beyond the scope of this project, yet a brief discussion is here presented. The main motivations for the development of molecular QCA are the high density of devices and the operation at room temperature, which both depend on the size of the quantum-dot and QCA cells. Even if the molecules could be synthesized and placed according to the desired patterns, there would still be the challenge of interconnecting them with the conventional electronics for read/write operations.

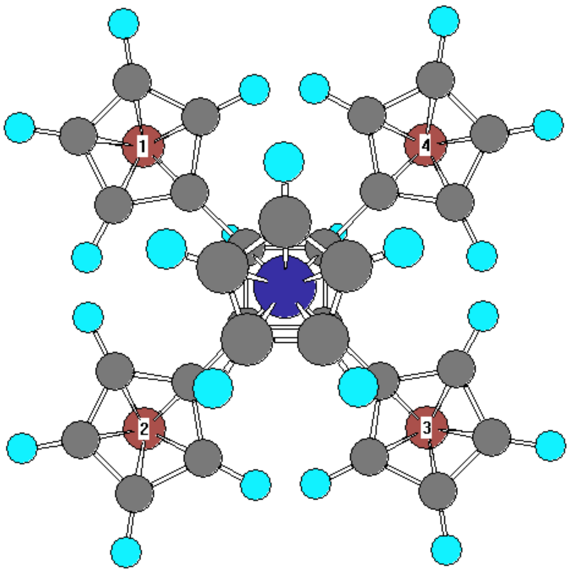


Figure 2.13: A possible QCA cell molecule.

A molecule proposed by Fehlner *et al* as a possible QCA cell is presented in Figure 2.13 In this case the key strategy is to use non-bounding orbitals (π or d) at the redox centers to act as quantum-dots, as these orbital can have fluctuations on the number of occupant electrons.

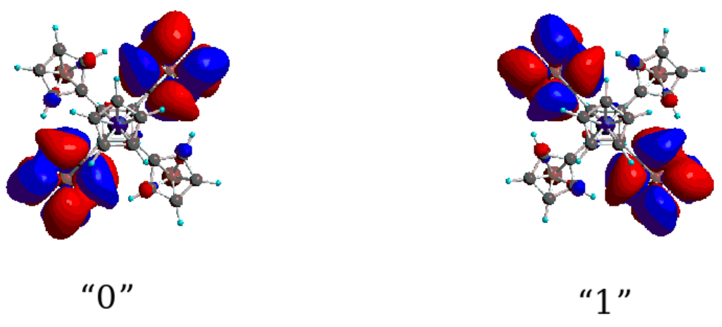


Figure 2.14: The two possible QCA cell molecule polarizations and the respective logic values.

The two possible polarizations of this molecule are shown in Figure 2.14, and the cell to cell coupling is depicted in Figure 2.15, to evidence that the interactions between these molecules make them suitable for QCA. The extra charges in the molecule can be defined by reduction or oxidation of the functional groups.

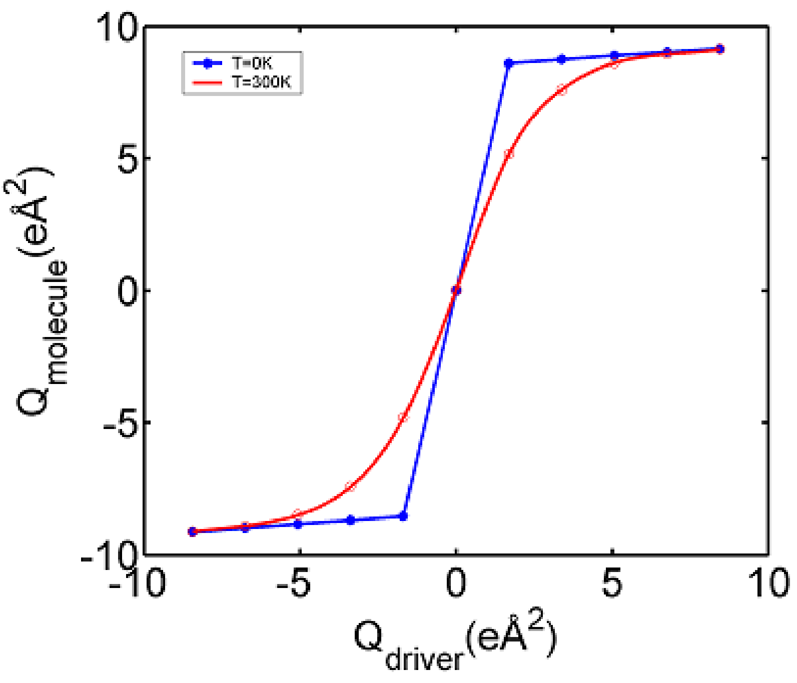


Figure 2.15: The Coulomb coupling interactions between two QCA cell molecules.

2.3 Logic gates

Logic operations are performed by means of the interactions between adjacent cells. The basic logic operation is the majority vote of three inputs and the majority gate shown in Figure 2.16. Let us consider the five central cells in Figure 2.16, assuming that the inputs are imposed by cells A, B and C. Given the Coulomb interactions explained in Section 2.1, and linear superposition of the electrical field, the central cell will sense the field imposed by the top, left and bottom neighbour cells by assuming a polarization equal to the polarization of the majority of these three inputs. Although there may be a "kink" between one of the inputs and the central cell, that configuration is still in the ground state, given that the central cell will always switch as needed to achieve the lowest energy state of the local system. Therefore there can never be two or three "kinks" in this system, and if in some transitory moment it happens, the central cell will switch its polarization to become coherent with the majority (two or three) of the neighbours. After the central cell switches to make the system reach the ground state, the cells on the right side will also switch accordingly to avoid "kinks", thus ensuring the ground state is preserved.

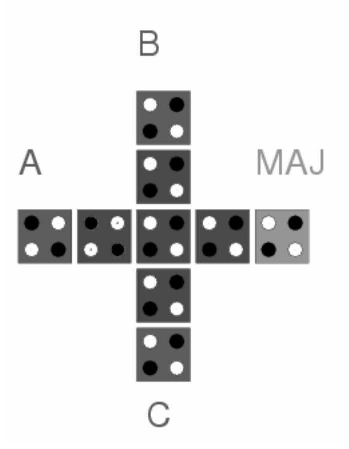


Figure 2.16: QCA layout of a majority gate.

The majority logic function performed is $M(a,b,c) = a.b + b.c + a.c$. Therefore the logic product can be performed as $M(a,b,'0')$ and the logic sum as $M(a,b,'1')$. The possibility of having a selectable AND/OR gate may be an advantage for implementing dynamically programmable logic circuits, which may be another way to take advantage of QCA technology.

A new approach is taken in this thesis to implement signal inversion, which is simpler than the most common ones. As the most basic element in QCA is half cell [20], every wire can be seen as a set of linearly disposed QCA cells or the corresponding half cells. It is also known that a wire of half cells works as an inverter chain [21]. Therefore the inversion of a signal can be attained only by adding or removing a half cell to a given wire. In the present case, it was chosen to remove a half cell and to distribute the spare space along the inter cell spacing in a wire, so the

Table 2.2: The logic truth table for the three input majority function.

A	B	C	MAJ
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

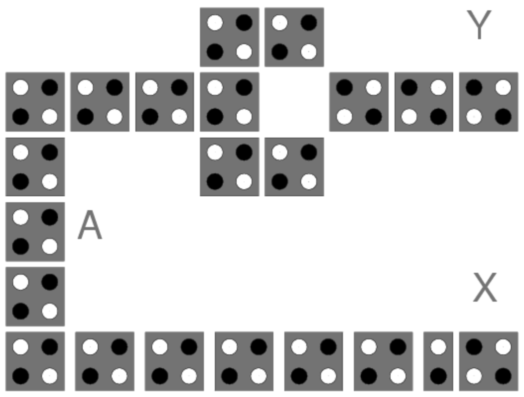


Figure 2.17: Comparison between the traditional and the proposed structure for signal inversion. The input cell is labeled as A, the usual inverter's output is labeled as Y and the output of the in wire inverter is X.

inverter wire stays exactly with the same start and end points it would have if made of complete cells. In order to safely remove a half cell from a given wire, it has to have a minimum number of cells that constitute the wire. Otherwise, when distributing the half cell distance among the inter cell spacing along the wire, cells may not interact properly given the increased distance between them.

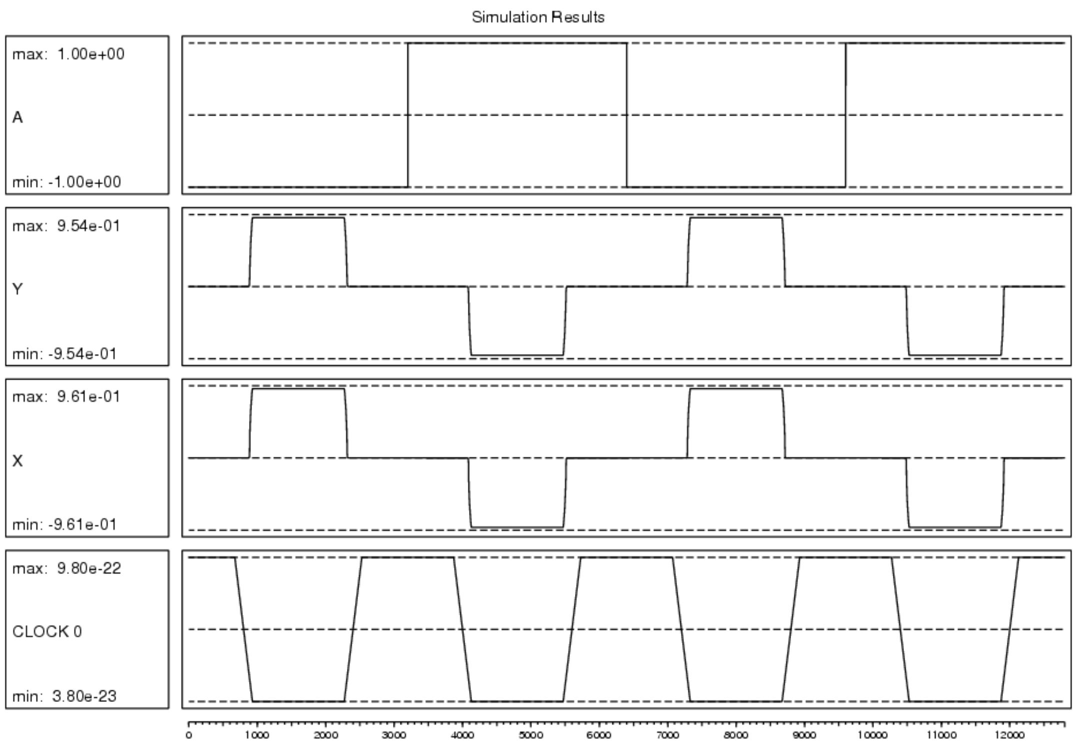


Figure 2.18: The presented simulation results show the functional equivalence between the two structures show in Figure 2.17. Signal A is the input and signals X and Y are the outputs.

Some inverting wires that adopt the proposed scheme were simulated using QCADesigner, to evaluate its proper response. One of the layouts used in the tests is presented in Figure 2.17, and its simulation results are shown in Figure 2.18. The results for the proposed inverting structure depicted in Figure 2.18 are equivalent to the ones obtained for the commonly used structure.

2.4 Synchronization

QCA has a clocking mechanism that consists on four clock signals with equal frequencies. One of the clock signals can be considered the reference ($phase = 0$) and the others are delayed one ($phase = \pi/2$), two ($phase = \pi$) and three ($phase = 3\pi/2$) quarters of a period (see Figure 2.20). Each clock signal imposes its pace to a given set of layout regions, as explained before in Subsection 2.2.1. Signals are pushed from one clock region to another as the phase of the

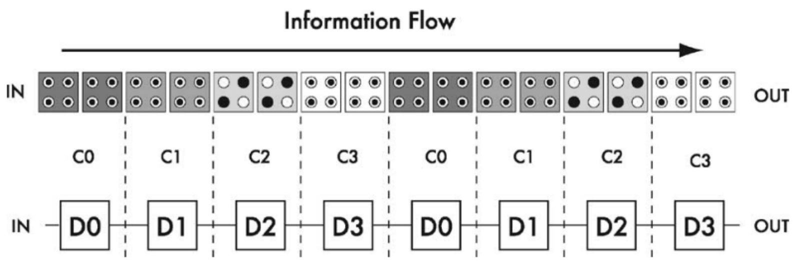


Figure 2.19: The signals are pushed through the clock zones as if the wire was a D latch chain.

clock of these regions increases, as depicted in Figure 2.19. A given clock zone receives the signal from adjacent clock zones, which have a clock in advance by one quarter period. Regarding the QCA wire represented in Figure 2.19, which corresponds to the moment marked with a thick vertical red line in Figure 2.20, the following considerations can be made about the represented moment in time:

- cells in clock zone $C0$ are in the NULL state;
- cells in clock zone $C1$ are relaxing from a well defined state to the NULL state, thus loosing their polarization;
- cells in clock zone $C2$ are in a well defined state;
- cells in clock zone $C3$ are being forced into a well defined state, and their polarization will be set accordingly to their neighbours polarization.

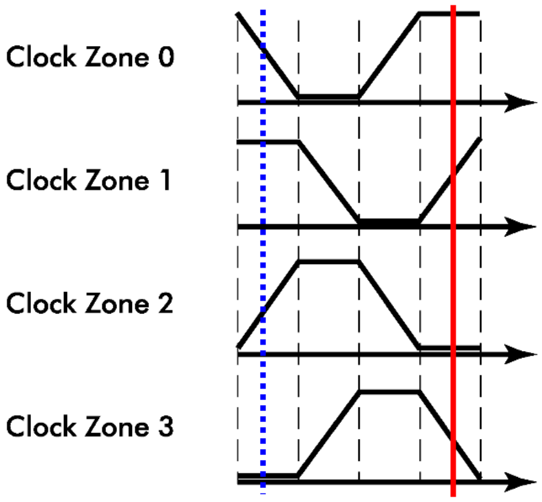


Figure 2.20: The four clock signal needed to control QCA circuits.

After a quarter period, in the moment of time marked with a thick vertical blue dotted line in Figure 2.20, the states will be the following:

- cells in clock zone $C0$ are being forced into a well defined state, and their polarization will be set accordingly to their neighbours polarization.
- cells in clock zone $C1$ are in the NULL state;
- cells in clock zone $C2$ are relaxing from a well defined state to the NULL state, thus loosing their polarization;
- cells in clock zone $C3$ are in a well defined state;

Therefore, the logic value encoded in the state of cells in clock zone $C2$ is pushed forward into the cells in clock zone $C3$. Detailed information on this issue can be found in [22].

2.5 Memory

The clocking mechanism of QCA systems allows each clock zone to act as a memory cell, for one quarter clock period, so connecting four subsequent clock zones will produce a memory effect during one clock period. Thus, as shown in Figure 2.21, applying feedback to such a wire, allow us to obtain a memory.

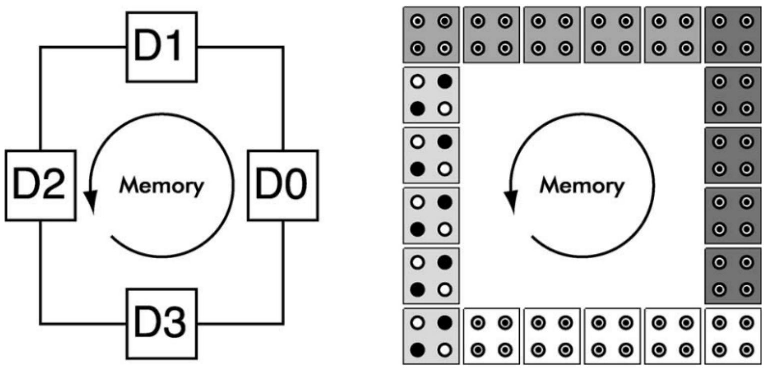


Figure 2.21: A basic memory loop of QCA cells.

Given the basic memory cell construct, a value has to be inserted into the loop and read from it, in order to make it useful. Some additional logic has to be added around and also within the memory loop [17], as presented in Figure 2.22. This memory cell topology shows two Majority gates acting as AND and OR gates incorporated in the memory loop which are used to set value stored to ’0’ or ’1’, respectively.

2.6 Signal routing

In QCA technology, connection wires are made of QCA cells (like the logic gates).

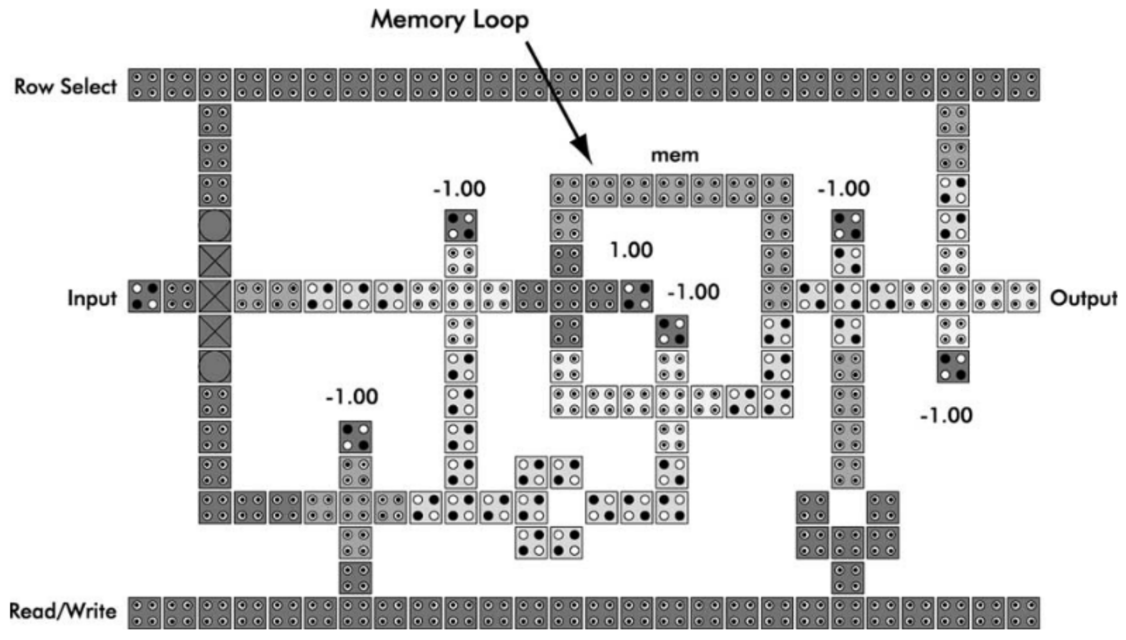


Figure 2.22: A 1 bit memory cell with data input, control signals and data output.

Two ways have been proposed to solve wire crossing conflicts: *in plane* crossing, shown in Figure 2.23, and *multi layer* crossing, shown in Figure 2.24. By observing the *in plane* crossing, in Figure 2.23, it should be noticed that the vertical wire has all the cells rotated by 45 degrees, relative to the cells in the horizontal wire. With such organization there is no interference, as two cells with a 45 degree rotation relative to each other have no “kink” energy associated to their polarizations. But a problem may arise from the increase in distance between the cells of the horizontal wire, right in the crossing point, since the “kink” energy is lower than with the normal spacing, and transmission error may occur.

In this thesis *multi layer* crossover was considered, but with small changes *in plane* crossover could also be accommodated. Although *multi layer* crossover gives better simulation results (seems safer), it may not be as easily fabricated as *in plane* crossovers, thus a compromise between the best (in theory) and the possible must be assumed. And regarding *multi layer* crossover, it should be noticed that for an even number of separation layers the via connection will act as an inverter.

2.7 Majority logic synthesis

Given the particular characteristics of QCA, such as the single cell based layout, or majority logic benefits, it becomes necessary to have a special care regarding logic synthesis. For example, some usual optimizations applied for CMOS based logic, such as the extensive use of NAND gates, must be avoided. Therefore, some mapping techniques and logic representations often

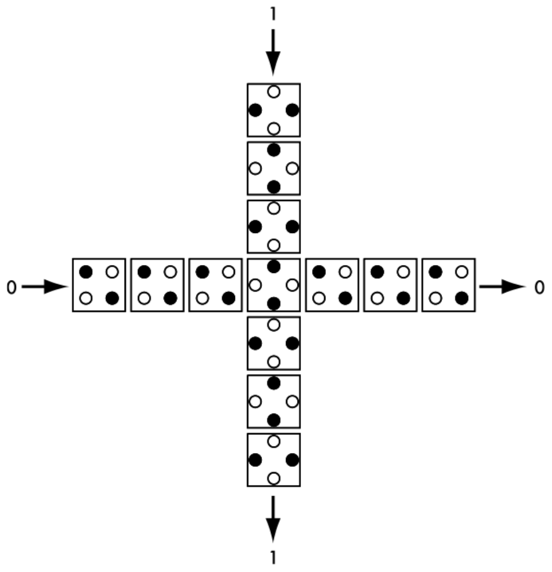


Figure 2.23: In plane crossing of two independent QCA wires.

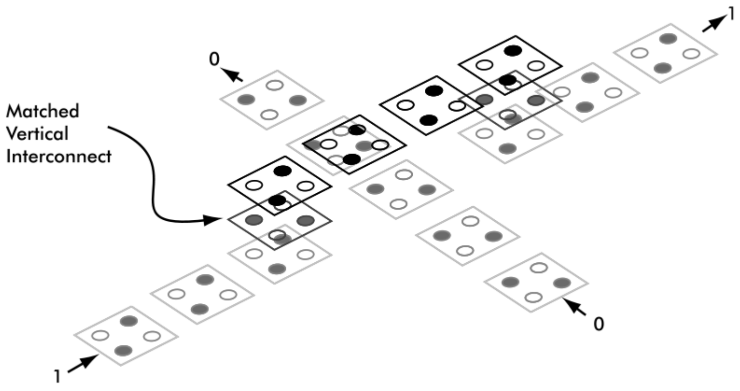
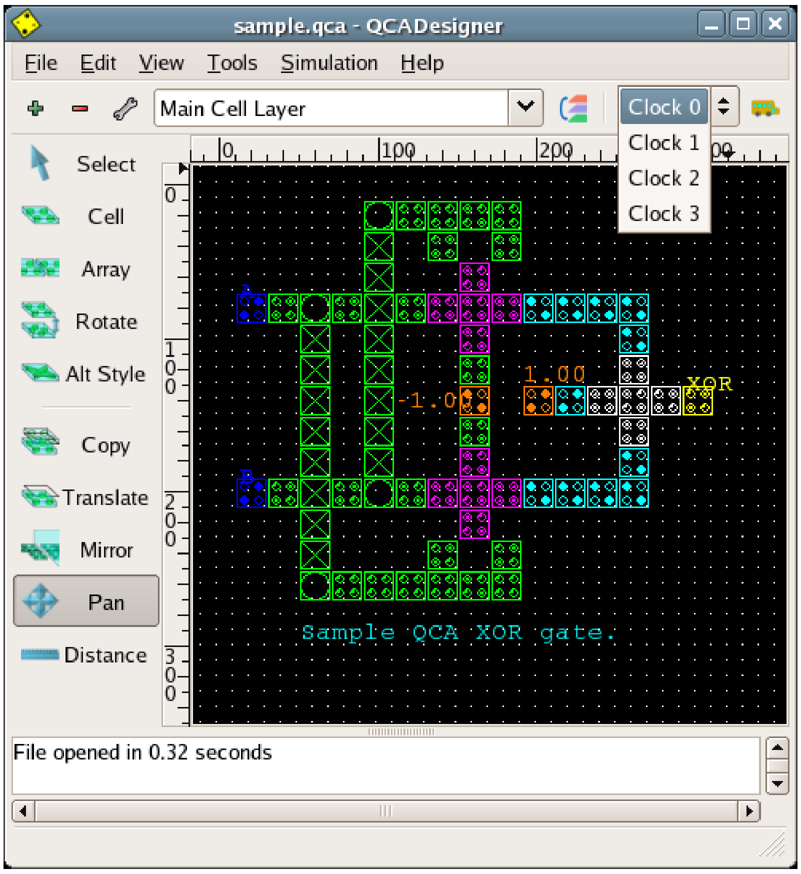


Figure 2.24: Multi layer crossing of two independent QCA wires.

used are not suitable for majority logic optimization. For instance, the And Inverter Graph (AIG) is a very good representation to produce logic circuits using only NAND gates and Inverters, but it is not suitable for producing logic circuits optimized for majority logic gates.

To deal with majority logic synthesis problem new approaches may have to be taken. New synthesis tool can be engineered from scratch, with new supporting data structures and new decomposing/manipulating primitives, where majority function is a basic logic gate and logic functions can be decomposed into it. Some work on this logic has been done, as it is the case of [9] and [10], but much more work is expected since this is a key issue to the success of QCA.

2.8 QCADesigner



File opened in 0.32 seconds

Figure 2.25: QCADesigner layout editor window.

QCADesigner is the “state of the art” QCA layout editor and simulator. The main layout design window of QCADesigner is presented in Figure 2.25. The physical layout editing facilities include:

- Drawing QCA cells individually or in arrays, optionally aligned to a grid with a default spacing (20 nm) equal to the default cell size (18 nm) plus the default inter cell spacing (2 nm).

- Setting clock signal for each QCA cell, which is required to have synchronous circuits working properly.
- Multi-layer QCA layout design, which is required to have *multi layer* signal crossing.
- Drawing QCA cells with 90 degrees rotation, which is required to have *in plane* signal crossing.
- Graphical marking of special cells (on via and crossover layers), according to the convention presented in Figure 2.26. Although cell in via and crossover structures may look different in the layout, they are regular QCA cells and no distinction is made during simulation. Cells acting as vertical via interconnections between layers are represented by a square with a circle inside, and cells in crossover layers are represented by a square with a cross inside, the normal cells are represented as a square with four little circles inside and the arrangement of those circles depend on the rotation of the cell.

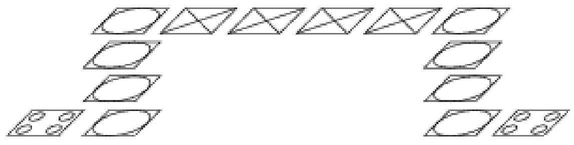
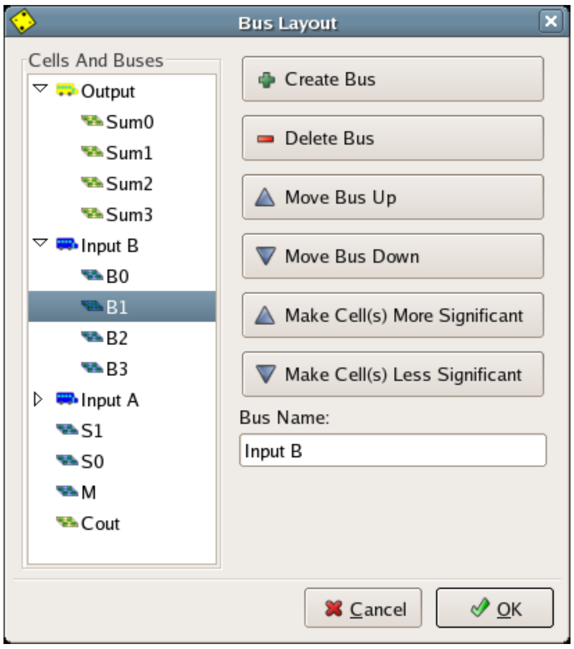


Figure 2.26: QCA cell style convention used to visually distinguish the cells on the Main Cell Layer from the cell used in via and crossover connections.

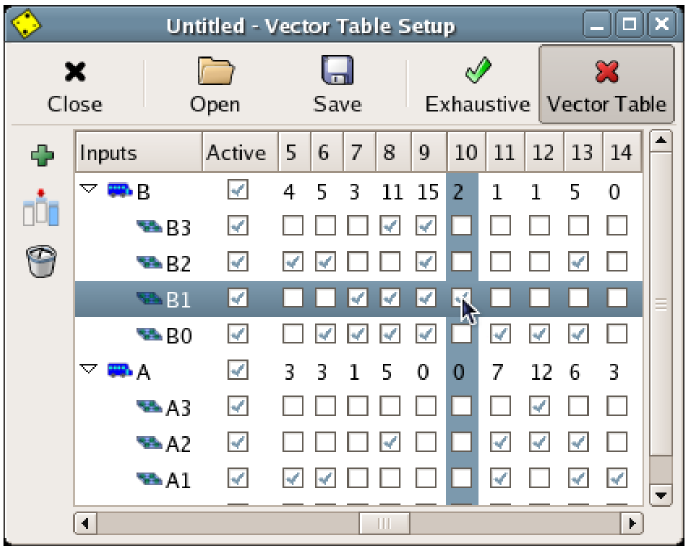
- Grouping the input/output signals in buses, to simplify signal name handling, simulation input vectors definition, and simulation results inspection (see Figure 2.27).
- Importing and exporting layout blocks to files, which allow, for example, to easily import QCA layout blocks produced by the developed QCA-LG tool.

The simulation can be performed with an exhaustive set of input vectors, or alternatively with a user-defined set of input vectors (see Figure 2.28). There are two integrated simulation engines available with QCADesigner: the Coherence Vector Simulation Engine, which is slower, but provides more accurate results, than the Bistable Simulation Engine. Simulation results are presented as waveforms, optionally grouped in buses (see Figure 2.29)



Bus Name: Input B

Figure 2.27: Grouping the QCA layout signals using buses in QCADesigner.



Inputs	Active	5	6	7	8	9	10	11	12	13	14
B	✓	4	5	3	11	15	2	1	1	5	0
B3	✓	☐	☐	☐	✓	✓	☐	☐	☐	☐	☐
B2	✓	✓	✓	☐	☐	✓	☐	☐	☐	✓	☐
B1	✓	☐	☐	✓	✓	✓	✓	☐	☐	☐	☐
B0	✓	☐	✓	✓	✓	✓	☐	✓	✓	✓	☐
A	✓	3	3	1	5	0	0	7	12	6	3
A3	✓	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐
A2	✓	☐	☐	☐	✓	☐	☐	✓	✓	✓	☐
A1	✓	✓	✓	☐	☐	☐	☐	✓	☐	✓	✓

Figure 2.28: QCADesigner simulation inputs windows, with the option to specify a set of user-defined input vectors, or alternatively choose an exhaustive simulation.

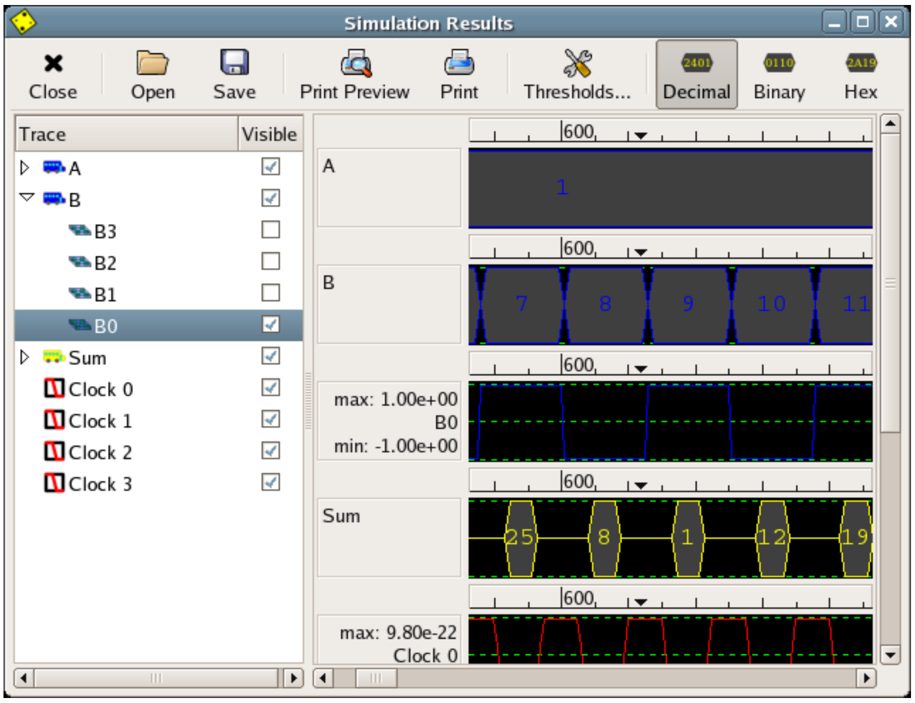


Figure 2.29: QCADesigner simulation results window.

3

QCA-LG: a tool for automatic generation of QCA layouts

Contents

The main goal of the work produced in this thesis is to automatically generate a quantum dot layout for a given combinational circuit, in a format compatible with the QCADesigner tool, and not to develop a professional “Place and Route” tool. The flow of the program actions is shown in Figure 3.1.

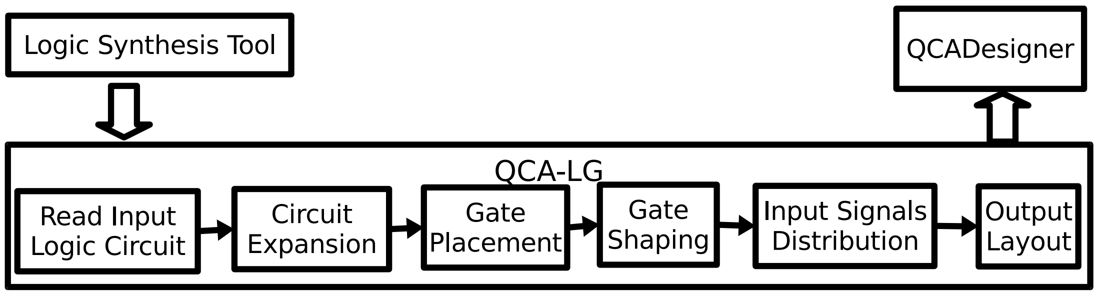


Figure 3.1: Block diagram of the operations carried out by the QCA-LG tool.

The QCA-LG tool aims to be integrated in the QCA design flow as the bridge from the logic description of a circuit to the its physical layout. The place of QCA-LG in the design flow of QCA technology is presented in Figure 3.2. The tool QCADesigner has the functionalities of a QCA

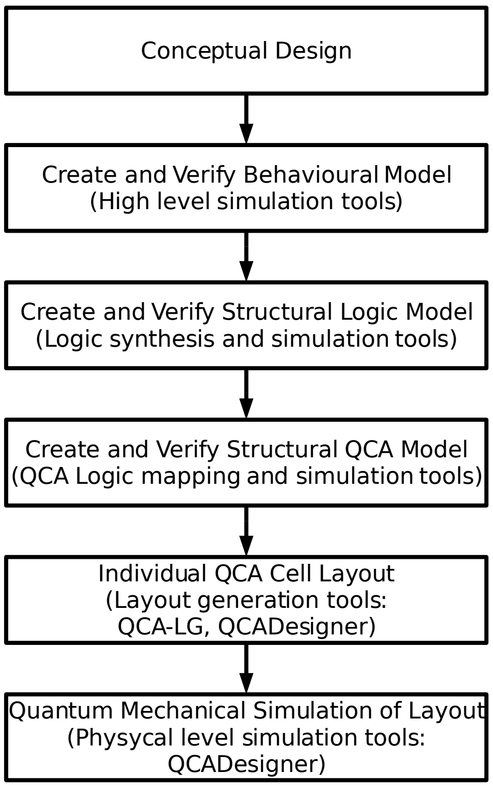

Conceptual Design
Create and Verify Behavioural Model (High level simulation tools)
Create and Verify Structural Logic Model (Logic synthesis and simulation tools)
Create and Verify Structural QCA Model (QCA Logic mapping and simulation tools)
Individual QCA Cell Layout (Layout generation tools: QCA-LG, QCADesigner)
Quantum Mechanical Simulation of Layout (Physycal level simulation tools: QCADesigner)

Figure 3.2: The place of QCA-LG tool within the QCA technology design flow.

layout editor and a QCA layout simulator, but the layout must be all drawn by hand. Therefore, there was the need for an automatic layout generation tool for QCA circuits.

3.1 Read Input Logic Circuit

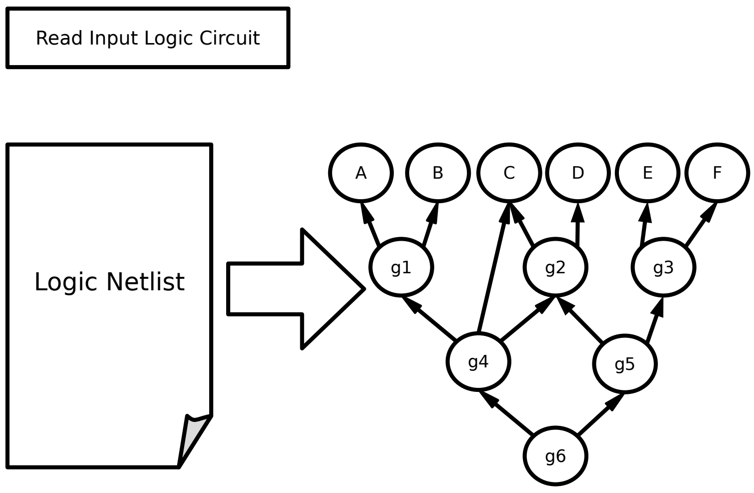


Figure 3.3: Representation of the Input Reading operation.

Importing the combinational circuit is the first stage of the procedure. Two formats are actually supported by the tool:

- LSI (Synopys tools can write in this format.)
- Gate (This is the logic netlist format used by MVSIS, SIS[23] successor)

A Lex&Yacc parser was specified for each of the two formats. The use of Lex&Yacc makes easier to support new input file formats. To make possible the compilation of a given design to a logic netlist formed only by the supported gates, technology libraries were specified containing only Majority, AND, OR, NAND, NOR and Inverter gates. Libraries with only the six supported components were specified, both for the *Synopsys* and *MVSIS* tools, in order to generate netlists based only on those components: majority gates, NOT gates, and 2-inputs AND, OR, NAND and NOR gates.

The circuits are internally represented by the tool as directed graphs and stored in a hash table, where each object represents a gate of the circuit or a primary input.

3.2 Circuit Expansion

The first operation over the circuit is its expansion, where any shared node is duplicated. At the end of this step the fanout of every gate is only one. This operation makes the place and route

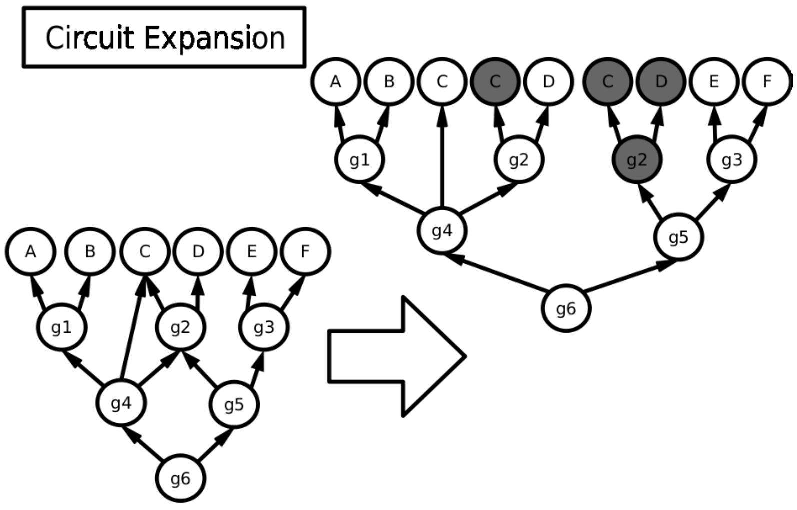


Figure 3.4: Representation of the Circuit Expansion operation (the grey nodes, gates, are copies).

task easier, avoiding wire crossing at the expense of an increase in the circuit area.

The logic replication is not supposed to increase the circuit delay, but it can happen because the extra area implies extra length in some wires, and even sometimes this forces wires to be split up on some more clock zones. The method applied for replication consists on performing breadth-first exploration of the circuit, starting from each primary output towards the inputs, marking every node as visited if it was not visited yet. Every time a node is revisited it is duplicated, as well as all the nodes included in the sub tree rooted at it. The duplication is a recursive process that may "explode" if there are loops in the circuit, however since there are no loops in combinational logic circuits this should not be an issue.

After this operation the circuit is composed by independent sub circuits. Each one of these sub circuits is rooted at a primary output of the circuit and will be treated separately in the Gate Placement step.

3.3 Gate Placement

For mapping the circuit from a gate level to physical level, three different referentials are considered:

- gate level coordinates - used to define relative positions of the gates; each unit in the vertical direction represents one gate level (vertical distance between the inputs and the output); in the horizontal direction the unit represents the minimum width of one wire (three cells, so that two adjacent wires have a two cell separation);
- cell level coordinates - in this referential the unit represents one cell dimension in both directions;

- physical coordinates - correspond to the mapping of the cell coordinates to the system of coordinates used in the QCADesigner. In this case the unit is nanometer, and each cell has 18.0 units of width and length; the considered spacing between each cell is 2.0 units in both directions.

The determination of the coordinates where each gate (of the present sub circuit) will be placed is done in three phases, and may be adjusted after; the coordinates resulting from this operation are in the gate level referential. In the first phase, the level in the graph is determined for each gate, by using the breadth-first approach, by starting from each primary output, defining zero as the root's level and proceeding towards the inputs. When visiting each gate, is assigned to its inputs the present gate's level plus one. The graph level is the y coordinate in gate level coordinates, and the maximum level found indicates the maximum y coordinate, called y_{max}. The value of y_{max} allows to give a rough estimate of the circuit's dimensions, and once the graph is a subset of a complete ternary tree, it also indicates that the maximum number of nodes in any given level is $3^{y_{max}}$. The second phase is to assign different numbers to the gates within each level. This is done again with breadth-first exploration, and the numbers range from 0 to $3^{y_{max}} - 1$. When visiting a given gate its inputs receive a number determined as follows:

- Left input $n = 3 * n_{gate}$
- Central input $n = 3 * n_{gate} + 1$
- Right input $n = 3 * n_{gate} + 2$

Note that only for Majority Gates there is a Central input, otherwise, for two-input gates only the left and right inputs are considered. In the third phase, the x coordinate of the gate level coordinates is calculated according to the expression:

$$x = (1 + 2n)\frac{(3^{y_{max}-y} + 1)}{2} - n, \tag{3.1}$$

Equation 3.1 was obtained by the visual inspection of the coordinates of the nodes of a complete ternary tree.

After these three steps the gates in the present sub circuit are arranged as an incomplete ternary tree, in which some nodes may lack the middle child. This is the case of all non Majority gates, and it leads to an undesired waste of area.

It is important to notice that all Inverter gates are ignored in the this step, because they are processed in a separate and additional phase.

3.4 Gate Shaping

Right after the place is defined in the gate level coordinates, the gate's dimensions are calculated.

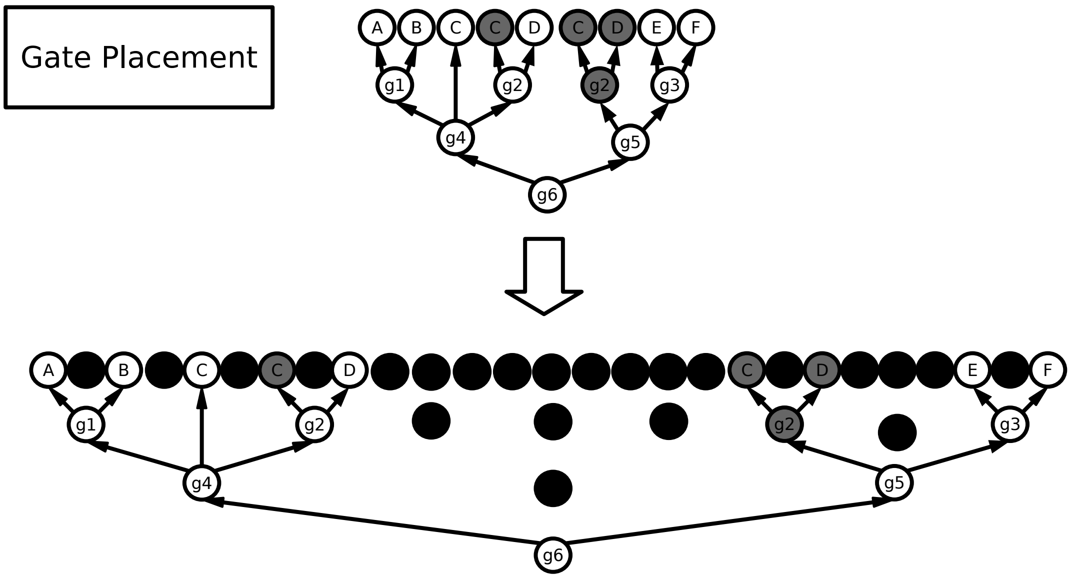


Figure 3.5: Representation of the Gate Placement operation. Black spots show wasted area.

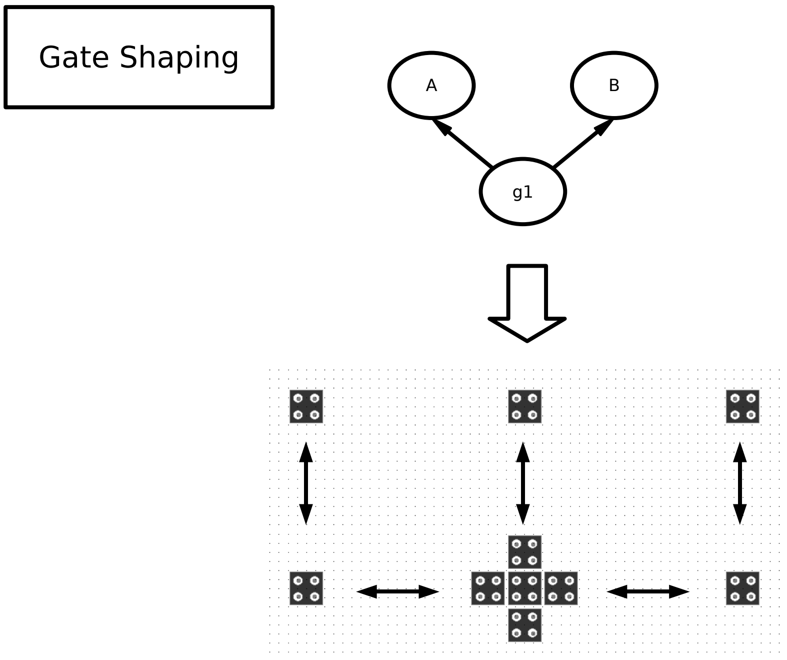


Figure 3.6: Representation of the Gate Shaping operation.

At this point it also takes place the mapping from the gate level referential to the cell level referential. In a general way this is ruled by the expressions:

$$x_{cell} = (x_{gate} + x_{gate}offset)x_{factor}, \tag{3.2}$$

$$y_{cell} = (y_{gate})y_{factor} + y_{cell}offset, \tag{3.3}$$

where $x_{gate}offset$ is used to separate the various sub circuits along the layout, and $y_{cell}offset$ is used to adjust the gate position when needed, as the input wire may have to be split into more than one clock zone, as it will be explained later.

Inverter gates will be place in wires. NAND and NOR gates will be treated as AND and OR gates, respectively, where an additional Inverter gate is considered at the gate's output.

3.4.1 Integrated routing

The input wires of each gate connect directly to the output cells of the gates at its inputs. This way the routing is done implicitly and as a part of gate representation. Input wire length is calculated from both the coordinates of each gate and the coordinates of its inputs. If the wire length exceeds the maximum clock zone length [24], it will be split into more than one clock zone.

For a two or three input gate, if the longest wire has to be split into more than one clock zone then the shortest wire has also to be split into the same number of clock zones to maintain the input arrival timing; this usually means the shortest wire has to become longer to respect the minimum clock zone length. Once the horizontal alignment of the first cell of all input wires must be ensured, when one wire gets longer then the others must also get longer in the vertical direction. When the increase in wire length occurs, it is necessary to reassure that the number of cells per clock zone does no exceed the maximum clock zone length, iteratively repeating the last steps if needed.

A half QCA cell replaces a full QCA cell when a given input wire is connected to an inverting gate, which means an Inverter, NAND or NOR gate; and this replacement implies some extra space between cells to keep the original wire length. Signal inversion can be done this way because the half cell is itself an Inverter (see Figure 2.17). This feature can not be easily achieved when drawing the layout by hand using QCADesigner, once "snap to grid" option has to be turned off, and the half QCA cell isn't available as a design object.

In fact, any rectangular shaped object with an arbitrary number of quantum dots can be correctly imported and simulated in QCADesigner, if specified in a text file according to the supported (XML-like) format.

3.4.2 Circuit synchronization

In order to ensure the proper operation of the circuit it is necessary to set the clock zone of each gate's output cell as the preceding clock zone of the input cell of the next gate towards the primary outputs. To achieve this, the clock zone of the Majority Gate is set arbitrarily and then the clock zones of the input wires are set accordingly, so the signals may flow from the inputs (earliest clock zone), along the input wire (ascending clock zones), to the output (latest clock zone). The clock zone in which the input wires start is then saved in the gate's record for future use.

As the circuit is drawn from the outputs towards the inputs, except for the primary outputs, every gate is drawn after the gate they feed. So the clock zone of the output cell of each gate is known as soon as its direct dependant gate is drawn. A given gate is considered to be dependant of other if it uses the output of this other as an input.

3.5 Input Signals Distribution

As one given primary input signal can be used as input for more than one logic gate, that signal must be distributed from one unique source to all the places where it is needed. The expansion of the circuit generates copies of some logic gates, which means some input signals must be delivered to some extra points.

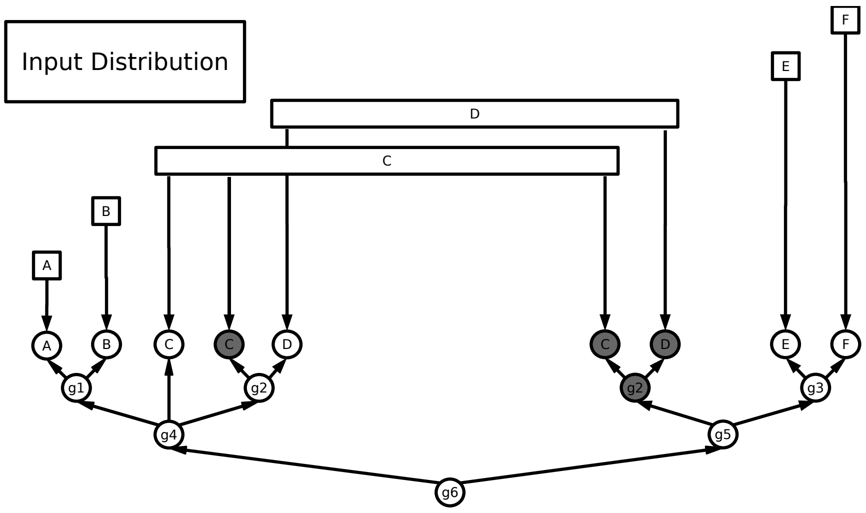


Figure 3.7: Representation of the Input Signals Distribution operation.

An input signal also must arrive at its destination(s) synchronized with other signals. Therefore it is necessary to take care of this problem. The signal's destinations are sorted by (descending) priority, considering that the primary input of the circuit will be place above the leftmost destination and as low as possible. A main distribution horizontal wire will be rooted at the primary input, and follows to the right. All the distribution wires are placed in a different layer than the logic gates.

Note that in this case we are dealing with cell level coordinates. The expression used to evaluate the "urgency" to arrive at a given destination(x,y) at a given moment (clk) is:

$$pri(x, y, clk) = (x - x_{min}) + (y - y_{max}) - (clk - clk_{min}) * Zone, \tag{3.4}$$

where x_{min} is the minimum horizontal coordinate value found among all destinations of the signal (left most), y_{max} is the maximum vertical coordinate (lower, as y values grow downwards) and clk_{min} is the minimum earliest zone; $Zone$ is the maximum length allowed for a clock zone. The destination is considered to have higher priority as the distance to the primary input grows and the number of clock zones to transverse diminishes.

Starting by the most restrictive, an iterative method is used to determine where the main distribution wire will be placed, and where the signal will derive to secondary distribution wires (leading to the logic gate where it is needed). On each iteration, the borders of the clock zones in the main distribution wire are updated according to the most restrictive destination. And then the sorted set of destinations is scanned and, given the present position of the main distribution wire, it is checked if it is possible to respect the maximum and minimum length allowed for a clock zone. If the signal can travel correctly to every destination, the main distribution wire is at its final position. Otherwise, the main distribution wire is pushed up (this is done with care to avoid collisions with other primary inputs' main distribution wires) and a new iteration takes place.

The initial conditions are critical to ensure this iterative method stops and they depend heavily on the most restrictive destination. So the sorting is a fundamental step.

3.5.1 Delay equalization

In order to ensure the correct timing of all input signals, some may have to be intentionally delayed to arrive at the logic gates at the same time that most delayed signals do, and thus the outputs will all be generated at the same exact time. To accomplish this objective, after the distribution wires are put in place some extra clock zones may have to added to the beginning of a wire, just before the signal is received from the primary input.

Additionally, to facilitate the interpretation of the simulation results, the clock zone of all the inputs and the outputs is the same. This way, the input-output delay of the circuit is always known and is an integer number of clock periods.

3.6 Output Layout

The output format aims to be compatible with QCADesigner (version 2.0.3) [7]. The produced layout is stored as a QCA layout block, and can be imported into any QCADesigner layout. The tags of the nodes in the original netlist appear as labels of the output cell of the corresponding logic gates and inputs, so that manual editing is possible.

4 Implementation and Experimental Results

Contents

4.1 Implementation

The implementation of the QCA-LG software tool was performed in a UNIX-like environment (Linux) using standard open-source tools. The programming language chosen was C, and Lex & Yacc were used to specify the parsers for the supported netlist formats. The debug was carried out using GDB, and Valgrind was used to validate memory usage.

4.1.1 Building the graph

The circuit is imported through the appropriate parser and stored, gate by gate, in an identifier table implemented as a hash table. Simultaneously, the graph is built by connecting the records in data structures, which represent logic gates, with pointers from a given gate to its inputs. Therefore, it is possible to reference a logic gate by its name, or through a dependency tree. Every primary output has its own dependency tree, which may have shared subtree with other output's dependency tree, when it was built, but no longer after the Circuit Expansion operation.

4.1.2 Graph transversing

The graph search methods approach followed to transverse the circuit's graph representation were Depth First Search (DFS) and Breadth First Search (BFS). The DFS is used in two situations; first, in the Circuit Expansion (see in Section 3.2), when DFS is used in conjunction with BFS, and second to determine the level of every node in the graph, that is the distance between a given node and the output it is "appended" to.

In the Circuit Expansion operation the graph is transversed in BFS and every time a node is revisited the subtree rooted at that given node is duplicated with a recursive approach that implements DFS. For identifying the level, the transversing of the graph is implemented also with a recursive procedure. In both these cases, the First In Last Out (FILO) queue implicitly used was the function call stack during the recursive function calls.

On the other hand, BFS is extensively applied throughout the program flow to transverse the graph. The floor-planing of the circuit's graph is a good example of a BFS usage. BFS was implemented with the help of a First In First Out (FIFO) queue to store the exploring border. This method was preferred over the DFS, when this choice was possible, but in some cases the DFS has to be used to ensure the proper result of the performed operations.

4.1.3 Input and Output

The input and output operations, for reading the input netlist file and writing the output layout description file, respectively, are implemented through the standard input and standard output file descriptors for simplicity Therefore, the input and output must be "piped" in and out on the command line when the program is called to be executed. Some debug code, that can be optionally activated at compile time, writes to standard error file descriptor, so there is no mixing between the program output and the possible error/debug messages.

4.2 How to use the tool

Although the tool's internal operation has been discussed in Chapter 3, here the focus is mainly in utilization and applicability. The relevant files mentioned here are included in Appendix B to improve the readability.

4.2.1 Getting a netlist

First, a netlist for the circuit must be built, either in Gate or LSI format. Considering the example of a VHDL file in Appendix B.1, as a starting point, the LSI netlist can be obtained with Synopsys tools (see Appendix B.2). The VHDL file must be compiled using a custom logic library (see Appendix B.3). After the design is compiled, a schematic view can be observed, as in Figure B.1.

An alternative to a VHDL/Verilog description is the usage of a Berkley Logic Interchange Format (BLIF) file, such as the one included in Appendix B.4. A simple script (see Appendix B.6) can be used to obtain the corresponding logic netlist (see Appendix B.5) in the Gate format, through MVSIS. Additionally, the library files in Appendix B.7 and Appendix B.8 must be supplied in order to build a compliant netlist.

4.2.2 Execute the QCA-LG tool

Once the netlist is ready, the QCA-LG tool can be used in command line, feeding the input file as stdin. The output layout will be dumped to stdout, and must be redirected to a file.

The QCA-LG tool takes, as an optional parameter, the format of the input netlist: *i)* if the format is LSI, no parameter has to be given (default format); *ii)* otherwise, the "gate" option must be passed to indicate that the netlist is in the MVSIS Gate format. Examples of command lines are:

- ./qca-lg < example.lsi > example.qca
- ./qca-lg gate < simple2.gate > simple2.qca

4.2.3 View and simulate the layout

The produced QCA files are composed by layout blocks, and can be directly imported by the QCADesigner (menu Tools -> Import Block. . .). For the LSI format input netlist the "example.qca", presented in Figure 4.1, is obtained. For the Gate format input netlist, "simple2.qca" was generated, and it is presented in Figure 4.2. These input netlist can be found in Appendix B.

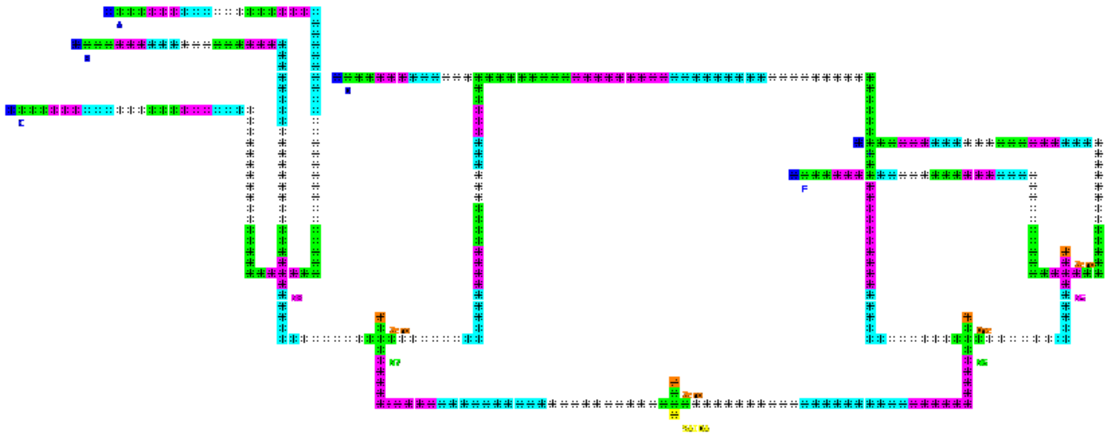


Figure 4.1: The layout representation of an example. (File: example)

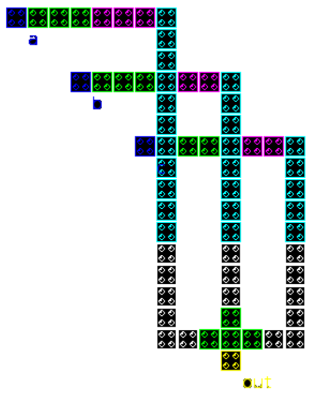


Figure 4.2: The layout representation of an example. (File: simple2)

In conclusion, this QCA-LG tool can be used in the design flow of QCA systems after the logic synthesis, and prior to physical simulation. It takes as input the logic circuit resulting from the former and produces the layout to be given as input to the latter.

4.3 Manually elaborated and automatically generated layouts

In this section, obtained layouts with the QCA-LG tool are presented. The first example is a multiplexer, and a handmade version of the same circuit is also presented for comparison purposes. A few more automatically generated layouts are discussed here. However, since the other examples lead to the same conclusions, the figures with the layouts are not presented here but in Appendix A.

The manually designed version of the multiplexer circuit is presented in Figure 4.3. The respective simulation results are shown in Figure 4.7, where one clock period delay exists between the inputs and the corresponding output.

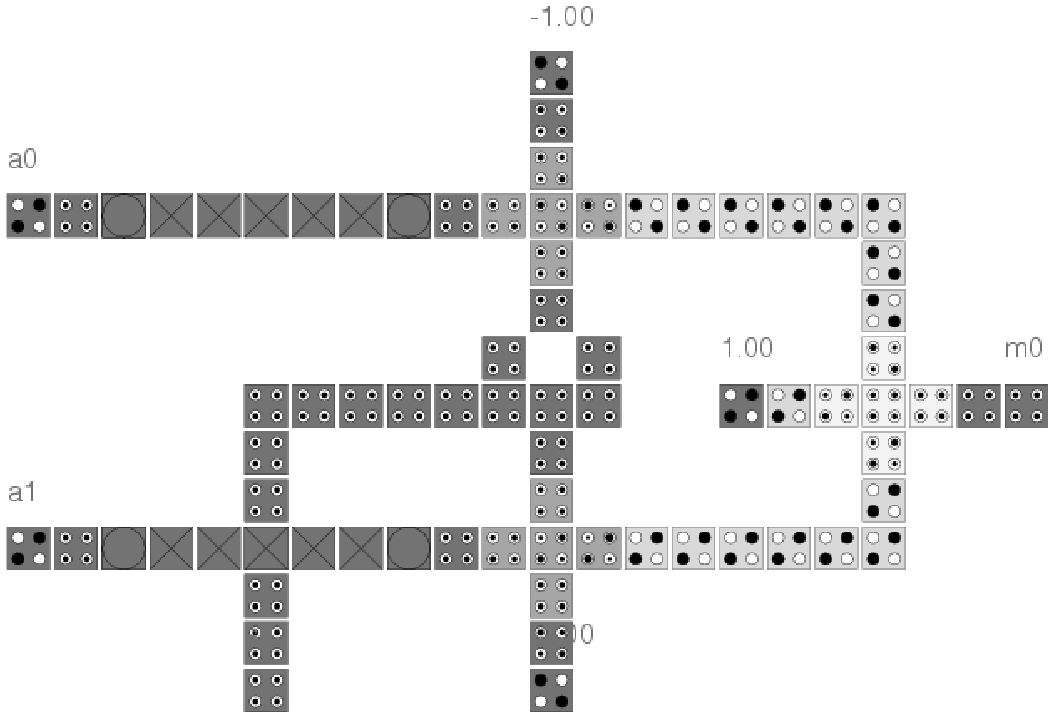


Figure 4.3: Layout of a 2 to 1 multiplexer designed manually.

The layouts generated by the QCA-LG tool don't have special styles for vias and crossovers cells, however these kind of cells are present. The top horizontal wires, from which the input signals are distributed, are in the *crossover* layer, and the points of vertical contact between the *Main Cell Layer* layer and the *crossover* layer have via cells in an intermediate layer specially used to make vertical connections.

The automatically generated layout for the 2 to 1 multiplexer is presented in Figure 4.4, and the wave forms resulting from the simulation can be observed in Figure 4.8. This layout introduces a delay of two full clock periods.

In Figure 4.5, a 1 bit full-adder automatically generated layout can be observed. The total delay from the inputs to the output of the circuit is four full clock cycles. It can be observed large clock zones in the branches where the signal flows faster (right most vertical wires), traveling through more (10 at most) cells per clock cycle, and small clock zones where the signal flows slower (left most vertical wires), traveling through less (3 at least) clock zones per clock cycle. Therefore, although the number of clock zones transversed is the same, the physical distance traveled by the signal is different.

The layout generated for the *simple4* benchmark circuit is presented in Figure 4.6. This is an example of a medium sized circuit that clearly shows the cost of synchronisation, in terms of area

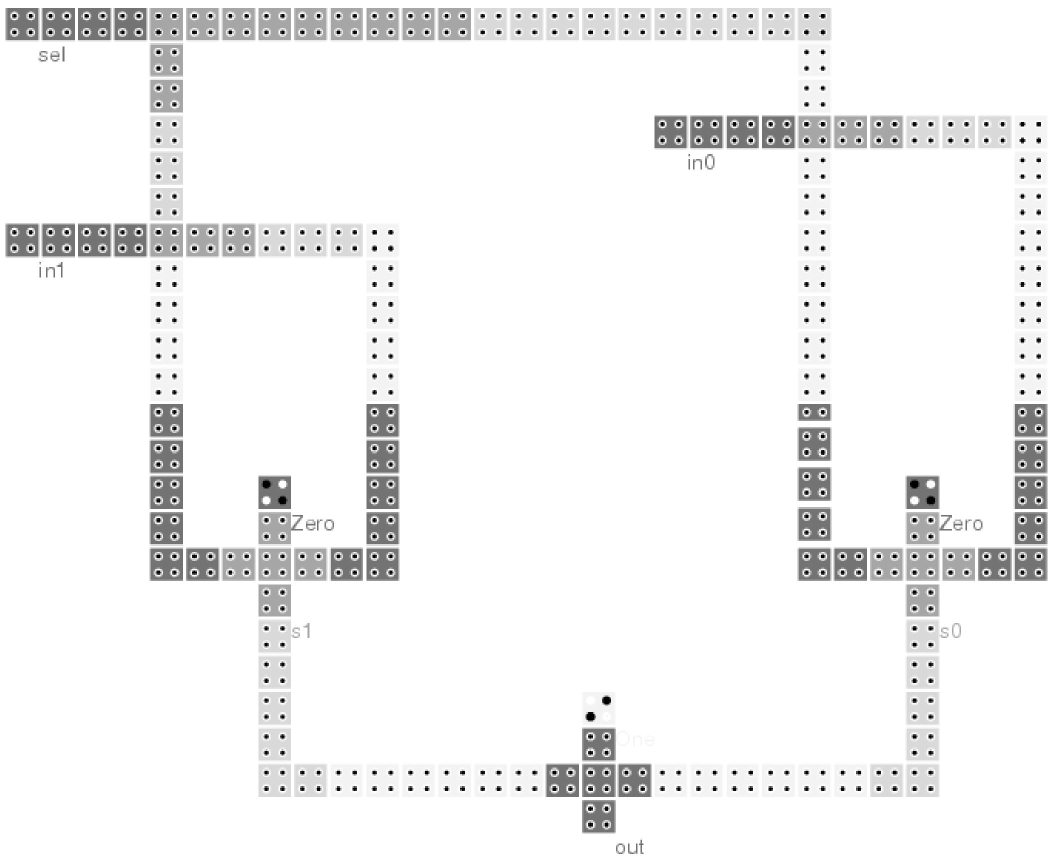


Figure 4.4: Layout of a 2 to 1 multiplexer generated by the tool.

needed, to make the same signal arrive at the right place in the right time.

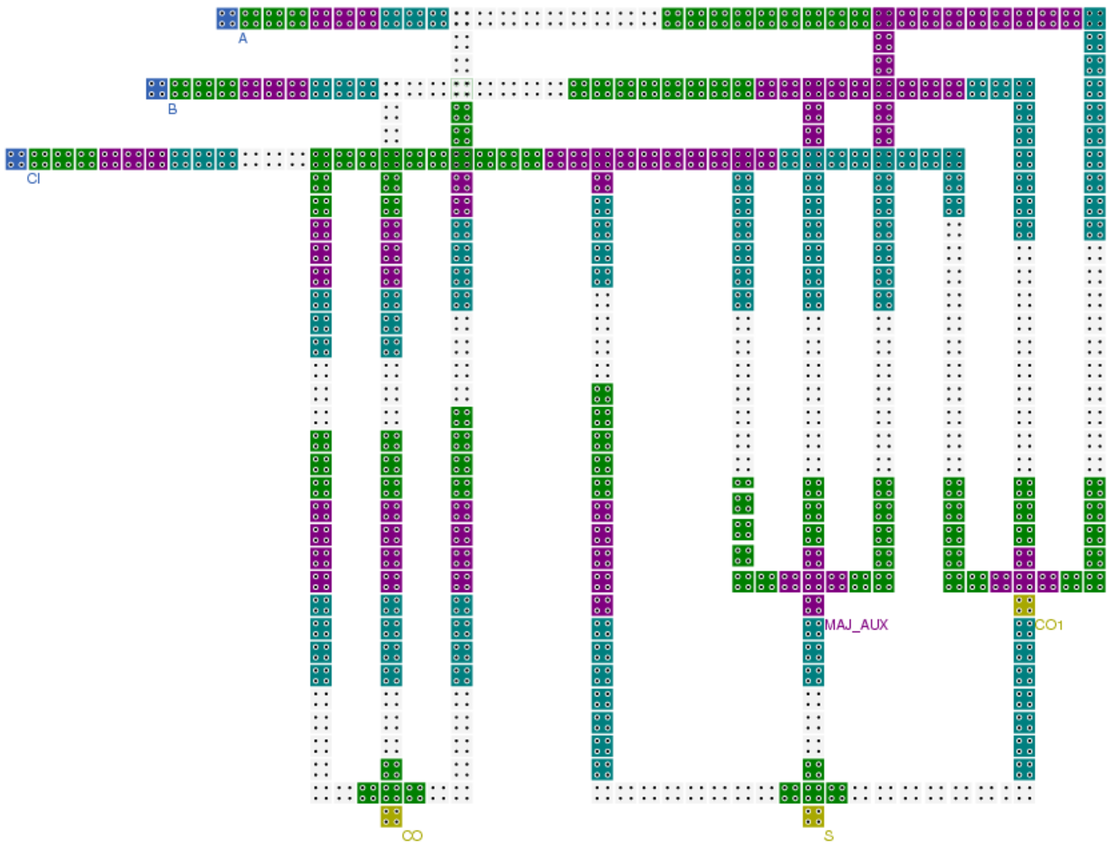


Figure 4.5: The layout representation of a 1 bit full-adder. (File: adder4)

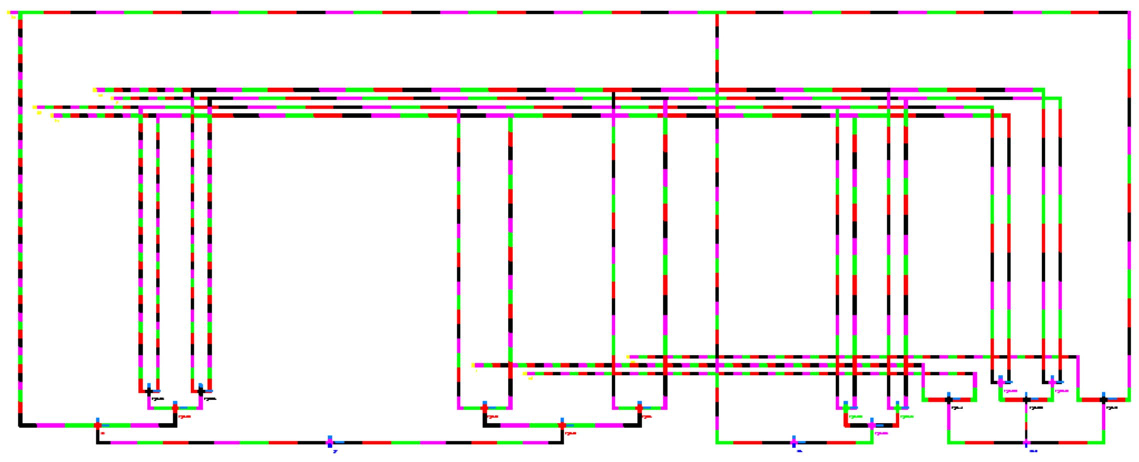


Figure 4.6: The layout representation of an example. (File: simple4)

4.4 Simulation results

Here the exhaustive simulation results obtained with QCADesigner are presented for both the handmade and automatically generated multiplexer layouts presented in Section 4.3 and Section 4.3.

The results are equivalent, but the handmade multiplexer has better performance than the automatically generated multiplexer. The input-output delay of the handmade circuit is only one clock period, and for the automatically generated version is two clock periods.

The simulation results for the 1 bit full-adder layout generated by the tool, presented in Figure 4.5, are shown in Figure 4.9. The input signal are the two operands A and B, and the carry in, CI; the outputs are the sum bit, S, and the carry out, CO. The truth table of the full adder is surrounded by a dashed line, thus it can be clearly seen the correct behaviour of the circuit, and also that the input-output delay introduced is three clock periods.

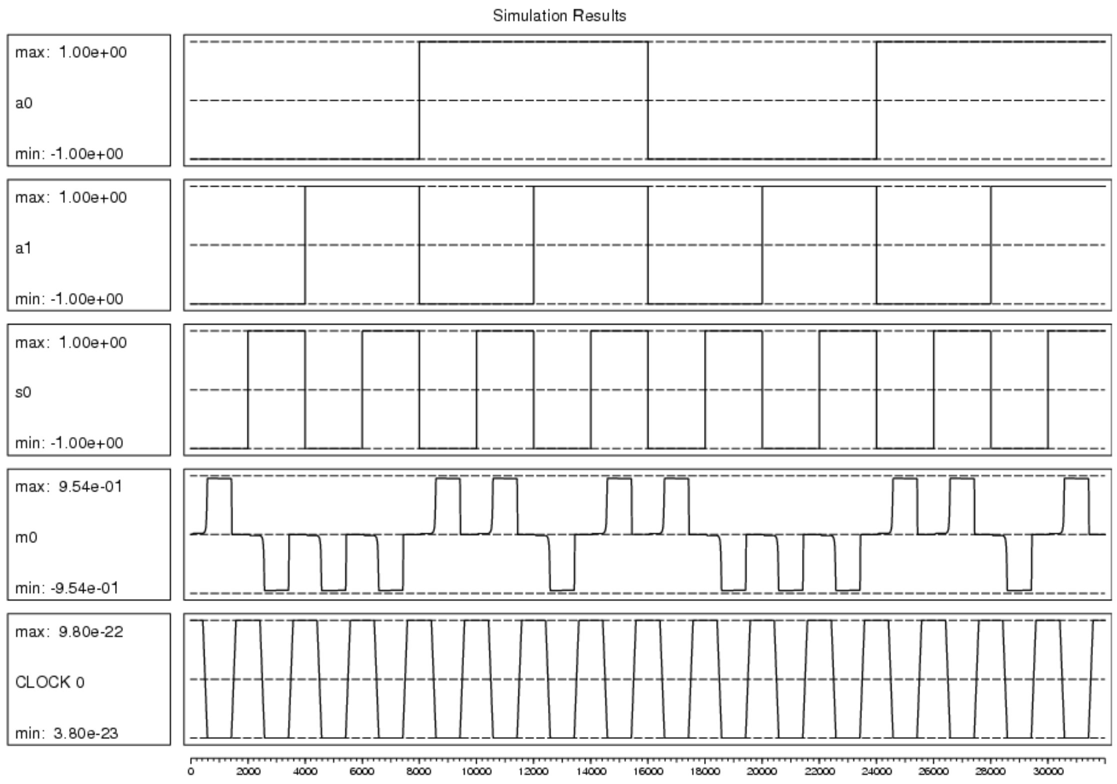


Figure 4.7: Simulation results of the handmade 2 to 1 multiplexer layout.

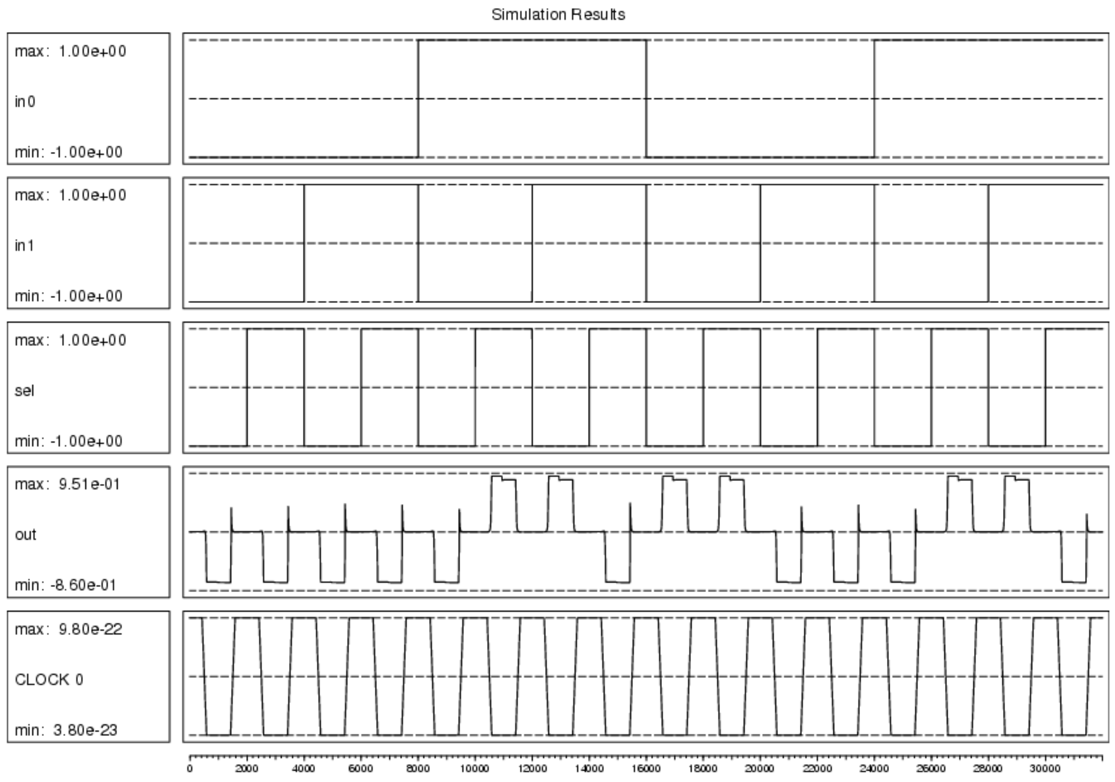


Figure 4.8: Simulation results of the 2 to 1 multiplexer layout generated by the tool.

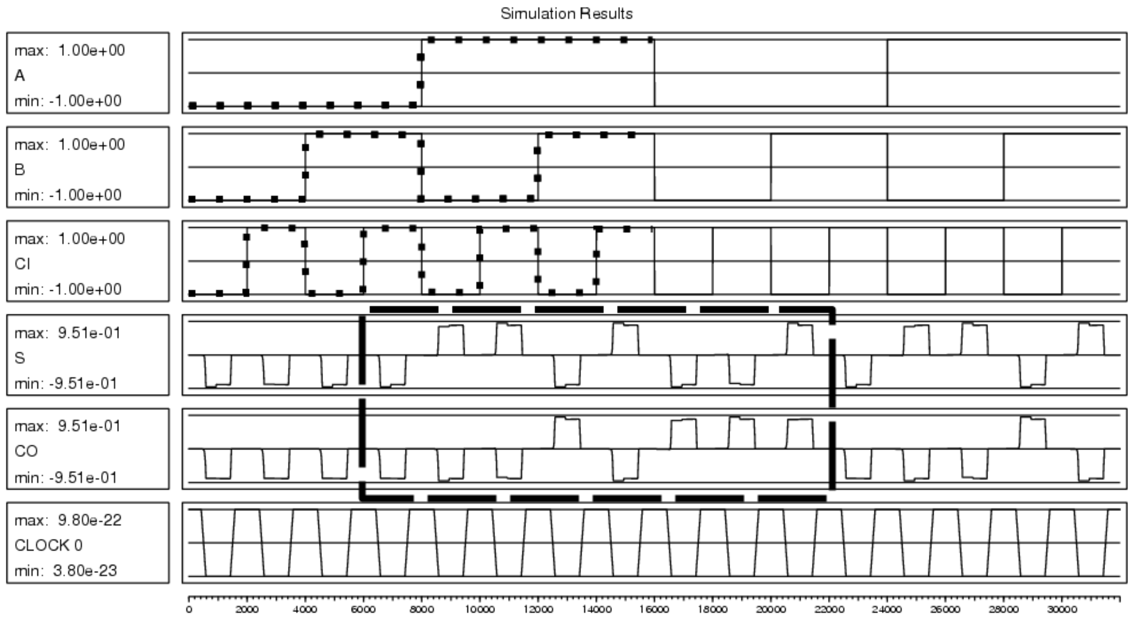


Figure 4.9: Simulation results of the 1 bit full-adder layout generated by the tool. The resulting waveforms for S and CO bit enclosed by a dashed line correspond to the full-adder function for the input waveforms A, B and CI marked by a dotted line.

5 Conclusions

Contents

The technology studied in this project, QCA, reveals to be a strong competitor, along with SET, to in a near future, complement CMOS technology in digital integrated circuits. It must be remembered that analog CMOS technology will be needed, at least, to bound the real analog world to QCA quantum-dots.

This QCA technology seams particularly suitable for high throughput and deeply pipelined architectures, given the inherent pipelined operation of a single QCA wire, acting as a chain of latches. Applications such as audio and video stream processing might benefit much with QCA architectures. On the other hand, heavily conditional processing would be penalized, given the high cost of a stall in an extremely deep pipeline.

Regarding the design flow of QCA circuits, from system specifications to physical fabrication, many points have to be tuned to reach a viable alternative for CMOS in the digital domain. Logic synthesis is a key operation that may drastically reduce the area and delay of the circuits. Once the professional tools currently available are deeply oriented to the most basic logic gates feasible in CMOS (usually NAND gates), this will be an area of great interest. The greatest challenge is, perhaps, to adapt current tools and design flows from current CMOS processes in order to accommodate the special features of QCA, such as gate level synchronization and in wire memory.

There may be need to adapt many existing CMOS circuits to QCA, and this may result in the exclusive use of And, Or and Inverter gates in QCA circuits. Although this may be the solution for small sized circuits, it can become very ineffective for larger ones.

5.1 The developed QCA-LG software tool

The QCA-LG tool proposed and developed in the scope of this thesis can be used to produce suitable QCA layouts for combinational circuits. The QCA-LG accepts standard netlist formats, such as *LSI* and *Gate*, and generates QCA layouts that can be physically simulated by the well known QCADesigner tool.

The presented results show that the developed QCA-LG tool is able to automatically generate layouts for small sized circuits. However, for medium and large sized circuits the wasted area can be significant, namely for circuits not exclusively composed by majority gates.

More research has to be performed in order to optimize the gates placement, which is statically performed and can be one of the main sources of inefficiency. With QCA-LG the design flow for QCA technology is now almost complete. Combinational VHLD/Verilog circuits can be mapped into logic netlists with existing synthesis tools. These netlists can be transformed into QCADesigner compatible layout using QCA-LG, and validated by physical simulation.

An evaluation version of the QCA-LG tool is available for download at *http://web.ist.utl.pt/ttt/qca-lg/*.

5.2 Future work

As it was observed on the results provided in section 4.3, while for small sized circuits the resulting layout obtained with the QCA-LG is acceptable, for medium or large sized circuits the wasted area is significant. The optimization effort should be focused on the gate's placement; as this operation is performed in a static way, it can be very inefficient.

Additionally here is a list of the current identified problems or missing features in the developed QCA-LG software tool:

- Inverting gates are turned into non-inverting when they produce primary outputs, as the signal are not inverted when they don't feed any other logic gate.
- The via cells are not placed yet, so they have to be added manually to get the correct simulation results in QCADesigner.
- When a primary input is needed only in one place, some wire length can be saved. This kind of waste can be seen in "simple2.qca" example.

These issues and the need for optimized layouts leave room for further development of the QCA-LG tool developed in this thesis.

Bibliography

[1] C.S. Lent, P.D. Tougaw, W. Porod, and G.H. Bernstein. Quantum cellular automata. Nanotechnology, 4:49–57, 1993.

[2] University of Notre Dame. Department of electrical engineering. web: http://www.nd.edu/ qcahome/.

[3] Alexei O. Orlov, Islamshah Amlani, Ravi K. Kummamuru, Rajagopal Ramasubramaniam, Geza Toth, Craig S. Lent, Gary H. Bernstein, Gregory L. Snider, Wolfgang Porod, and James L. Merz. Quantum-dot cellular automata: Introduction and experimental overview. IEEE NANO, pages 465–470, October 2001.

[4] G. Bazan, A. O. Orlov, G. L. Snider, and G. H. Bernstein. Charge detector realization for AlGaAs/GaAs quantum-dot cellular automata. Vacuum Science Technology B, 14(6):4046–4050, Nov./Dec. 1996.

[5] Alexandra Imre, Lili Ji, Gyorgy Csaba, Alexei Orlov, Gary H. Bernstein, and Wolfgang Porod. Magnetic logic devices based on field-coupled nanomagnets. IEEE Semiconductor Device Research Symposium, pages 25–25, December 2005.

[6] Yuhui Lu, Mo Liu, and Craig Lent. Molecular electronicsfrom structure to circuit dynamics. IEEE NANO, 1:62–65, June 2006.

[7] K. Walus, V. Dimitrov, G.A. Jullien, and W.C. Miller. QCADesigner: A CAD tool for an emerging nano-technology. Micronet Annual Workshop, 2003. web: http://www.qcadesigner.ca.

[8] Steven C. Henderson, Eric W. Johnson, Jason R. Janulis, and P. Douglas Tougaw. Incorporating standard CMOS design process methodologies into the QCA logic design process. IEEE Transactions on nanotechnology, 3(1):2–9, March 2004.

[9] Rui Zhang, Pallav Gupta, and Niraj K. Jha. Synthesis of majority and minority networks and its applications to QCA, TPL and SET based nanotechnologies. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pages 229–234, 2005.

[10] Zhi Huo, Qishan Zhang, Sansiri Haruehanroengra, and Wei Wang. Logic optimization for majority gate-based nanoelectronic circuits. Circuits and Systems, page 4 pp., May 2006.

[11] Synopsys Inc. Synopsys tools, May 2003. Version 2003.06 for linux.

[12] Donald Chai, Jie-Hong Jiang, Yunjian Jiang, Yinghua Li, Alan Mishchenko, and Robert Brayton. MVSIS 2.0 Users Manual. Department of Electrical Engineering and Computer Sciences.

[13] R. van de Haar, R.H. Klunder, and J. Hoekstra. SPICE model for the single electron tunnel junction. Electronics, Circuits and Systems, 2001. ICECS 2001. The 8th IEEE International Conference on, 3:1445–1448, 2001.

[14] Rui Tang, Fengming Zhang, and Yong-Bin Kim. Quantum-dot cellular automata spice macro model. In GLSVSLI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI, pages 108–111, New York, NY, USA, 2005. ACM Press.

[15] Norchip, November 2007 Aalborg, Denmark. web: http://www.norchip.org/.

[16] T. Teodósio and L. Sousa. QCA-LG: A tool for the automatic layout generation of qca combinational circuits. In Norchip, 2007.

[17] Konrad Walus and Graham A. Julien. Design tools for an emerging SoC technology: Quantum-dot cellular automata. In Proceedings of the IEEE, volume 96, pages 1225–1244, June 2006.

[18] C.S. Lent and P.D. Tougaw. Dynamic behavior of quantum cellular automata. J. Appl. Phys., 80(8):4722–4736, October 1996.

[19] V. Vankamamidi, Marco Ottavi, and Fabrizio Lombardi. Clocking and cell placement for qca. IEEE-NANO, 1(17-20):343–346, June 2006.

[20] Alexei O. Orlov, Islamshah Amlani, Ravi K. Kummamuru, Rajagopal Ramasubramaniam, Geza Toth, Craig S. Lent, Gary H. Bernstein, and Gregory L. Snider. Experimental demonstration of clocked single-electron switching in quantum-dot cellular automata. Applied Physics Letters, 77(2):295–297, July 2000.

[21] Alexei O. Orlov, Islamshah Amlani, Geza Toth, Craig S. Lent, Gary H. Bernstein, and Gregory L. Snider. Experimental demonstration of a binary wire for quantum-dot cellular automata. Applied Physics Letters, 74(19):2875–2877, May 1999.

[22] Geza Toth and Craig S. Lent. Quasiadiabatic switching for metal-island quantum-dot cellular automata. Applied Physics Letters, 85(5):2977–2984, March 1999.

[23] Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R. Stephan, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Department of Electrical Engineering and Computer Sciences, Memorandum No. UCB/ERL M92/41 edition, May 1992.

[24] Kyosun Kim, Kaijie Wu, and Ramesh Karri. Towards designing robust qca architectures in the presence of sneak noise paths. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, volume 2, pages 1214–1219. IEEE Computer Society, 2005.

A

Appendix A

Contents

A.1 Layout images

In this appendix, several layout images are presented. The images correspond to layout visualization in QCADesigner.

For each figure there is the indication of the name of the associated circuit file. These circuit files are available at the project website.

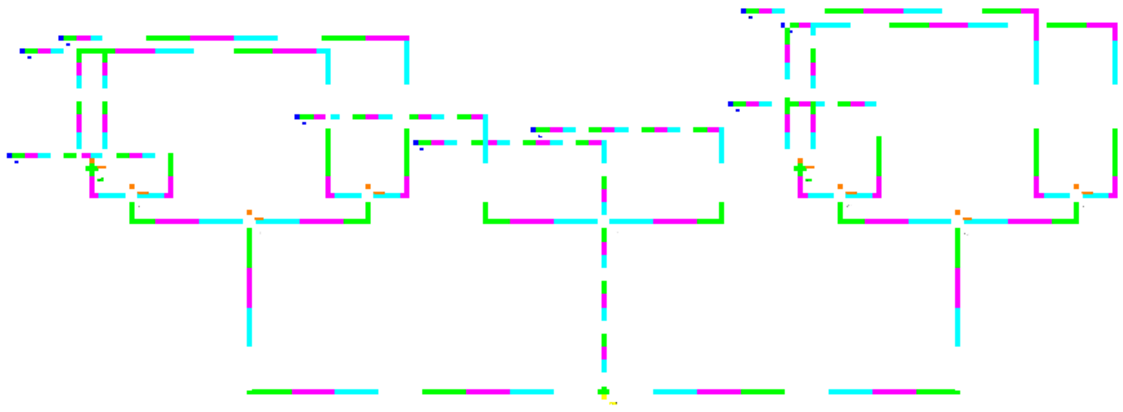


Figure A.1: The layout representation of an example. (File: maj2)

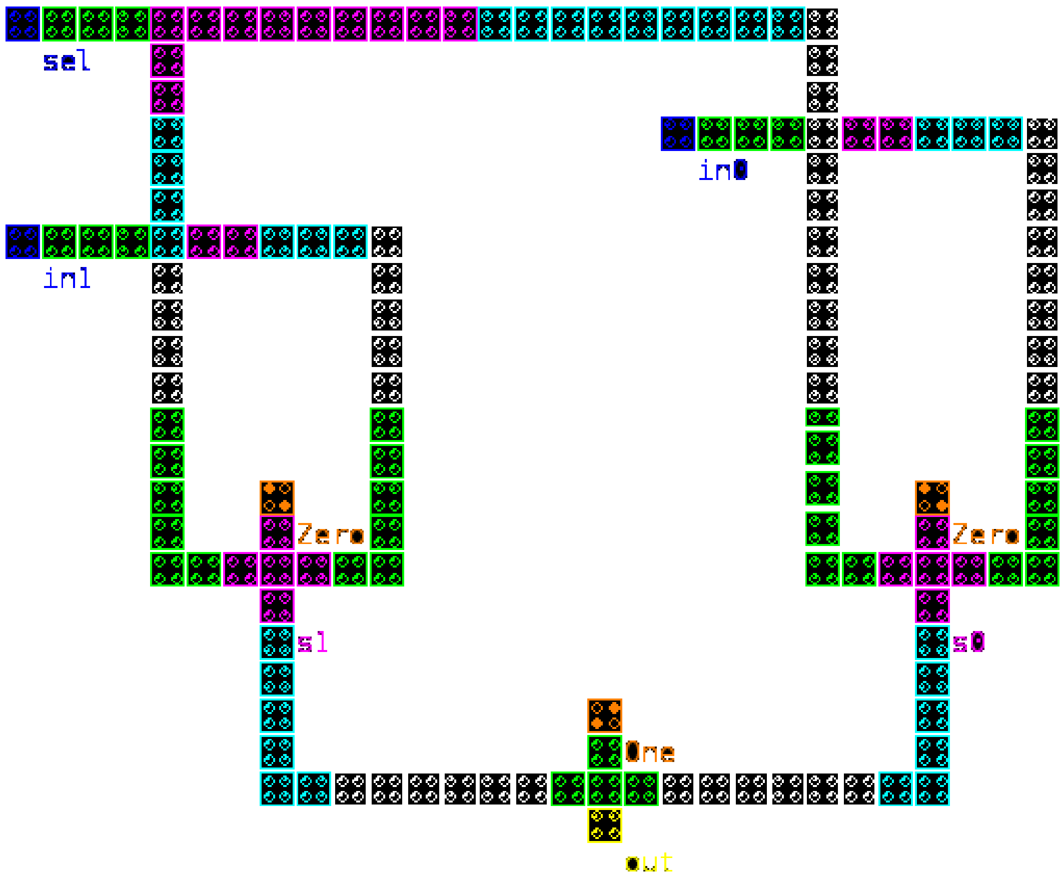


Figure A.2: The layout representation of a 1 bit, 2 to 1 multiplexer. (File: mux2)

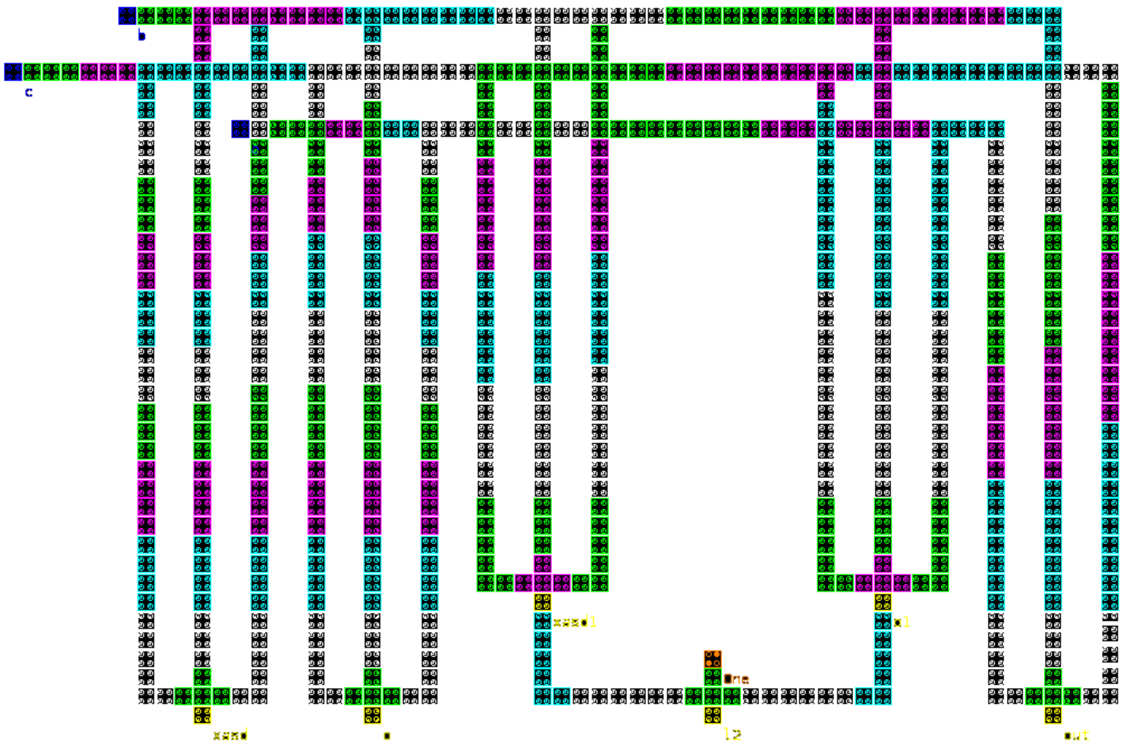


Figure A.3: The layout representation of an example. (File: simple32)

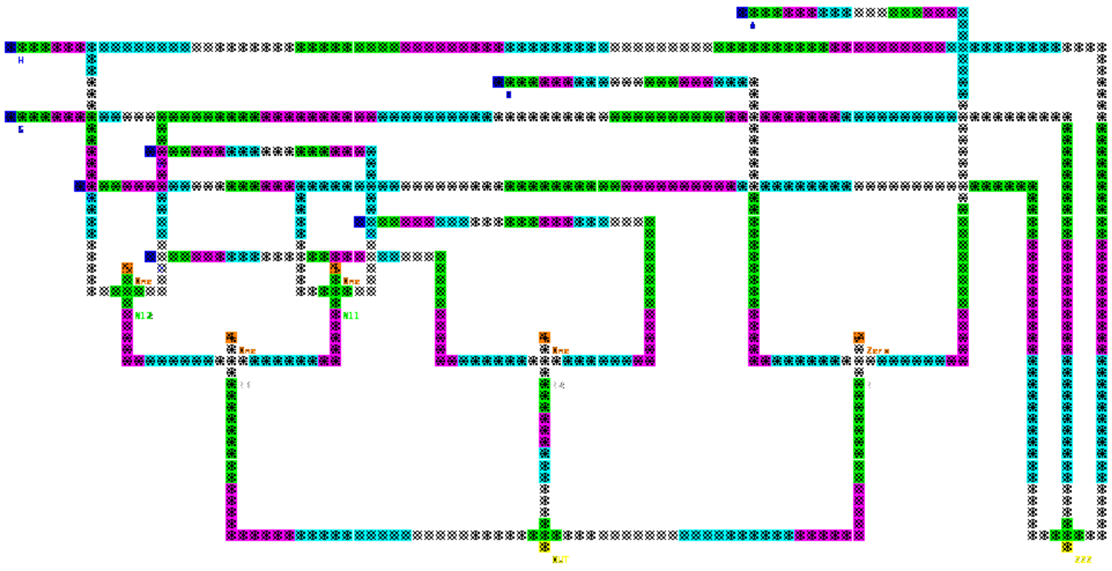


Figure A.4: The layout representation of an example. (File: simple-lsi)

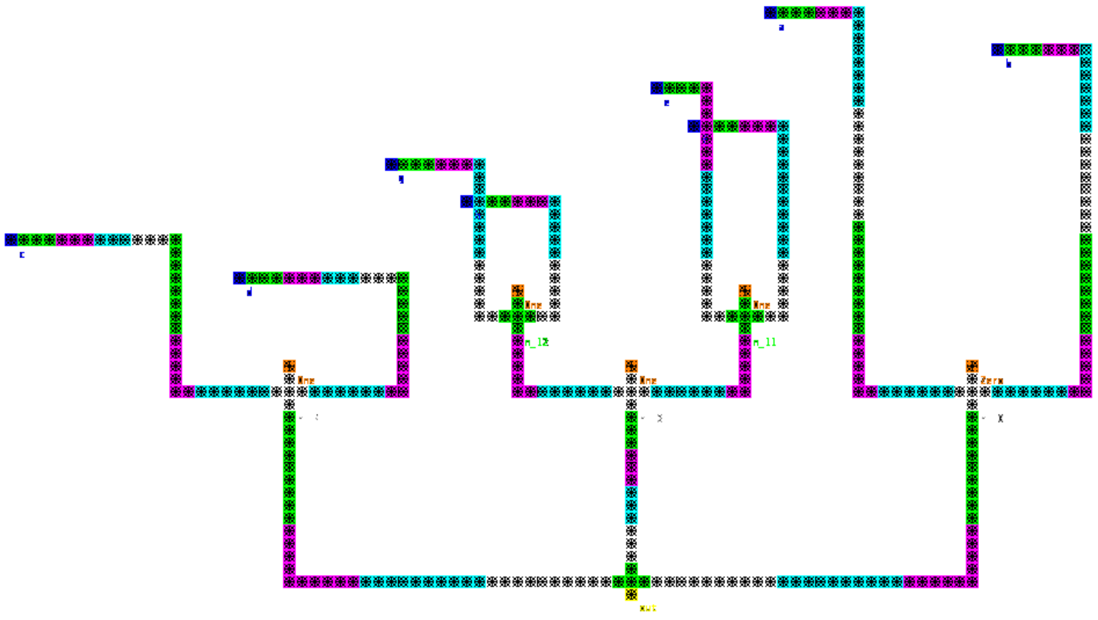


Figure A.5: The layout representation of an example. (File: simple)

B

Appendix B

Contents

B.1 VHDL description

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
------------------------------------------------------------
entity EXAMPLE is port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
        C: in STD_LOGIC;
        D: in STD_LOGIC;
        E: in STD_LOGIC;
        F: in STD_LOGIC;
SAIDA : out STD_LOGIC);
end EXAMPLE;
------------------------------------------------------------
architecture Behavioral of EXAMPLE is

component maj3 is port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
        C: in STD_LOGIC;
        Z: out STD_LOGIC);
end component;

component and2 is port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
        Z: out STD_LOGIC);
end component;

signal node1, node2 : STD_LOGIC;
------------------------------------------------------------
begin
        MAJ_1 : maj3 port map( A, B, C, node1 );
        MAJ_4 : and2 port map( E, F, node2 );
        process (D) begin
                if( D = '1' ) then
                        SAIDA <= node1;
                else
                        SAIDA <= node2;
                end if;
        end process;
end Behavioral;
------------------------------------------------------------
```

B.2 LSI netlist

```
COMPILE;
DIRECTORY MASTER;
/****** Technology used: qca4 ******/
MODULE EXAMPLE;
INPUTS  A,B,C,D,E,F;
OUTPUTS  SAIDA;
LEVEL FUNCTION;
DEFINE
A = (A);
B = (B);
C = (C);
D = (D);
E = (E);
F = (F);
SAIDA = (SAIDA);
U10(N5=Z) = NAND2(E=A,F=B);
U11(SAIDA=Z) = NAND2(N6=A,N7=B);
U12(N8=Z) = MAJ3(A=A,B=B,C=C);
U13(N7=Z) = NAND2(D=A,N8=B);
U14(N6=Z) = OR2(N5=A,D=B);
END MODULE;
END COMPILE;
END;
```

B.3 Synopsys library

```
library (qca4) {

date : "June 19, 2007";
revision : 0.2;

default_inout_pin_cap        :  1.0;
default_inout_pin_fall_res   :  0.0;
default_inout_pin_rise_res   :  0.0;
default_input_pin_cap        :  1.0;
default_intrinsic_fall       :  1.0;
default_intrinsic_rise       :  1.0;
default_output_pin_cap       :  0.0;
default_output_pin_fall_res  :  0.0;
default_output_pin_rise_res  :  0.0;
default_slope_fall           :  0.0;
default_slope_rise           :  0.0;

time_unit : "1ns";
voltage_unit : "1V";
current_unit : "1uA";
pulling_resistance_unit : "1kohm";
capacitive_load_unit (0.1,ff);


cell (MAJ3) {
  area : 25
  pin(A) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(B) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(C) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(Z) {
    direction : output
    function : "(A B)+(B C)+(A C)"
    max_fanout : 10
    timing() {
      intrinsic_rise : 0.490000
      intrinsic_fall : 0.800000
      rise_resistance : 0.185000
      fall_resistance : 0.059000
      related_pin : "A B C"
   }
 }
}

cell (OR2) {
  area : 4500
  pin(A) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(B) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(Z) {
    direction : output
    function : "(A+B)"
    max_fanout : 10
    timing() {
      intrinsic_rise : 10.490000
      intrinsic_fall : 10.800000
      rise_resistance : 10.185000
      fall_resistance : 10.059000
      related_pin : "A B"
   }
 }
}

cell (NOR2) {
  area : 4500
  pin(A) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(B) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(Z) {
    direction : output
    function : "(A+B)'"
    max_fanout : 10
    timing() {
      intrinsic_rise : 10.490000
      intrinsic_fall : 10.800000
      rise_resistance : 10.185000
      fall_resistance : 10.059000
      related_pin : "A B"
   }
 }
}

cell (AND2) {
  area : 4500
  pin(A) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(B) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(Z) {
    direction : output
    function : "(A B)"
    max_fanout : 10
    timing() {
      intrinsic_rise : 10.490000
      intrinsic_fall : 10.800000
      rise_resistance : 10.185000
      fall_resistance : 10.059000
      related_pin : "A B"
    }
  }
}

cell (NAND2) {
  area : 4500
  pin(A) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(B) {
    direction : input
    capacitance : 1
    fanout_load : 1.0
  }
  pin(Z) {
    direction : output
    function : "(A B)’"
    max_fanout : 10
    timing() {
      intrinsic_rise : 10.490000
      intrinsic_fall : 10.800000
      rise_resistance : 10.185000
      fall_resistance : 10.059000
      related_pin : "A B"
   }
 }
}

cell(INV) {
  area : 9
  pin(A) {
    direction : input
    capacitance : 1.0
    fanout_load : 1.0
  }
  pin(Z1) {
    direction : output
    max_transition : 3.0
    function : "A’"
    timing() {
      intrinsic_rise : 0.780000
      intrinsic_fall : 0.370000
      rise_resistance : 0.180000
      fall_resistance : 0.053000
      related_pin : "A"
    }
  }
}


}
```

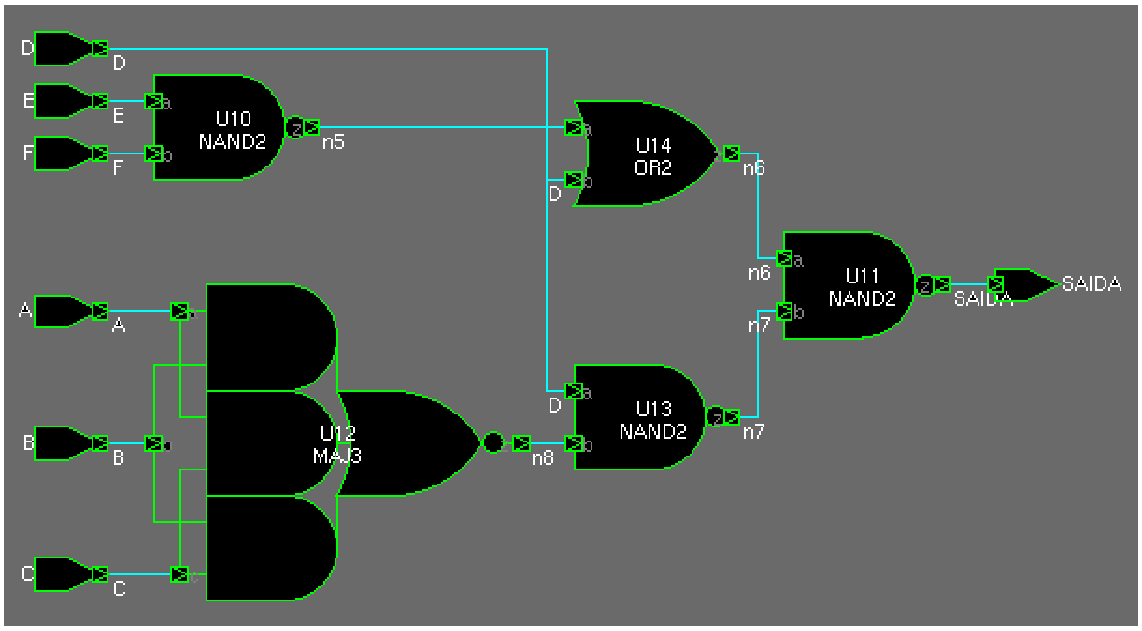


Figure B.1: The schematic representation of an example. (File: example)

B.4 BLIF description

```
.model blif_de_teste
.inputs a b c
.outputs out
.default_input_arrival 0 0
.names b c and_bc
11 1
.names b c or_bc
1- 1
-1 1
.names a or_bc and_or_a_bc
11 1
.names and_or_a_bc and_bc out
1- 1
-1 1
.end
```

B.5 Gate netlist

```
.model blif_de_teste
.inputs a b c
```

```
.outputs out
.default_input_arrival 0 0
.gate MAJ3 a=c b=b c=a Z1=out
.end
```

B.6 MVSIS script

```
read_library ./qca.genlib
read_super ./qca.super
read_blif ./simple2.blif
map
write_gate ./simple2.gate
```

B.7 MVSIS library

```
GATE INV 1 Z1=!a; PIN * UNKNOWN 1 3 1 1 1 1
GATE AND2 1 Z1=a*b; PIN * UNKNOWN 1 3 1 1 1 1
GATE NAND2 1 Z1=!(a*b); PIN * UNKNOWN 1 3 1 1 1 1
GATE OR2 1 Z1=a+b; PIN * UNKNOWN 1 3 1 1 1 1
GATE MAJ3 1 Z1=(a*b)+(b*c)+(a*c); PIN * UNKNOWN 1 3 1 1 1 1
GATE zero 0 O=CONST0;
GATE one 0 O=CONST1;
```

B.8 Custom super gate library

```
#
# Supergate library derived for "qca.genlib" on Wed May 16 18:46:05 2007.
#
# Command line: "super  -i 3  -l 2  -d 1000.00  -a 1000.00  -t 60    qca.genlib".
#
# The number of inputs      =          3.
# The number of levels      =          2.
# The maximum delay         =    1000.00.
# The maximum area          =    1000.00.
# The maximum runtime (sec) =         60.
#
# The number of attempts    =       1755.
# The number of supergates  =         96.
# The number of functions   =          0.
# The total functions       = 256 (2^8).
#
# Generation time (sec)     =       0.01.
#
qca.genlib
3
96
105
* AND2 1 0
NAND2 2 1
* AND2 3 4
* NAND2 1 0
AND2 2 0
* AND2 6 7
* AND2 0 6
* AND2 0 4
* OR2 1 0
* AND2 11 4
OR2 2 0
* AND2 4 13
AND2 2 1
* AND2 6 15
* AND2 1 6
NAND2 2 0
* AND2 1 18
* AND2 11 18
OR2 2 1
* AND2 18 21
* AND2 6 11
* AND2 2 6
* AND2 6 13
* AND2 6 21
* AND2 0 15
* AND2 1 7
* AND2 2 3
* MAJ3 0 3 18
* MAJ3 1 3 4
* MAJ3 0 6 7
* AND2 0 21
* MAJ3 1 0 7
* MAJ3 2 0 3
* AND2 0 11
* MAJ3 0 4 15
* MAJ3 1 4 7
* MAJ3 1 0 4
* MAJ3 2 4 7
* MAJ3 2 3 4
* MAJ3 2 0 4
* MAJ3 3 4 21
* MAJ3 0 4 21
* MAJ3 1 6 15
* AND2 1 13
* MAJ3 1 0 15
* MAJ3 2 1 3
* MAJ3 0 15 18
* AND2 1 11
* MAJ3 1 18 7
* MAJ3 1 0 18
* MAJ3 2 15 18
* MAJ3 2 3 18
* MAJ3 3 13 18
* MAJ3 2 1 18
* MAJ3 1 13 18
* AND2 2 11
* MAJ3 2 1 7
* MAJ3 2 0 15
* MAJ3 0 6 15
* MAJ3 1 6 7
* MAJ3 6 11 15
* MAJ3 2 1 0
* OR2 0 15
* MAJ3 2 0 11
* MAJ3 1 0 13
* OR2 1 7
* MAJ3 2 1 11
* MAJ3 1 0 21
* MAJ3 2 6 3
* MAJ3 2 0 6
* MAJ3 2 1 6
* MAJ3 2 6 11
* MAJ3 2 1 13
* OR2 2 3
* MAJ3 2 0 21
* OR2 0 21
* OR2 1 13
* OR2 2 11
* INV 11
* NAND2 2 11
* NAND2 1 11
* NAND2 1 13
* NAND2 0 11
* NAND2 0 21
* NAND2 1 7
* NAND2 0 15
* NAND2 2 3
* NAND2 6 21
* NAND2 6 13
* NAND2 2 6
* NAND2 6 11
* NAND2 18 21
* NAND2 11 18
* NAND2 1 18
* NAND2 1 6
* OR2 0 4
* NAND2 4 13
* NAND2 11 4
* NAND2 0 4
* NAND2 0 6
* OR2 1 18
* OR2 2 6
```