UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Tese

Cell Implementation Through Boolean Satisfiability for Conventional and Emerging Technologies

Maicon Schneider Cardoso

Pelotas, 2022

Maicon Schneider Cardoso

Cell Implementation Through Boolean Satisfiability for Conventional and Emerging Technologies

Tese apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Advisor: Prof. Dr. Felipe de Souza Marques
Coadvisor: Prof. Dr. Leomar Soares da Rosa Jr.

Pelotas, 2022

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

C268c Cardoso, Maicon Schneider

Cell implementation through boolean satisfiability for conventional and emerging technologies / Maicon Schneider Cardoso ; Felipe de Souza Marques, orientador ; Leomar Soares da Rosa Júnior, coorientador. — Pelotas, 2022.

141 f. : il.

Tese (Doutorado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2022.

1. Projeto físico de células lógicas. 2. Satisfatibilidade booleana. 3. Static CMOS Complex Gate. 4. Quantum-dot cellular automata. 5. Nanomagnetic logic. I. Marques, Felipe de Souza, orient. II. Rosa Júnior, Leomar Soares da, coorient. III. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Maicon Schneider Cardoso

Cell Implementation Through Boolean Satisfiability for Conventional and Emerging Technologies

Tese aprovada, como requisito parcial, para obtenção do grau de Doutor em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 23 de fevereiro de 2022

Banca Examinadora:

Prof. Dr. Felipe de Souza Marques (orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Leomar Soares da Rosa Jr. (coorientador)
Doutor em Microeletrônica pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Bruno Zatt
Doutor em Microeletrônica pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Paulo Francisco Butzen
Doutor em Microeletrônica pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. José Augusto Miranda Nacif
Doutor em Ciência da Computação pela Universidade Federal de Minas Gerais.

Para Pam e meus pais.

ACKNOWLEDGEMENTS

I took the liberty of writing this section in Portuguese and English to address the people here cited.

Começo essa seção de agradecimentos com uma pequena história: certa feita, lá por meados de 2002, o desejo de seguir a carreira científica surgiu em mim após entender que o que eu mais desejava para o restante da minha vida profissional era algo relativamente simples: seguir aprendendo e descobrindo. No entanto, tornar-me um cientista, seja da área que fosse, era algo quase que impensável: não haviam exemplos próximos a mim e nem perspectivas concretas. As chances eram mínimas, segundo o que eu compreendia na época. Se eu pudesse retornar a esse momento, diria para esse jovem Maicon que, ainda que seja muito (muito mesmo!) difícil, com as pessoas e as oportunidades certas, nada é impossível e todos os objetivos são alcançáveis. É para essas pessoas fundamentais no meu caminho - em especial nesses últimos anos de doutorado - que endereço boa parte desses agradecimentos.

Inicialmente, agradeço às agências brasileiras de fomento à pesquisa (Capes, CNPq e Fapergs) pelo suporte financeiro durante o período de doutorado, além do fundamental apoio para a realização do doutorado sanduíche no exterior, crucial para o desenvolvimento dessa tese. Aproveito para agradecer à Universidade Federal de Pelotas (UFPel) e à Universitat Politècnica de Catalunya (UPC) pela infraestrutura disponibilizada para a realização de parte dos experimentos dessa pesquisa.

Agradeço aos amigos de laboratório pelos momentos ótimos (e também pelos não tão ótimos assim) que dividimos em meio a muito café e chimarrão. Cito aqui os amigos Renato de Souza e Plínio Finkenauer, os quais estiveram comigo durante boa parte dessa jornada. Vocês transformaram a rotina diária em algo leve e descontraído.

Agradeço aos meus professores, que pavimentaram o caminho até aqui. Em especial, agradeço aos meus orientadores Felipe Marques e Leomar da Rosa Jr. pelo apoio não somente em questões técnicas e de pesquisa, mas também pela parceria e amizade firmada nesses últimos anos. Espero que possamos seguir trabalhando juntos mesmo após o fim do doutorado (e que possamos lembrar dos bons e maus momentos vividos em cada churrasco realizado daqui pra frente). Furthermore, I would like to thank professor Jordi Cortadella for their incredible and crucial support during my time in Barcelona. Jordi was not only the mentor of most of the models here presented but also a special human being. Thanks for your patience, advice, and for kindly welcoming me and my wife as long-time friends, even without knowing us. Moltes gràcies!

Gostaria de agradecer aos meus pais, Darli Schneider, Algeu Cardoso e Dalva Schneider (tia e mãe de coração). A dedicação de vocês fez com que eu e meu irmão pudéssemos buscar os nossos objetivos e sonhos, e serei sempre grato por isso.

Obrigado por me inspirarem e por toda a confiança depositada em mim. Amo vocês.

Finalmente, agradeço a minha esposa, Pamela Acosta, pelo amor, companheirismo, carinho e dedicação. Serei eternamente grato pelo apoio incondicional para que eu pudesse seguir o doutorado. Obrigado por não ter desistido de mim mesmo nos momentos difíceis dessa caminhada. Obrigado por ter me inspirado a buscar a experiência internacional e por encarar esse desafio comigo mesmo tendo que abrir mão de muita coisa pra isso. Obrigado pelos momentos - dos muitos bons aos poucos ruins -, por dividir o chimarrão em terras distantes, pelas viagens e passeios. Obrigado pelas memórias. Agradeço pela leveza que só tu tens, mesmo em tempos sombrios e difíceis de pandemia. Por fim, agradeço por te ter ao meu lado todos os dias. Eu te amo.

Por último, presto meu respeito e solidariedade às famílias que perderam seus entes queridos nessa pandemia.

Happiness only real when shared.
— CHRISTOPHER J. MCCANDLESS

ABSTRACT

CARDOSO, Maicon Schneider. **Cell Implementation Through Boolean Satisfiability for Conventional and Emerging Technologies**. Advisor: Felipe de Souza Marques. 2022. 141 f. Thesis (Doctorate in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2022.

The electronic design automation (EDA) tools take a crucial role in the modern digital circuits and systems synthesis, where the design challenges are not only numerous but also complex. In this scenario, the Boolean satisfiability (SAT) solvers have been employed lately as a useful engine for computing the solutions on these EDA tools, producing circuits with good quality in a reasonable computing time. On a similar note, the versatility provided by the satisfiability paradigm can be explored for different design purposes ranging from conventional to emerging technologies. Thus, in this thesis, we employ this approach to generate area-optimized circuits in three different technologies: static CMOS complex gates (SCCG), quantum-dot cellular automata (QCA), and nanomagnetic logic (NML). Considering this, the proposed methods were able to encode all the design constraints into a discrete constraint model, using satisfiability solvers as the core of the optimization task. Regarding the SCCG synthesis, the experiments have shown that, besides providing improvements on layout area, the solutions produced using the proposed method also presented optimization in other geometrical parameters such as in wirelength and number of contacts when compared to a traditional meta-heuristic approach. Furthermore, following the experiments on the emerging technologies, the QCA and NML synthesis methodologies were able to provide solutions with less area when compared to other graph-based techniques available in the literature for most of the assessed cases. Moreover, the latency of these solutions also presented an optimization, thus providing a wide design exploration scenario where it is possible to choose whether to use smaller or faster circuits depending on the specifications.

Keywords: Electronic Design Automation. Cell Implementation. Boolean Satisfiability. Static CMOS Complex Gate. Quantum-dot Cellular Automata. Nanomagnetic Logic.

RESUMO

CARDOSO, Maicon Schneider. **Projeto Digital de Células via Satisfatibilidade Booleana em Tecnologias Convencionais e Emergentes**. Advisor: Felipe de Souza Marques. 2022. 141 f. Thesis (Doctorate in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2022.

As ferramentas de apoio ao projeto eletrônico (EDA) desempenham um papel fundamental na síntese de circuitos e sistemas digitais modernos, em que os desafios de projeto envolvidos são numerosos e complexos. Nesse cenário, os resolvedores de satisfatibilidade Booleana (SAT) vêm sendo amplamente empregados nos núcleos de ferramentas de EDA, gerando soluções com boa qualidade em um tempo de computação viável. Dada a versatilidade dessa abordagem, o projeto de circuitos via modelagem SAT pode ser explorado para atender a diferentes propósitos, incluindo o uso tecnologias convencionais e emergentes. Nessa tese utilizamos a modelagem e os resolvedores SAT para o propósito de gerar soluções com otimização em área em três diferentes tecnologias: *static CMOS complex gates* (SCCG), *quantum-dot cellular automata* (QCA) e *nanomagnetic logic* (NML). Para tanto, os métodos propostos apoiam-se na descrição formal das regras de projeto de cada tecnologia, utilizando-se dos resolvedores SAT como protagonistas no processo de síntese. Em relação ao projeto para SCCGs, os experimentos apontam que a abordagem proposta, além de apresentar ganhos em área, também demonstrou-se eficiente considerando outros aspectos geométricos do leiaute como o comprimento de fio utilizado para roteamento (wirelength) e o número de contatos necessários para conectar as partes da célula. Ademais, considerando os experimentos conduzidos para as tecnologias emergentes, tanto a síntese voltada QCA quanto para NML também apresentaram bons resultados quanto à diminuição de área do circuito para boa parte do *benchmark* utilizado. Por fim, a latência dessas soluções também apresentou otimizações, propiciando um cenário onde o projetista pode escolher qual circuito atende melhor as suas necessidades de acordo com o perfil de área e latência especificados.

Palavras-chave: Ferramenta de Apoio ao Projeto Eletrônico. Projeto Físico de Células Lógicas. Satisfatibilidade Booleana. Static CMOS Complex Gate. Quantum-dot Cellular Automata. Nanomagnetic Logic.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

P4	4-input p-class
53NSP	53 NSP handmade network
ASTRAN	Automatic Synthesis of Transistor Networks
BANCS	Bidirectional Alternating Nanomagnetic Clocking Scheme
SAT	Boolean satisfiability
CMOS	Complementary metal-oxide-semiconductor
CNF	Conjunctive normal form
DAG	Directed acyclic graph
EDA	Electronic design automation
FCN	Field-coupled nanocomputing
FPGA	Field-programmable gate array
FinFET	Fin field-effect transistor
ILP	Integer linear programming
IC	Integrated circuit
ISOP	Irredundant sum-of-products
KF	Kernel Finder
LE	Logical Effort
MQCA	Magnetic QCA
mRQ	Minor research questions
NML	Nanomagnetic logic
NP	Non-deterministic polynomial
NSP	Non-series-parallel
NMOS	N-type metal-oxide-semiconductor
OGD	Orthogonal graph drawing
PMOS	P-type metal-oxide-semiconductor
PD	Pull-down

P&R	Place-and-route
PU	Pull-up
QCA	Quantum-dot cellular automata
RH	Research hypotheses
RES	Robust, Efficient and Scalable
SMT	Satisfiability modulo theories
SCCG	Static CMOS complex gate
TA	Threshold accepting
UFPel	Universidade Federal de Pelotas
UPC	Universitat Politècnica de Catalunya
USE	Universal, Scalable and Efficient
UNSAT	Unsatisfiable

CONTENTS

1 INTRODUCTION

Digital circuits and systems take a leading role in the modern world. They are crucial components of different areas such as medicine, agriculture, engineering, communication, education, among many others. In this scenario, the constant development of methodologies and tools to accelerate the digital design task are necessary to follow the pace at which the technology evolves over the years.

Considering this, the electronic design automation (EDA) tools play an important role as they enable the modern digital design processes in all its stages. Since the first pioneer researches and developments of EDA solutions back in the 1960s (WANG; CHANG; CHENG, 2009), these tools have been evolving following the growth of the circuits that they give support. Nowadays, EDA tools can deal with instances containing billions of components - an important feature considering actual commercial FPGAs (INTEL, 2019) and processors (AMD, 2020). Thus, the core of these tools needs to be powerful and versatile to be able to work on different design problems that emerge depending on the conditions and technologies involved.

Related to that, Boolean satisfiability (SAT) modeling recently become a valuable tool for designing EDA solutions. SAT is an old problem in computer science known to be NP-complete which consists in finding if there is a satisfiable assignment for a Boolean variable set such that a function built over this set evaluates to true. The recent advances in this topic were driven by the fact that the SAT solvers, the engines capable of solving this fundamental computing problem, have evolved tremendously in the last years, such as Figure 1 illustrates. Nowadays, these tools can handle instances containing over a million variables and several million clauses (GOMES et al., 2008), being suitable for real-world applications. Indeed, SAT is a useful tool for dealing with different problems in various fields such as in artificial intelligence (KAUTZ; SELMAN, 1992; RINTANEN; HELJANKO; NIEMELÄ, 2006), bioinformatics (LYNCE; MARQUES-SILVA, 2006; NEIGENFIND et al., 2008), software model checking and testing (CLARKE; KROENING; LERDA, 2004; JACKSON; SCHECHTER; SHLYAKHTER, 2000; KHURSHID; MARINOV, 2004), automated theorem proving (BROWN, 2011), electronic design automation (MARQUES-SILVA; SAKALLAH, 2000), among

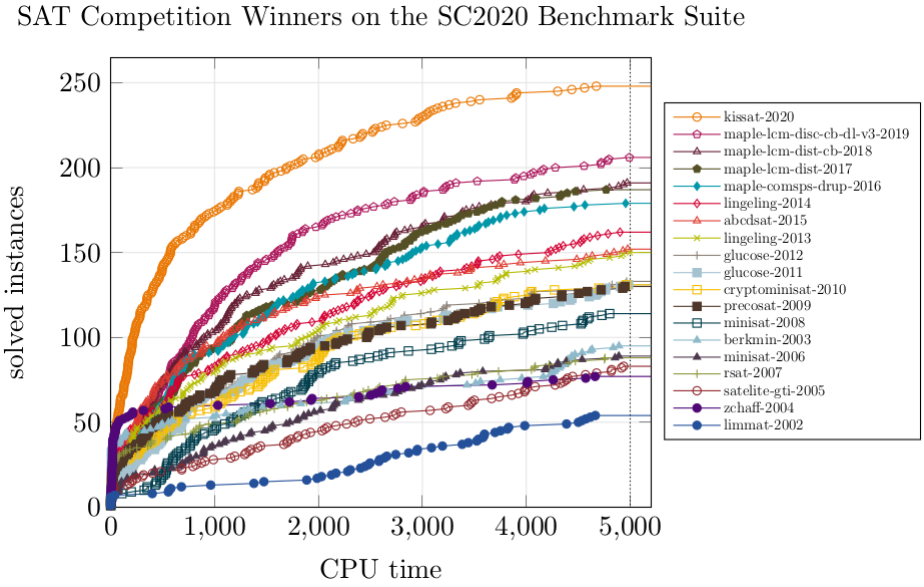


Figure 1 – Evolution of the Boolean satisfiability solvers assessed from the results of the SAT competition winners from 2002 until 2020.
Source: (BIERE; HEULE, 2020).

many others.

Considering the SAT for EDA paradigm, different approaches have been proposed in the past years. From the first FPGA routing methods based on SAT (WOOD; RUTENBAR, 1998), numerous applications were described aiming at different topics such as timing analysis (SILVA et al., 1998), fault diagnosis and logic debugging (CHEN; GUPTA, 1996; SMITH et al., 2005), among others. Driven by the success in formal verification (GANAI; GUPTA, 2007), SAT is intensively applied in different fields of logic synthesis, such as in the areas of automatic test pattern generation (FUJITA et al., 2015), equivalence checking (MISHCHENKO et al., 2006), functional dependency (JIANG et al., 2010), functional decomposition (LEE; JIANG; HUNG, 2008), Boolean matching (SOEKEN et al., 2016), technology mapping (LING; SINGH; BROWN, 2005), etc.

Specifically, in the cell synthesis scenario, SAT-based methods are becoming an important ally to deal with different aspects of this design task. From the first SAT-based method on this topic, which deals with the channel routing problem (DEVADAS, 1989), other applications emerge, such as for the technology-dependent place-and-route (P&R) (IIZUKA; IKEDA; ASADA, 2004, 2005), for FinFET placement (SOROKIN; RYZHENKO, 2019), for technology-independent cell routing (CORTADELLA et al., 2014), for regular logic bricks design (TAYLOR; PILEGGI, 2007), among others.

In this scenario, we identify that SAT can be exploited even more for solving problems in the cell implementation domain. The next chapters of this thesis will explore this valuable tool for solving problems in different topics of cell synthesis in conventional and emerging technologies.

1.1 Research Questions and Hypotheses

Considering the scenario illustrated in the previous section, where the capabilities of SAT solvers increases dramatically in the past years, the following research question arises: **how to employ the discrete constraint modeling and the Boolean satisfiability solvers to deal with various challenges in the cell synthesis?** Indeed, this question can be divided into three minor research questions (mRQ) to simplify and clarify the discussion. This way, we can define the minor research questions as follows:

- **mRQ1**: how to make use of the capabilities of modern SAT solvers for dealing with cell synthesis problems in conventional (transistor-based) technologies?
- **mRQ2**: how to employ SAT solvers to deal with cell synthesis challenges for field-coupled nanotechnologies?
- **mRQ3**: what is the quality of the solutions provided by the SAT-based approaches in comparison with other traditional methodologies?

To answer these questions, we raise some research hypotheses (RH) listed as follows:

- **RH1**: the transistor placement task plays an important role in the cell synthesis since it is a procedure taken in the first stage of the physical design flow, i.e., it is employed previous to the routing and compaction tasks. Thus, a good solution for placement can lead to optimizations in the cell area even though maintaining the same routing and compaction routines. In this sense, we hypothesize that it is possible to encode the placement restrictions of a particular technology node into an SAT problem and, afterward, use SAT solvers to compute the minimal solutions that fulfill the requirements of the designed SAT model. This way, it can serve as a good alternative to other placement approaches since the cells produced through the SAT-based method potentially use fewer columns for the layout.
- **RH2**: the physical synthesis on field-coupled nanotechnologies such as QCA and NML is a complex process where the placement and routing should be computed taking into account the synchronization restrictions inherent to these technologies. Thus, we hypothesize that encoding all the design restrictions - i.e., the placement, routing, and synchronization constraints - of QCA and NML into an SAT problem and, then, solving it through an SAT solver can lead to area-optimized solutions that fulfill all the design requirements.
- **RH3**: to draw this research hypothesis, first it is necessary to define the assessment scenarios:

i. Considering the transistor-based design, all the assessments consist of a comparison with a threshold accepting method implemented in an EDA tool for automatic cell design known as ASTRAN. In this sense, to assess the quality of the solutions produced through the SAT-based proposed approach, we draw two testing scenarios: the first consists in assessing the number of columns used only for the placement, while the second aims to evaluate the layout (in terms of area, wirelength and number of contacts) produced using the proposed method integrated with the ASTRAN infrastructure;

ii. Considering the synthesis for QCA and NML, the assessments consist of a comparison with well-known graph-based approaches under the same conditions (clocking schemes and synchronization profiles). Besides area, other cell parameters are evaluated such as latency and cell density.

Thus, based on these proposed assessments, we hypothesize that the constraints designed for each technology lead to optimized circuits in comparison with different approaches available in the literature. We also hypothesize that, even though the focus of the method is area optimization, the other cell parameters also are optimized since they are related to the area of the circuit.

1.2 Goal and Thesis Contributions

Based on the research questions and hypotheses raised in the previous section of this chapter, we are now able to define the main goal of this thesis. Thus, this thesis **proposes a new Boolean satisfiability approach for cell synthesis in transistor-based and field-coupled technologies that leads to solutions with optimizations in the layout area.**

The main contributions of this thesis can be summarized as follow:

- SAT-based model for SCCG placement: we propose a new approach for encoding the transistor placement task as a decision problem, where transistors and their positions are represented as Boolean variables, and the constraints are built based on the design rules of the 65nm technology over this set of variables. Thus, a new algorithm is proposed - where a satisfiability solver is used as the core of implementation -, enabling to find area-optimized solutions that fulfill the design rules.

- Improvements on ASTRAN: as a byproduct of the previous item, we integrate our new placement approach into ASTRAN, an academic open-source solution widely used for cell synthesis. Thus, a new automatic placement approach aiming to obtain area-optimized cells through ASTRAN is available for covering the scenarios where this cell profile is desirable.

- SAT-based model for QCA: we propose a new methodology for modeling the synthesis problem in QCA, where the QCA cells and the grid configurations are encoded as Boolean variables, and the placement, routing, and synchronization constraints are defined based on the design rules described by a clocking scheme called USE. From that, we describe a new algorithm - which also uses a satisfiability solver as its core - to compute area-optimized solutions in QCA. Along with that, an idea of the tradeoff between area and throughput is introduced based on the proposed formulation.
- SAT-based model for NML: we propose a new methodology for modeling circuit design challenges in NML, where the NML cells and the grid configurations are encoded as Boolean variables, and the placement, routing, and synchronization restrictions are built based on the design rules defined by a clocking scheme called BANCS. Moreover, we describe a new synthesis algorithm - which also adopts a satisfiability solver in its core - to obtain solutions with optimizations in the layout area. Furthermore, a tradeoff profile between area and throughput is introduced based on the proposed formulation.

1.3 Thesis Outline

This thesis is organized as follows:

- **Chapter 2** reviews some basic concepts to enable a more complete understanding of the thesis. Fundamentals of Boolean algebra, such as Boolean functions, variables, and operators, are presented. Besides that, a brief introduction about Boolean satisfiability is conducted, including the presentation of the Boolean satisfiability solver adopted. Finally, the cardinality constraints broadly used in the proposed models are formally defined.
- **Chapter 3** presents the first method proposed in this thesis, which is focused on the design of static CMOS complex gates (SCCGs). The placements constraints of SCCGs are modeled in terms of an SAT problem so it can be solved through an SAT solver.
- **Chapter 4** introduces the second method proposed in this monograph, which is focused on the QCA design. Based on a well-known clocking scheme called USE, the placement, routing, and synchronization constraints are encoded in a SAT problem to be computed through a SAT solver.
- **Chapter 5** provides the third method proposed in this thesis, which is focused on the NML design. Similar to the chapter 4, the NML synthesis constraints are based on a well-known clocking scheme called BANCS, where all placement,

routing, and synchronization restrictions are modeled in terms of a SAT problem to be handled by a SAT solver.

- **Chapter 6** concludes this thesis, where the advances proposed are summarized and compared with the objectives and goals. Furthermore, ideas for future work are provided so it can drive the next researches and extend the presented work.

2 BACKGROUND

In this chapter, we provide background information on the basic concepts necessary for a complete understanding of this thesis. First, we introduce some fundamentals of Boolean algebra (Boolean variables, functions, operators, etc.). Afterward, we define Boolean satisfiability and satisfiability solvers. Finally, we present the cardinality constraints which are used along this monograph.

2.1 Boolean Algebra

Firstly, consider $B = \{0, 1\}$ as the **Boolean set** whose elements are logic values typically interpreted as 0 for $false$ and 1 for $true$. In this sense, a **Boolean function** is a function $f : B^n \rightarrow B$, such that $n \geq 1$. In other words, a Boolean function describes how to compute a Boolean value output based on logic calculations from the Boolean input vector of size n.

Boolean variables are variables defined in the Boolean domain B and are generally assigned using alphanumeric characters (such as a, b, x_0, x_1, etc.) Boolean variables can assume arbitrary values in the Boolean domain, i.e., they can assume the values 0 or 1. A **literal** is an occurrence of a variable or its complement in a Boolean function.

The basic **Boolean operators** are the conjunction (and), disjunction (or) and negation (not). The symbols used for this operators are $\wedge$, $\vee$, and $\neg$, respectively. Thus, considering the set of n Boolean variables $X = \{x_0, x_1, ..., x_{n-1}\}$, we have

$$x_0 \wedge x_1 \wedge ... \wedge x_{n-1} \equiv true \text{ iff } x_0, x_1, ..., \text{ and } x_{n-1} \text{ are all } true, \quad (1)$$

$$x_0 \wedge x_1 \vee ... \vee x_{n-1} \equiv true \text{ iff } x_0, x_1, ..., \text{ or } x_{n-1} \text{ are, at least one, } true, \quad (2)$$

$$\neg x_i \equiv true \text{ iff } x_i \text{ is } false \text{ and } \neg x_i \equiv false \text{ iff } x_i \text{ is } true. \quad (3)$$

2.2 Boolean Satisfiability

A disjunction of n literals forms a **clause**, i.e., $c = l_0 \vee l_1 \vee ... \vee l_{n-1}$. To solve a Boolean satisfiability problem, a Boolean formula is converted into its **conjunctive**

normal form (CNF) as a conjunction of m clauses, $F = c_0 \wedge c_1 \wedge ... \wedge c_{m-1}$. Algorithms such as the Tseitin transformation (TSEITIN, 1983) convert a Boolean function into a set of CNF clauses.

A **Boolean satisfiability** (SAT) problem is a decision problem that takes a propositional formula in CNF form and returns that the formula is satisfiable if there is an assignment of the variables from the formula for which the CNF evaluates to $true$. Otherwise, the propositional formula is unsatisfiable (UNSAT). A program that solves SAT problems is called a **SAT solver**. SAT solvers provide a satisfying assignment when the problem is satisfiable (PETKOVSKA, 2017).

Throughout this thesis we adopted the **CryptoMiniSat** (SOOS; NOHL; CASTELLUCCIA, 2009) as our SAT solver. CryptoMiniSat is a version of the well-known MiniSat solver (EÉN; SÖRENSSON, 2003). We decided to use CryptoMiniSat for its compatibility with C++ and Python (both languages used for implementing the algorithms proposed). Together with CryptoMiniSat we used **PyEDA** (AQAJARI et al., 2021) for dealing with the Tseitin transformations.

2.3 Cardinality Constraint

A **cardinality constraint** is a constraint on the number of literals set to $true$ in a given Boolean formula (BIERE et al., 2009). Hence, considering a set of n Boolean variables $X = \{x_0, x_1, ..., x_{n-1}\}$, a cardinality constraint can be represented in the form of a pseudo-Boolean formula as

$$\sum_{i=0}^{n-1} x_i \gtreqless k, \tag{4}$$

where k is an integer bound such that $0 \leq k \leq n-1$ and $\gtreqless$ is any relation of the set $\{\leq, \geq, =\}$. Considering this, three cardinality constraints are defined:

- $atmost(k, X)$ is $true$ iff at most k variables of X are $true$;
- $atleast(k, X)$ is $true$ iff at least k variables of X are $true$;
- $exactly(k, X)$ is $true$ iff exactly k variables of X are $true$.

There are several strategies proposed in the literature for encoding these constraints as presented in (WYNN, 2018).

3 STATIC CMOS COMPLEX GATES

Digital circuits and systems are fundamental to our modern world. They are applied in different areas of society such as in health, education, agriculture, economy, sports, transportation, among many others. In this scenario, most of these solutions are designed into integrated circuits (IC), devices that comprises multiple components into a single piece of silicon. Nowadays, it is possible to build chips containing billions of transistors, the basic computing unit.

In this scenario, the classic problems in digital circuit design are related to the optimizations of circuit parameters, such as area, power, and delay. In the past decade, Static CMOS Complex Gates (SCCGs) have been pointed as a promising alternative to supplant the traditional standard cell design (REIS, 2011). Producing solutions where Boolean functions with several inputs are implemented using a single gate, through this paradigm it is possible to provide on-demand circuits that meet restrictive project specifications.

Typically, the design flow of a complex gate is composed of three major steps: the first consists of generating and sizing a specialized transistor network that implements a given Boolean logic function; the second procedure is the layout generation, in which the transistor network is designed at physical level considering a specific technology; finally, the last task is the validation, verification and models generation, where the solution produced by the previous steps is evaluated regarding its geometric and electric aspects in order to analyze if it meets the expected behavior and the specified constraints.

Regarding the transistor network generation, there are several methodologies to perform this task (MARTINS et al., 2010; GOLUMBIC; MINTZ; ROTICS, 2008; ROSA et al., 2009; KAGARIS; HANIOTAKIS, 2007; POSSANI et al., 2016). The well-known Boolean factoring paradigm is the main way to handle this problem. It consists of finding an equivalent factored form solution (therefore, with fewer literals) from a given Boolean function. Hence, a complex gate can be obtained through series-parallel associations according to the and/or operations between the literals. On the other hand, recent papers have shown that graph-based methodologies can deliver cells with fewer

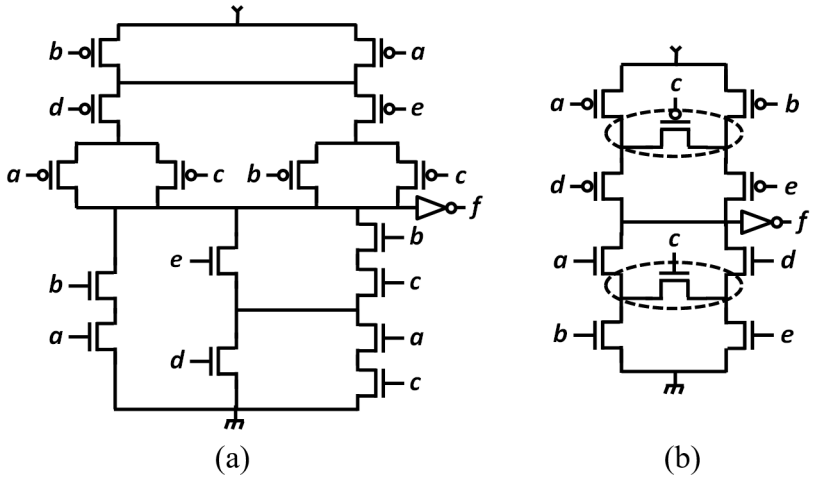


Figure 2 – Transistor networks implementing the Boolean function presented in equation 5. (a) Transistor network obtained through Boolean factoring. (b) Transistor arrangement provided by a graph-based approach.

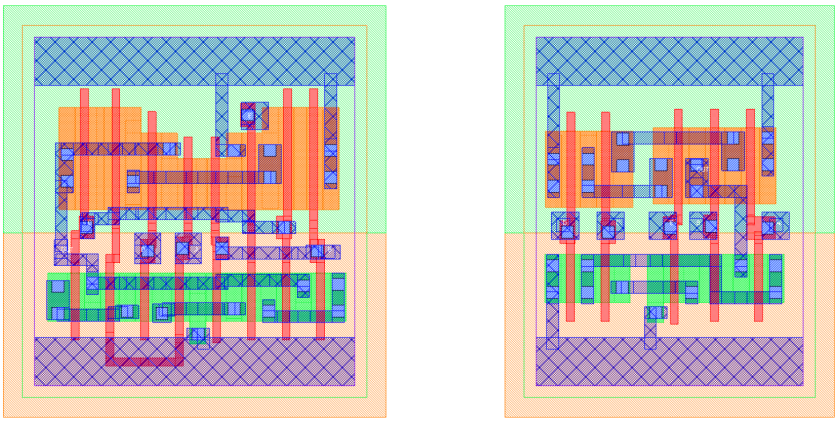
transistors compared to the solutions obtained through the classical Boolean factoring paradigm. This approach is based on the path sharing technique performed in order to optimize the transistor count. Due to the adoption of graphs as its data structure, some manipulations in the transistor arrangements potentially lead to non-regular topologies, i.e., graph-based methodologies can produce non-planar and non-dual cells.

To illustrate the potential of the SCCG-based design, consider the circuits presented in Figure 2 which implements the Boolean equation 5 below. Each solution is associated with the number of transistors necessary for its implementation, a parameter used for a estimate of the quality of the solution in terms of area, delay, and power (REIS, 2011; POSSANI et al., 2016). Therefore, the on-demand design of logic cells can be exploited as a good alternative for designing optimized digital circuits since it presents fewer transistors in general in comparison with traditional approaches.

$$f = a \cdot b + a \cdot c \cdot e + d \cdot e + b \cdot c \cdot d \tag{5}$$

Considering the gate layout generation, usually a full-custom approach is taken for the design of cell libraries. Since the number of gates available in a standard cell library is limited and most of the gates are composed of few transistors, the design task is feasible by hand. However, if we consider an on-demand project, where complex Boolean functions can be extracted from any node of the digital system and with gates containing a larger number of transistors, the handcraft design cannot be satisfactory since it is a process directly related to the complexity of the internal placement, routing, and compaction procedures. In this scenario, an automated approach may lead to optimized solutions quicker (ZIESEMER JR, 2014).

Nowadays, automatic layout generation tools present results comparable to hand-made designs (ZIESEMER; REIS, 2015; KARMAZIN; OTERO; MANOHAR, 2013). AS-



(a) (b)

Figure 3 – Layouts of the complex gates presented in Figure 2 generated through ASTRAN. (a) Layout of the Figure 2 (a). (b) Layout of the Figure 2 (b).

TRAN (ZIESEMER; REIS, 2015), considered the state of the art tool for this purpose, is an academic open-source solution that handle circuits without any topological constraints (an important feature for gates produced by graph-based methodologies, as mentioned before). The tool is composed of four main stages: folding, placement, routing, and compaction. Improvements in each stage lead to better solutions and quicker processes to achieve them. In order to illustrate the quality of the solutions produced through ASTRAN, Figure 3 (a) and (b) presents the circuits generated for the transistor networks shown in Figure 2 (a) and (b), respectively.

Therefore, this chapter presents a SAT-based algorithm for the transistor placement task in the automatic cell implementation flow. The proposed method was integrated with ASTRAN for assessing its quality in comparison with the original placement procedure of this tool and for validation purposes.

3.1 Chapter Organization

The sections in this chapter are organized as follows: Section 3.2 reviews some important concepts for the fully understanding of the synthesis process around the SCCG-based design; Section 3.3 introduces the related work; Section 3.4 presents a brief overview of the proposed approach; Section 3.5 formalizes the proposed methodology; Section 3.6 presents the experiments conducted and the obtained results; finally, Section 3.7 concludes this chapter.

3.2 Preliminaries

The design of SCCGs relies on automatic methodologies not only because of the large number of gates in a digital system, such as what occurs in the standard cell approach, but also because of the fact that each circuit need to be produced on-demand.

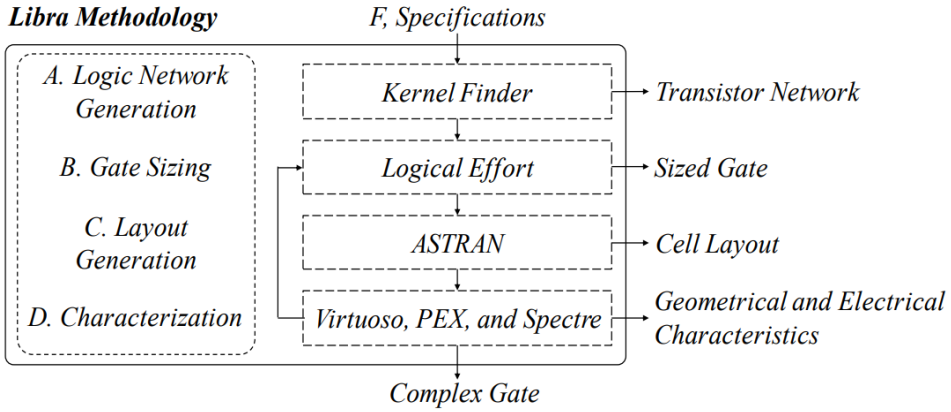


Figure 4 – Design of SCCGs through the Libra methodology.

In this scenario, the Libra methodology (CARDOSO et al., 2018) is the state of the art methodology for producing SCCGs. It can be divided in four main stages: logic network generation, responsible for producing the transistor network that implements the Boolean function desired, gate sizing, which computes the size of each transistor of the network, layout generation, the module that produces the layout of the cell, and, finally, the characterization, responsible for the verification and validation of the solution produced. Figure 4 illustrates the Libra methodology along with the tools and outputs associated with each part of the design flow.

The next subsections will detail the Libra methodology in order to illustrate the SCCG-based cell design. This way, we can understand more accurately the contribution of the placement method proposed in this thesis as well as its function in the SCCG design flow.

3.2.1 Logic Network Generation

Receiving as input a Boolean function, this procedure is responsible for delivering an optimized transistor arrangement obtained through Kernel Finder (KF) (POSSANI et al., 2016), the state of the art methodology for network design.

Since it is possible to obtain different solutions from divergent strategies of logic optimization in KF, Libra proposes an approach for transistor network generation which not only takes into account the number of transistors in the solution, but also the topology of the cell. This is due to the fact that non-planar structures may lead to vertical mismatches, increasing the number of columns necessary to implement the cell (CARDOSO et al., 2017).

Considering this, a method based on the KF tool was proposed in the Libra methodology. This method is presented in Algorithm 1 and it starts receiving an input Boolean function F in its irredundant sum-of-products (ISOP) form. From the input function in its direct and complementary polarity (F and $!F$, respectively) the pull-up *PU* and pull-down *PD* plans are generated via Kernel Finder. Right after that, *PU* and *PD* are tested

Algorithm 1: Pseudocode for logic network generation proposed by Libra

```
input  : Boolean function F
output: pull-up circuit PU, pull-down circuit PD
PU ← KernelFinder(¬F)
PD ← KernelFinder(F)
if isPlanar (PU) and isPlanar (PD) then
    if PD.size < PU.size then
        PD ← Dual (PU)
    else
        PU ← Dual (PD)
if isPlanar (PU) xor isPlanar (PD) then
    if isPlanar (PU) then
        PD ← Dual (PU)
    else
        PU ← Dual (PD)
return PU ∪ PD
```

concerning its topological aspects. If both plans are planar, then the dual network of the plan with fewer transistors is generated. In the case of a singular non-planar plan, the algorithm generates the dual network from that dual arrangement, regardless of the number of transistors on each plan. Finally, the method returns the arrangement composed by the logic plans *PU* and *PD*.

3.2.2 Gate Sizing

For the gate sizing, Libra implements the Logical Effort method (LE) due to its flexibility to handle different transistor topologies, such as non-series-parallel, including networks with a lack of duality and planarity[1]. Logical Effort is a well-known methodology that can be used to estimates the cell delay, as well as to define the transistors width. The procedure consists in computing the sizing of the devices of a complex gate in order to achieve the same current drive strength of a reference inverter. To perform this, the algorithm considers the transistor stacks for every path, where the size of the transistors is obtained taking into account its critical path, i.e., the largest transistor chain connecting the power lines to the output. Algorithm 2 illustrates the LE applied

[1]For computing the method we considered an output load of four minimal inverters (FO4)

Algorithm 2: Pseudocode for complex gate sizing through Libra

```
input  : pull-up circuit PD, pull-down circuit PD, λ
output: sized pull-up circuit PUs, sized pull-down circuit PDs
for transistor t ∈ PU do
    t.width ← ComputeCriticalPath (t, PU) * λ
    PUs ← PUs + t
for transistor t ∈ PD do
    t.width ← ComputeCriticalPath (t, PD)
    PDs ← PDs + t
return PUs ∪ PDs
```

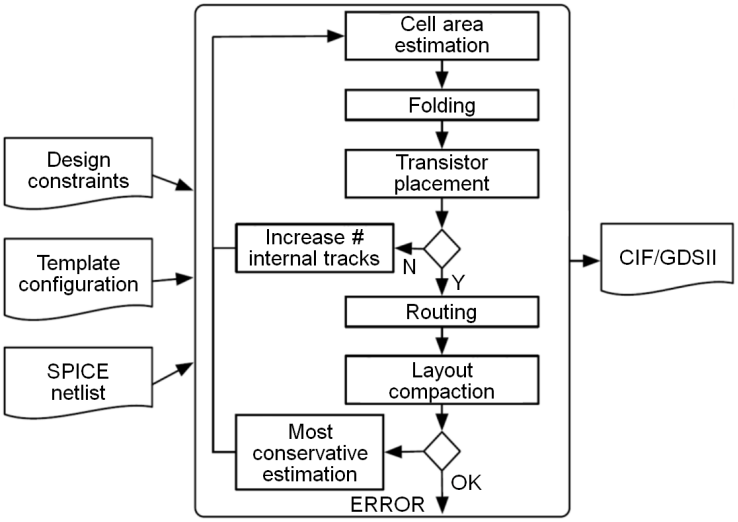


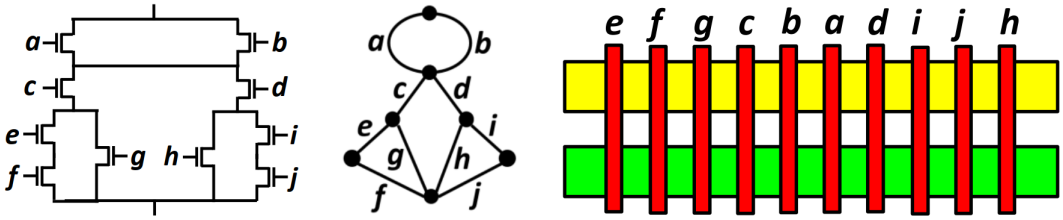
Figure 5 – ASTRAN cell generation.
Source: adapted from (ZIESEMER JR, 2014).

to the complex gate sizing problem. The algorithm receives as input the logic plans *PU* and *PD*, both computed through Algorithm 1, and λ, the ratio between the widths of the PMOS and NMOS transistors of the reference inverter. Then, the critical path of each transistor is calculated. Relative to the switches of the *PU* plan, the critical path is multiplied by λ. Finally, the plans are returned with all transistors properly sized.

3.2.3 Layout Design

The layout generation procedure of the Libra methodology is based on ASTRAN (which is an acronym for Automatic Synthesis of Transistor Networks) (ZIESEMER; REIS, 2015), a tool for automatic cell layout design. As already mentioned, ASTRAN is the state of the art academic open-source solution for physical synthesis of complex gates without topological constraints, i.e., it is possible to produce layouts from non-planar and non-dual arrangements, an important feature considering the SCCG-based design proposed in Libra.

The ASTRAN tool can be divided into four modules: folding, placement, routing, and compaction. The first module, folding, is responsible to break the transistors that surpass a specified row height, thus avoiding blank spaces which costs some layout area (SMANIOTTO et al., 2016). The placement procedure is responsible for defining the position of each transistor into the cell as detailed in Section 3.3.3. The routing task is done in order to find the paths that connect all the components of the layout through a modified version of the PathFinder routing tool (MCMURCHIE; EBELING, 1995). Finally, the last task is the compaction, where the blank spaces of the cells are minimized for the purpose of area optimization through an ILP approach. Figure 5 presents the automatic cell generation flow performed by ASTRAN where we can see all these modules.



(a) (b) (c)

Figure 6 – Transistor placement from an Euler path. (a) Transistor network. (b) Corresponding graph. (c) Pseudo-layout with no diffusion gaps.

3.2.4 Characterization

The last module of the Libra methodology for complex gate design consists of performing the cell characterization, gathering results concerning geometrical and electrical parameters of the generated layout. As the execution flow illustrated in Figure 4 shows, if the specifications are not met, then the gate sizing is repeated assuming relaxed parameters (changing the ratio λ or the weight associated with the critical path, for instance).

3.3 Related Work

This section presents some of the methodologies available in the literature for the placement of complex gate cells in transistor-based technologies.

3.3.1 Uehara et al. (UEHARA; VANCLEEMPUT, 1981)

Uehara et al. (UEHARA; VANCLEEMPUT, 1981) was the first method for transistor placement and it is still the base for some placement techniques used nowadays (for standard cell design, for instance). It is a graph-based methodology that relies on finding the Euler paths for producing single-row layouts.

The method receives as input a netlist with the transistors of the circuit. From that, a graph is created where each vertex represents a node of the netlist and each edge denotes a transistor of the netlist. With this graph created, the algorithm tries to find an Euler path, i.e., a path that traverses the whole graph visiting each edge exactly once. If an Euler path is found, then it is possible to create a layout in which all neighbors transistors share its diffusion areas. To accomplish that, the transistors must be placed in the same order they are visited in the graph.

To exemplify the algorithm, consider the transistor arrangement illustrated in Figure 6 (a), where an Euler path $E = (e, f, g, c, b, a, d, i, j, h)$ is present in (b), leading to the solution (c). Notice that, since an Euler path was found, then the layout does not present any diffusion break, minimizing the number of colums for placement.

In the case where there is no Euler path, then the algorithm splits the graph into

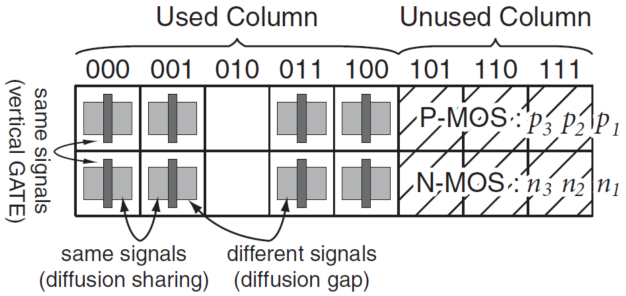


Figure 7 – Placement instance and layout style of (IIZUKA; IKEDA; ASADA, 2004).
Source: adapted from (IIZUKA; IKEDA; ASADA, 2004).

two subgraphs and the search is performed in each subgraph independently. This procedure is adopted until Euler paths are found in every subgraph. The circuits of each subgraph is then connected in the routing step of the layout design.

3.3.2 Iizuka et al. (IIZUKA; IKEDA; ASADA, 2004, 2005)

Iizuka et al. proposes two SAT-based methods for placement. The first method (IIZUKA; IKEDA; ASADA, 2004) focuses on dual topologies, while the second one (IIZUKA; IKEDA; ASADA, 2005) does not have any topological constraint but is implemented aiming at multi-row layouts. It is important to notice that the SAT-based method proposed in this thesis is able to deal with irregular topologies while implementing single-row layouts (an important feature for matching with ASTRAN). The next paragraphs will discuss each method.

In (IIZUKA; IKEDA; ASADA, 2004) it is introduced the idea of transforming the placement problem into a SAT formulation. The layout style employed is the single-row with fixed height, so the method only needs to deal with minimizing the width of the cell (i.e., the number of columns of the pseudo-layout). Figure 7 shows an instance of the Iizuka's placement. The method receives a spice netlist as input and generates its constraints based on the components of the circuit. Along with that, an intra-cell routing algorithm is defined so the layout placed can be tested if it is routable or not. Both the placement and the routing procedures are not focused on a particular technology node.

In (IIZUKA; IKEDA; ASADA, 2005) the first method is adapted to deal with non-regular topologies as the ones possibly obtained after folding, for instance. As described before, the resulting cell follows the multi-row layout style as presented in Figure 8. As in the previous method, the algorithm starts receiving an input netlist and the Boolean formulas are generated from this file based on the constraints designed. The algorithm also focuses on minimizing the width of the cells. The constraints are described in the form of pseudo-Boolean formulas so the authors can employ ILP solvers instead of SAT solvers (commercial solutions of ILP solvers are powerful enough to deal with the complex ILP instances generated through the method).

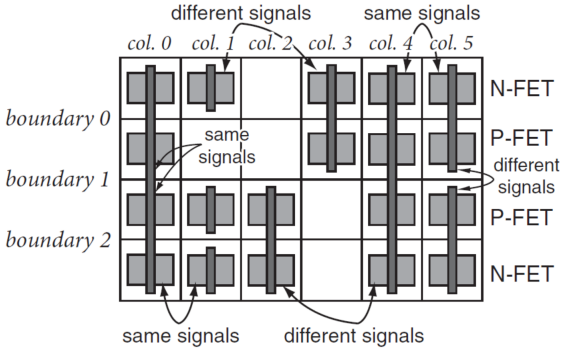


Figure 8 – Placement instance and layout style of (IIZUKA; IKEDA; ASADA, 2005).
Source: adapted from (IIZUKA; IKEDA; ASADA, 2005).

3.3.3 ASTRAN Placement (ZIESEMER; REIS, 2015)

Since the placement method presented in this thesis focuses on implementing SC-CGs using ASTRAN's infrastructure for dealing with the other design tasks, it is useful to have a deeper understanding of the transistor placement method implemented into the ASTRAN tool. Along with that, the assessments presented in the results section (Section 4.6) are done by comparing the solutions provided by ASTRAN's original tool (i.e., with its original placement method) and the solutions generated using the SAT-based methodology proposed.

ASTRAN implements a transistor placement through threshold accepting (TA) (DUECK; SCHEUER, 1990), a meta-heuristic similar to the well-known simulated annealing. Figure 10 illustrates the TA algorithm implemented. This way, local minimums can be achieved especially for larger cell instances.

The initial placement is generated randomly where a valid solution for the circuit is computed. Along with that, the perturbation function moves a set of connected transistors. There are two types of movements with the same chance of being executed: the first one consists of shifting a subset of transistors, while the second inverts the transistor ordering and orientation inside this subset (i.e., the subset is mirrored). Notice that the size of the subset is also randomly selected (ZIESEMER; REIS, 2015). Figure 9 illustrates the perturbation function.

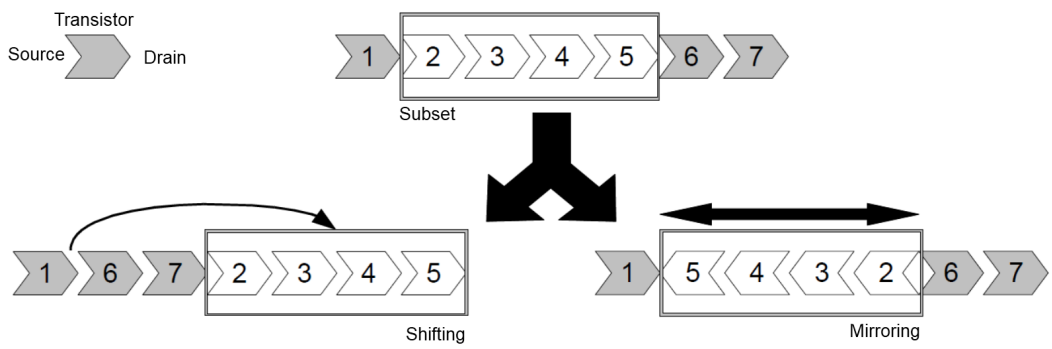


Figure 9 – Illustration of the perturbation function implemented into ASTRAN placement.
Source: adapted from (ZIESEMER JR, 2014).

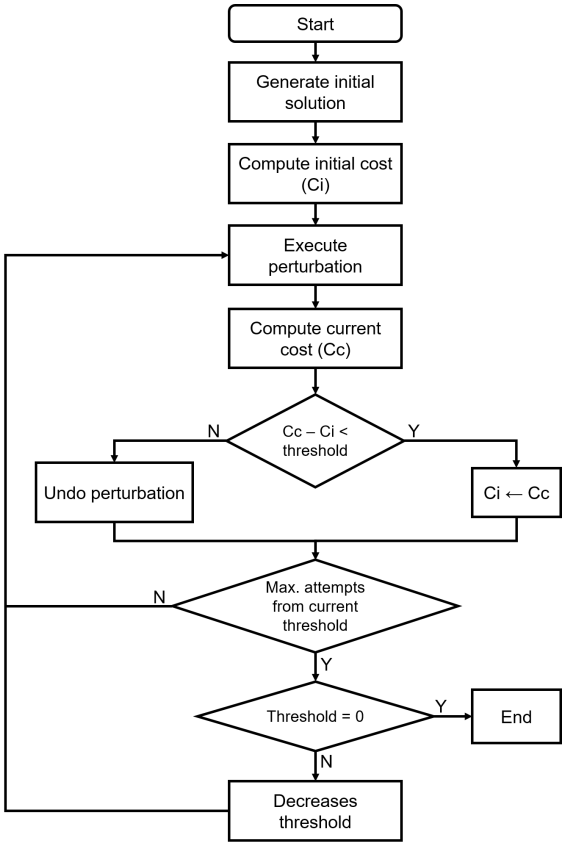


Figure 10 – Threshold accepting-based placement implemented into ASTRAN.
Source: adapted from (ZIESEMER JR, 2014).

Finally, the cost function f shown in equation 6 is defined

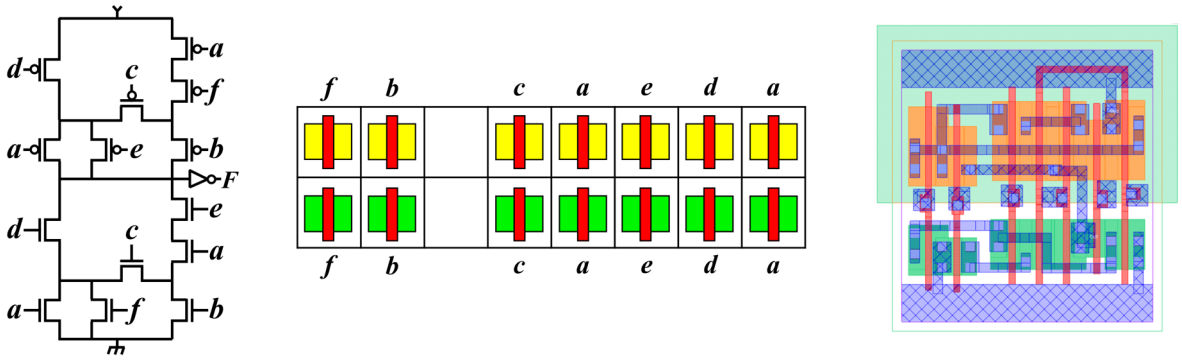
$$f = 100 * (4W_{gm} + 4W_{md} + 3W_c + 2W_g + W_{wl}) + W_{ld}, \tag{6}$$

where the following geometrical parameters are taken into account:

- W_{gm}: weight related to the misalignment of polysilicon gates in different plans;
- W_{md}: weight related to the density of the connections of the cell;
- W_c: weight related to the cell width;
- W_g: weight related to the number of diffusion gaps;
- W_{wl}: weight related to the total length of the connections;
- W_{ld}: weight related to the local routing density.

3.4 Method Overview

This section presents an overview of the proposed placement approach for SCCG design. As the Section 3.5 details, there are four constraints which must be satisfied



(a) (b) (c)

Figure 11 – Example of a transistor placement from a netlist. (a) Transistor netlist. (b) Pseudo-layout with transistors placed through the proposed method. (c) Final layout generated by ASTRAN.

for a placement to be valid. The constraints are:

- Transistor allocation constraint ($\mathcal{A}$): each transistor must be allocated at exactly one column;
- Transistor overlapping constraint ($\mathcal{O}$): each position of the layout must contain at most one transistor;
- Diffusion sharing constraint ($\mathcal{S}$): two transistors positioned laterally next to each other must share a node in the netlist;
- Gate alignment constraint ($\mathcal{G}$): two transistors positioned vertically next to each other must share the gate signal in the netlist.

In other words, a valid placement $\mathcal{P}$ is possible if and only if $\mathcal{P}$ is satisfiable such that

$$\mathcal{P} \equiv \mathcal{A} \wedge \mathcal{O} \wedge \mathcal{S} \wedge \mathcal{G}. \tag{7}$$

To illustrate the synthesis method, consider the example presented in Figure 11. In (a) we have the netlist which implements a Boolean function F. This netlist was obtained following the logic network generation step on Libra methodology. After sizing the transistors of this SCCG, the ASTRAN deals with the layout generation step: in (b) we have the solution for the placement - which was computed through our method, i.e., which fulfills all the constraints presented above -, whereas in (c) we illustrate the resulting cell (after routing and compaction).

The integration with ASTRAN can be summarized as illustrated in Figure 12, where we can see that we use the whole ASTRAN infrastructure but adopting our method as the core of the placement task.

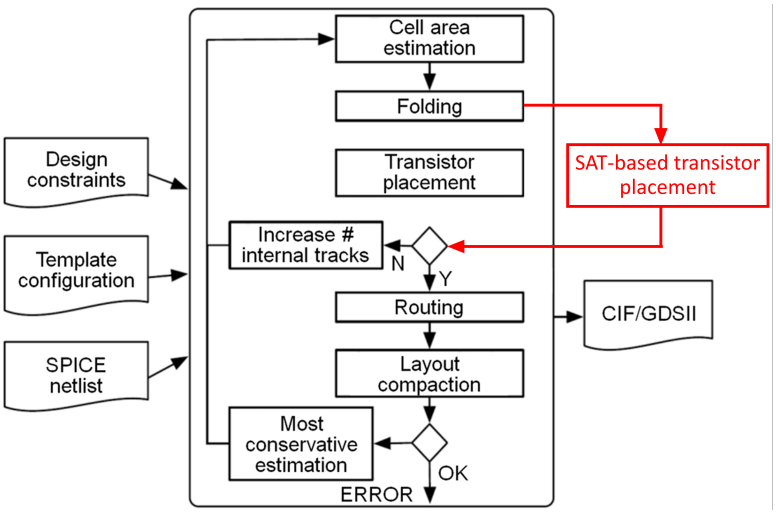


Figure 12 – Proposed modification (highlighted in red) on the ASTRAN cell generation flow. Source: adapted from (ZIESEMER JR, 2014).

3.5 SAT-based Modeling for SCCG Placement

This section is responsible for formalizing the proposed SAT-based placement method for SCCGs and it is organized as follows: Section 3.5.1 presents the input of the method; Section 3.5.2 defines the Boolean variables that represent the placement of each transistor in the circuit layout; Section 3.5.3 presents the formulas that model the design constraints; finally, Section 3.5.4 describes the proposed algorithm.

3.5.1 Input

Let N be the netlist illustrated in Figure 13 (a) represented as a hypergraph $N = (V, E)$, such that V is the set of vertices, representing the nodes of the netlist, and E is the set of hyperedges, representing the transistors of the netlist, as shown in Figure 13 (b). Thus, we define two disjoint subsets, one for each logic plain: the pull-up transistors $P = \{p_0, p_1, p_2, ..., p_a\}$ and the pull-down transistors $N = \{n_0, n_1, n_2, ..., n_b\}$, such that $E = P \cup N$.

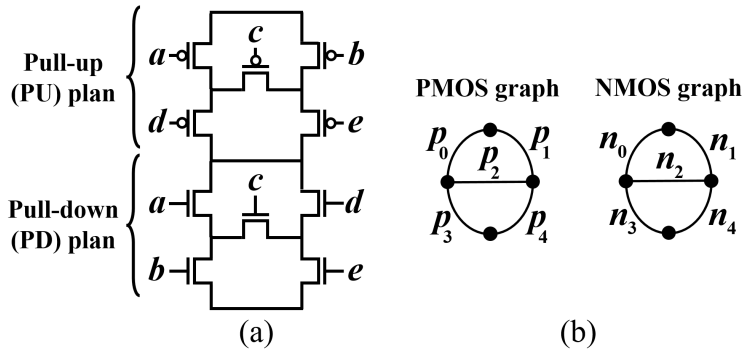


Figure 13 – Example of an input netlist (a) and its equivalent graphs (b).

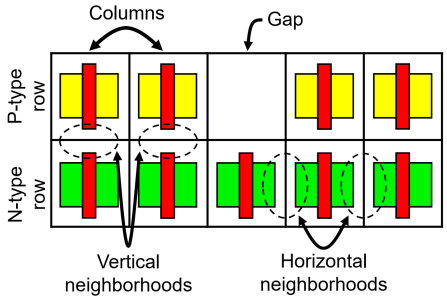


Figure 14 – Layout style adopted in our methodology highlighting some cell characteristics.

3.5.2 Boolean Variables

Before defining the variables of our formulation it is necessary to present the structure where the transistors will be placed. As described before, we adopt the single-row layout style to match with the cell created through ASTRAN. Thus, the pseudo-layout structure is a matrix-like grid with two rows, one for each logic plan (i.e., P-type row and N-type row), and several columns. Figure 14 illustrates this structure.

In this scenario, we can define the variables of our formulation as follows: the Boolean variable $p_i(c)$ denotes a PMOS transistor p_i located at the column c in its respective row. Similarly to that, the variable $n_i(c)$ denotes that the NMOS transistor n_i is located at the column c in its row[2].

Another important variable used throughout this chapter refers to the orientation of the transistor in the cell. There are two possible orientations considering the layout design purposes: drain-gate-source (a) and source-gate-drain (b). This way, for a PMOS transistor p_i located at the column c we have $p_i(c, d)$ to represent the case (a) - drain on the left of gate and source - and $p_i(c, s)$ to represent the case (b) - source on the left of gate and drain.

Thus, we can derive that

$$p_i(c) \iff exactly(1, \{p_i(c, d), p_i(c, s)\}). \tag{8}$$

3.5.2.1 Table of Variables

Table 1 summarizes the set of variables used in our method.

3.5.2.2 Number of Variables

The number of variables $\#V$ created through our method is given by

$$\#V = 3 * \#cols * (\#P + \#N), \tag{9}$$

[2]From now on, all the definitions will be presented in regard to the PMOS row since it is similar to the NMOS area.

Table 1 – Table of variables used on the SAT-based method for the placement of SCCGs.

Variables	Description
$p_i(c)$	PMOS transistor p_i is placed in column c.
$p_i(c,d)$	PMOS transistor p_i is placed in column c with the drain located on the left.
$p_i(c,s)$	PMOS transistor p_i is placed in column c with the source located on the left.
$n_i(c)$	NMOS transistor n_i is placed in column c.
$n_i(c,d)$	NMOS transistor n_i is placed in column c with the drain located on the left.
$n_i(c,s)$	NMOS transistor n_i is placed in column c with the source located on the left.

where $\#cols$ denotes the number of columns in the current iteration of the algorithm, $\#P$ is the number of PMOS transistors on the netlist, and $\#N$ is the number of NMOS transistors on the input.

3.5.3 Design Constraints

This section presents the constraints[3] for the placement of SCCGs cells through the proposed approach. All the constraints were meant to work with the 65nm and 45nm technology nodes, so it comprises all the design rules of these technologies considering the placement of the transistors. Along with that, we take into account the placement strategy described in (CARDOSO et al., 2017) that disallows the gate vertical mismatching. Finally, we present the restrictions in terms of a single transistor and a single column for simplicity reasons.

a. *Transistor allocation constraint:* each transistor must be allocated exclusively in one location into the cell. The formula that implements this constraint is

$$p_i(c) \implies \bigwedge_{c \neq c'} \neg p_i(c'). \tag{10}$$

b. *Transistor overlapping constraint:* each cell grid must contain at most one transistor, i.e., it should have exactly one transistor or a gap. Thus, we can derive

$$\bigwedge_{c=1,...,C} atmost(1, \{p_0(c), p_1(c), p_2(c), ...\}), \tag{11}$$

such that C is the number of columns in the current iteration.

c. *Diffusion sharing constraint:* all transistors must have their right neighbor with equivalent terminal value (except for those placed in the right boundary of the layout) or must be followed by a gap. In other words, if a transistor is followed by another, then

[3] Appendix A presents an example of how the following constraints are applied.

they must share a common node in the netlist. This is implemented through

$$\begin{aligned} p_i(c,d) &\implies \neg p_j(c+1,d), \text{if } s(p_i) \neq d(p_j), \\ p_i(c,d) &\implies \neg p_j(c+1,s), \text{if } s(p_i) \neq s(p_j), \\ p_i(c,s) &\implies \neg p_j(c+1,d), \text{if } d(p_i) \neq d(p_j), \\ p_i(c,s) &\implies \neg p_j(c+1,s), \text{if } d(p_i) \neq s(p_j), \end{aligned} \tag{12}$$

where $c \in \{1, 2, ..., C-1\}$, such that C is the number of columns in the current iteration, $s(p_x)$ is a function that returns the node connected to the source terminal of p_x accordingly with the input netlist and $d(p_x)$ is a function that returns the node connected to the drain terminal of p_x also accordingly with the input netlist.

d. *Gate alignment constraint:* all P and N-type transistors placed in the same column (in their respective rows) must share their gate signals. To implement this constraint we can define

$$p_i(c) \implies \neg n_j(c), \text{if } g(p_i) \neq g(n_j), \tag{13}$$

such that $g(p_x)$ is a function that returns the node connected to the source terminal of p_x accordingly with the input netlist (and similarly to $g(n_x)$ and n_x).

3.5.4 Algorithm

This section presents the proposed method (Algorithm 3) for the placement of SCCGs cells. Notice that the algorithm is based on the variables and constraints defined in the previous section of this chapter[4].

Algorithm 3: SAT-based pseudo-code for the placement of SCCGs

```
input  : input netlist N
output: pseudo-layout of the SCCG layout
C ← GetColumnsLowerBound(N)
while True do
    V ← CreateBooleanVariables(N, C)
    placement ← CreateClauses(N, C, V)
    SAT ← Solver(placement)
    if SAT ≠ False then
        layout ← Translate (SAT)
        return layout
    C ← C+1
```

The algorithm receives as its input the netlist *N* in SPICE format and computes the minimal number of columns C (lower bound) as described in (CORTADELLA, 2013) through the function `GetColumnsLowerBound(`N`)`. Following this, the

[4] It is important to notice that the formulas presented in the previous section are not in CNF format adopted for the SAT solvers. Thus, the Tseitin tranformation procedure (TSEITIN, 1983) is taken in order to convert all the formulas to the CNF format.

Boolean variables V and the clauses $placement$ are created through the methods `CreateBooleanVariables(`N, C`)` and `CreateClauses(`N, C, V`)`, respectively, where the last returns the conjunction of the formulas proposed (as presented in Section 3.4) in conjunctive normal form (CNF). In case of satisfiability - identified by the SAT solver through function `Solver(`$placement$`)` -, the satisfiable assignment of the Boolean variables is translated to its pseudo-layout form. However, if the formula is not satisfiable, then one column is added to each logic plan and the process of creating the variables, the clauses, and testing its satisfiability is repeated.

3.6 Experiments

The experiments presented in this section aim to assess the proposed method in comparison with the original ASTRAN placement procedure (reviewed in Section 3.3.3). It is reasonable to make this comparison since both rely on the same design rules (45nm and 65nm), layout style, and environment of the ASTRAN cell generator flow (i.e., other tasks - like routing and compaction - which directly impact the design of the cells are the same for both placement procedures). Considering this, two experiments were conducted: the first with the goal of assessing the placement methodology regarding the number of columns in the pseudo-layout and the second to assess the quality of the full design (in 65nm) in terms of cell area, wirelength, and number of contacts.

3.6.1 Assessment of the Number of Columns

The first experiment consists of measuring the number of columns in the pseudo-layouts produced through the SAT-based placement method proposed and the threshold accepting-based method from ASTRAN. In this scenario, the benchmarks used in this assessment are the following: the 53 NSP handmade networks (53NSP) (LOGICS, 2012) and a subset of the 4-input P-class (P4) (CORREIA et al., 2001). While the former is composed of 53 handcrafted non-series-parallel gates containing from 10 to 14 transistors, the latter is a subset of 150 cells that implements the functions defined in P4. To select the networks to be part of the P4 subset, we follow the methodology below:

- Firstly, we generate the transistor network for each instance of the P4 benchmark (comprising a total of 3982 functions) following the logic generation step of (CARDOSO et al., 2018) (detailed in Section 4);
- After this, we classified the networks based on the number of transistors on the complex gate. Twelve classes were created with cells ranging from 2 to 24 transistors;

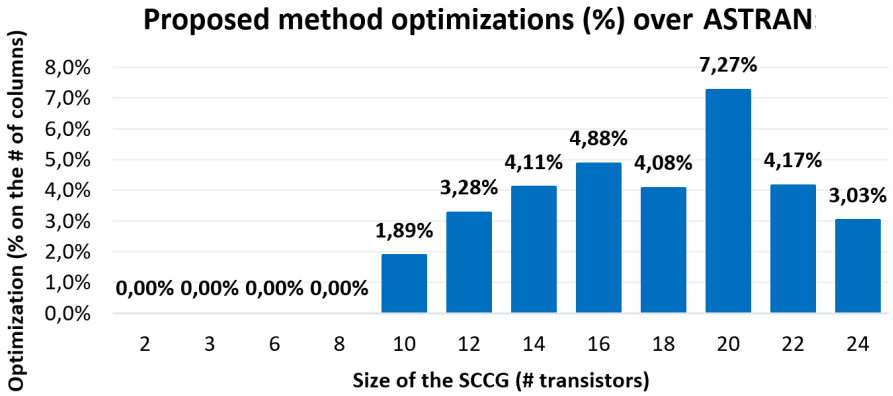


Figure 15 – Optimizations in the number of columns of the pseudo-layout.

- Finally, from each one of the twelve catalogs, we randomly selected 15 cells[5] to be implemented by both placement approaches, this way we achieved a homogeneous distribution regarding the size of the netlists. A total of 150 cells of the P4 benchmark were assessed.

The result concerning number of columns for the gates of the the 53NSP benchmark is the following:

- The cells F5, F10, F17, F19, F25, F26, F31, F32, F33, F39, F40, F41, and F43 generated through the proposed method has presented one less column in comparison with the ASTRAN solution;
- The cell F44 has presented two less columns in comparison with the ASTRAN solution;
- The other 39 cells has the same number of columns for both design approach.

The second result is presented in Figure 15. This chart illustrates the optimizations (on average) of the SAT-based solutions over the ASTRAN instances for each class of arrangements in respect to the number of columns of the pseudo-layout. As presented by the graph, the proposed method was able to deliver optimizations up to 7.27% for the SCCGs with 20 transistors regarding this parameter. For smaller instances (up to 8 transistors), the TA approach is able to find global minimal solutions, so no optimizations were obtained through the proposed approach.

Finally, Table 2 summarizes the results of this first assessment, where a frequency table illustrates the differences in the number of columns of the layouts assuming ASTRAN original solution as the reference (i.e., negative values means that the proposed method was able to find solutions with fewer columns). We can notice that, for the 53NSP catalog, 73.6% of the cells presented the same number of columns, while

[5]Except for the SCCGs with 2 and 4 transistors, where all 8 cells were tested, and the class of gates with 22 transistors, where all 13 cells were implemented.

Table 2 – Frequency table of the difference on the number of columns of the layout considering the ASTRAN placement as reference.

Difference	Benchmarks	
	53NSP.	P4
0	39 (73.6%)	116 (77.3%)
-1	13 (24.5%)	25 (16.7%)
-2	1 (1.9%)	7 (4.7%)
-3	0 (0.0%)	2 (1.3%)

26.4% of the pseudo-layouts reported optimizations in this aspect. Regarding the P4 benchmark, 77.3% of the solutions presented the same number of columns, while 22.7% reported reductions on this attribute.

3.6.2 Assessment of the Layout Geometries

The second experiment was conducted under the 53NSP catalog by generating the layouts through the complete flow provided by the ASTRAN tool. In this scenario, we employ the original routing and compaction procedures of ASTRAN for both versions of the layouts generated in order to have a fair comparison. Moreover, the STMicroelectronics 65nm node was adopted for the synthesis with all the ASTRAN's options in default.

The following parameters were extracted from the layouts: area, routing wirelength (considering polysilicon and metal 1), and number of contacts. Besides that, the execution times of the placement procedures were also collected. The chart illustrated in Figure 16 summarizes the optimizations (blue bars) and overheads (red bars) obtained, where we take as reference the original ASTRAN approach. As we can notice, an optimization of 2.1% was obtained considering layout area; regarding wirelength, an overhead of 0.1% was observed; finally, related to the number of contacts, the SAT-based solution presented a 6.0% optimization.

Along with that, the proposed approach took 0.49s on average to compute the placement, while the original ASTRAN procedure spent 11.42s on average for the same scenarios, thus corresponding to a 23.3x speedup. Furthermore, standard deviations of the optimizations in area, wirelength, number of contacts, and runtime are 3.8%, 8.3%, 6.1%, and 4.5%, respectively. Through the analysis of the charts presented in Figure 16 along with the corresponding standard deviation of each parameter, we can notice a small dispersion of the data, i.e, the results are uniform for most of the cases.

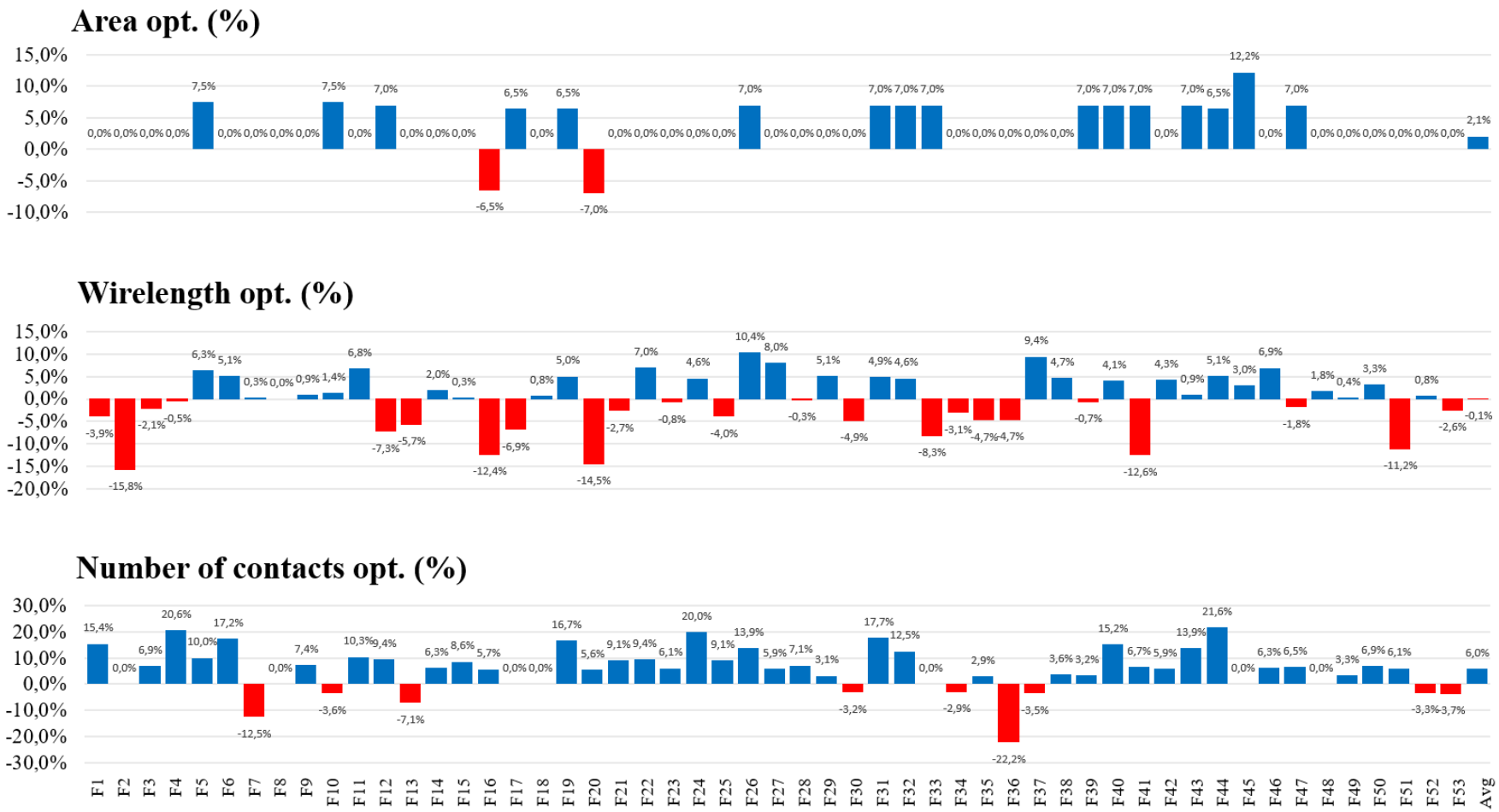


Figure 16 – Optimizations and overheads of the cells generated through ASTRAN with the proposed and original placement procedures.

Finally, even optimizing the number of columns for placement (as previously shown in Table 2), the layout area of two cells (F16 and F20) presented an overhead compared to the original ASTRAN version. This is due to the routing step, which uses extra columns to perform the internal connections of the gate. A routing algorithm integrated with the placement procedure must overcome this problem. This can also provide better results in terms of wirelength.

3.7 Chapter Conclusions

In this chapter, we presented a new SAT-based placement method for SCCGs. A set of Boolean variables and formulas were defined in order to model the design rules of the 65nm and 45nm technology nodes. An integration with ASTRAN was done so the tool was the responsible for dealing with routing and compaction processes, validating the proposed method in a real design environment.

The results of the experiments have shown that the SAT-based methodology was able to optimize the cells in comparison with a threshold accepting method implemented into ASTRAN (i.e., we also provide a valuable optimization for this open-source tool). Two assessments were done: the first comparing the number of columns in a pseudo-layout, where we can notice that the proposed approach delivers cells with a less or equal number of columns; the second assessment compares some cell geometrical parameters, where the proposed method also shows good results on average.

Therefore, the results showed the potential to use a similar approach for designing placement methods for SCCG cells in different technologies since the SAT solvers provide good results (not relying on local minimum or greedy algorithms) in a reasonable computing time.

4 QUANTUM-DOT CELLULAR AUTOMATA

Transistor shrinking, the fundamental process that enabled the well-known Moore's law to be valid for such a long time, may be coming to a halt. While devices reach their limits, the manufacturing problems in advanced lithographies make the design of new technology nodes a highly cost process. At the same time, reliability and power issues become important challenges in the design of modern digital circuits and systems. To overcome these obstacles, various emerging technologies have been studied in the last decades, such as single-electron transistors, molecular electronics, optical computing, carbon nanotube transistors, quantum-dot cellular automata, nanomagnetic logic, among others.

In this scenario, the class of the field-coupled nanocomputing (ANDERSON; BHANJA, 2014) technologies turns up to be a promising alternative to the conventional paradigm based on electron-charged devices such as the CMOS and FinFET. The FCN family handles the computation through a fundamentally different approach: the binary information is represented in terms of the polarity or magnetization of cells and can be propagated to adjacent devices using repelling forces of local fields (LENT; TOUGAW, 1997). This way, without the data transport relying on electrical current flows, the energy levels involved in data processing are orders of magnitude lower compared to the traditional transistor-based paradigm (LENT; SNIDER, 2014).

The first proposed device architecture of the FCN family was the quantum-dot cellular automata (QCA) originally presented in (LENT; TOUGAW, 1997). The QCA technology consists of bistable cells locally connected through field-effect forces which can be organized to perform logic operations (CAMPOS et al., 2016). It is expected that QCA circuits composed of nanodevices of length in the range of 2-18nm (LIU; O'NEILL; SWARTZLANDER, 2013) could ensure high clock frequencies (in the THz range) (KIM; WU; KARRI, 2006) and ultra-low power consumption (ANDERSON; BHANJA, 2014; LENT; SNIDER, 2014; TIMLER; LENT, 2002).

Another important aspect of the QCA technology is the clock. The clocking system is responsible for controlling the flow of information across the design and ensuring the correct functioning of the circuit. In this scenario, clocking schemes such

as USE (CAMPOS et al., 2016), RES (GOSWAMI et al., 2019) and 2DDWave (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008), among others, were proposed to enable the development of placement and routing algorithms that satisfy the design constraints of the technology. Besides that, clocking schemes are important tools for providing scalability for the QCA systems, a crucial feature for adopting this emerging technology in the future.

This chapter presents an automated method for placement and routing of synchronized QCA circuits relying on the USE clocking scheme, the state of the art scheme for QCA design. Such as in the previous chapter, the method is posed as a Boolean satisfiability problem aiming to obtain area-optimized solutions. As a byproduct, the proposed method also presents good results in terms of latency due to the correlation between this parameter and the area of the QCA circuit. The methodology presented along this chapter can be applied for several purposes: for the design of QCA cell libraries, for the on-the-fly synthesis of QCA circuits, for the integration with scalable divide-and-conquer strategies, for the validation and comparison with new design automation methods, among others.

4.1 Chapter Organization

The sections in this chapter are organized as follows: Section 4.2 reviews some important concepts for the fully understanding of the QCA technology; Section 4.3 introduces some related work; Section 4.4 presents a basic overview of the proposed approach; Section 4.5 formalizes the proposed methodology; Section 4.6 presents the experiments conducted and the obtained results; finally, Section 4.7 concludes this chapter.

4.2 Preliminaries

Quantum-dot cellular automata (QCA) is an emerging technology from the family of the field-coupled nanocomputing (FCN) (ANDERSON; BHANJA, 2014). Differing from the traditional transistor-based approach, the FCN paradigm relies on store, transfer, and compute information through Coulomb interactions, i.e., attractive and repulsive electrostatic forces, that occur intra and inter devices. In this way, there is no electric current directly involved in the processes, which potentially leads to ultra-low power consumption (ANDERSON; BHANJA, 2014; LENT; SNIDER, 2014; TIMLER; LENT, 2002), an important feature considering the challenges of modern digital systems and applications. IoT, biomedicine, and wearables are some examples in which a low power consumption profile is desirable due to the limited energy supply. In this section, we review some basic concepts for a fully understanding of the QCA technology.

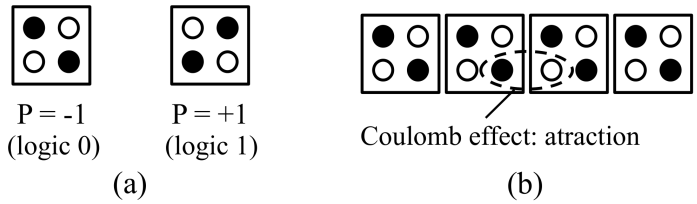


Figure 17 – QCA fundamentals. (a) QCA cells and polarization states. (b) A typical QCA wire highlighting the Coulomb effect between devices.

4.2.1 QCA Basics

The fundamental unit of QCA is a cell composed of four quantum dots that can confine an electric charge (LENT; TOUGAW, 1997; TIMLER; LENT, 2002). These dots are located in the corners of a square, as shown in Figure 17 (a). Along with that, two mobile electric charges are confined in this square such that they are able to tunnel between adjacent dots but unable to tunnel outside the cell due to the potential barrier of the frontier of the square. In this scenario, the Coulomb forces lead the system to a minimal energy state where the mobile charges are located in one of the diagonals of the QCA cell. Thus, binary information is encoded according to the polarization of the cell as shown in Figure 17 (a), where $P = -1$ and $P = +1$ represent the logic 0 and 1, respectively. The Coulomb effect which acts upon the QCA cells can be observed in Figure 17 (b), which shows a typical QCA wire highlighting the attraction between dots containing a mobile charge and a hole.

4.2.2 QCA Clocking System

The transition between these two polarized states should occur adiabatically since sudden changes could induce the system to a metastable configuration (LANDAUER, 1995). In this scenario, an external clock signal must be employed. The clock is also responsible for controlling the potential barriers of the cell. In this way, neighboring devices could interact, via the Coulomb phenomena, enabling the flow of information and the logic processing. Figure 18 shows the data transfer in a QCA wire.

The clock signal in QCA is composed of four phases:

1. Switch: this phase is responsible for slowly increasing the potential barrier of the cell (which avoids metastable states, as mentioned before), allowing the correct change in the polarization of the device accordingly to its neighbors;

2. Hold: this phase is responsible for keeping the potential barrier of the cell in a high state, thus avoiding the influence of neighboring devices and keeping its polarization. The release phase is responsible for slowly decreasing the potential barriers of the cell, leading to its depolarization;

3. Release: this phase is responsible for slowly decreasing the potential barriers of

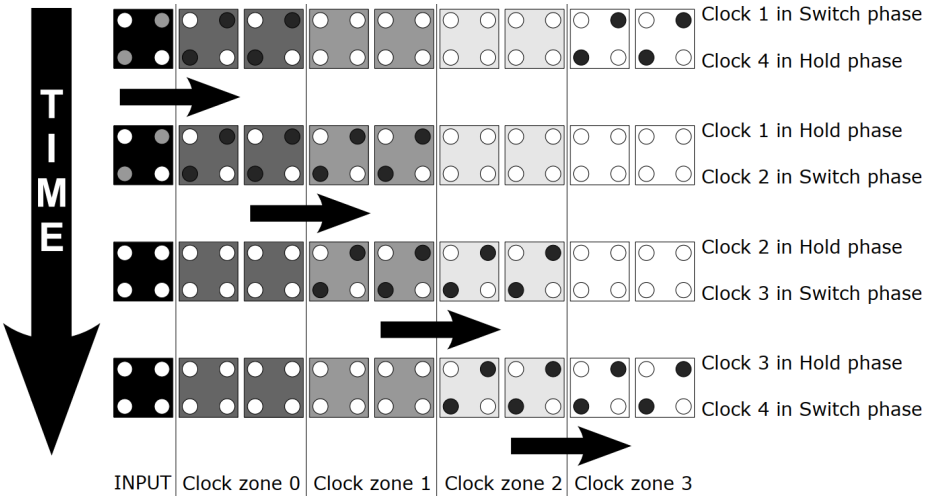


Figure 18 – Data transfer in a QCA wire.
Source: (CAMPOS, 2015)

the cell, leading to its depolarization;

4. Relax: this phase is responsible for keeping the potential barriers in a low state, maintaining the cell depolarized.

After the completion of the relax phase, the cell starts a new cycle, i.e., a clock cycle is composed of four clock phases. Regarding the clock signal generation, one of the main challenges for the adoption of the QCA technology, (ANDERSON; BHANJA, 2014) made a broad discussion on this topic.

4.2.3 QCA Clocking Schemes

There are several ways of designing QCA circuits, each one presenting its own advantages and drawbacks. One of the most adopted is the design based on clocking schemes. Although they produce solutions with overheads in area compared to the free (manual) approach, clocking schemes provide modularity and scalability. Additionally, it can be used as a regular structure amenable for design automation solutions of QCA layouts, which is a crucial aspect to potentially enable this technology in the future.

One of the most popular clocking schemes available in the literature is USE (CAMPOS et al., 2016). As other alternatives, such as (GOSWAMI et al., 2019) and (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008), USE proposes a design grid in which each cell located on a clock zone receives data from the neighbors located on a preceding zone and sends data to the neighbors located on a succeeding zone, thus guaranteeing a correct flow of information within the circuit. It is important to notice that USE differs from other approaches on how the clock zones are organized and the number of QCA devices located in each clock zone. Figure 19 illustrates the USE clocking scheme.

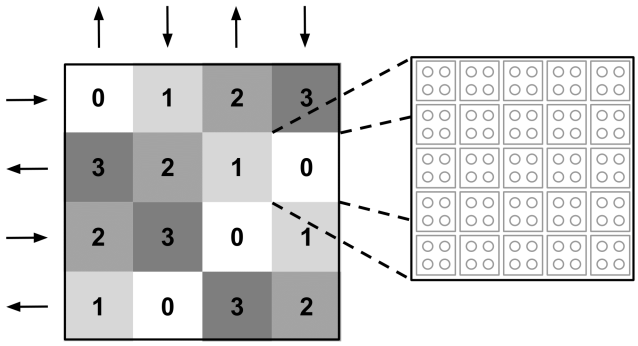


Figure 19 – The USE grid and a USE cell composed of a 5x5 QCA matrix.
Source: adapted from (CAMPOS et al., 2016).

4.2.4 QCA Cells

In order to be functional complete, it is necessary to define the set of gates which implements the Boolean operations 3-input majority (MAJ3) and negation (NOT). Along with that, all the logic structures produced through the proposed method, i.e., the gates and the wires, are built over the structure of a 5x5 USE cell as proposed in (CAMPOS et al., 2016) - detailed in Figure 19 and in the Section 4.2.3.

In this scenario, we adopt the cell library used in (FONTES et al., 2018), as well as two parallel routing tracks in the horizontal and vertical axis of each USE cell. The multiple routing tracks is useful for providing more flexibility for connecting different parts of the circuit. Notice that it is possible to have crossing wires as used in (CAMPOS et al., 2016). Figure 20 presents some examples of the 3-input majority gate (a), the inverter gate (b) and the routing wires (c) proposed in (FONTES et al., 2018) which serves as basis for composing the design constraints in our method.

4.3 Related Work

This section discusses some of the methodologies available in the literature for the automatic physical synthesis of QCA circuits.

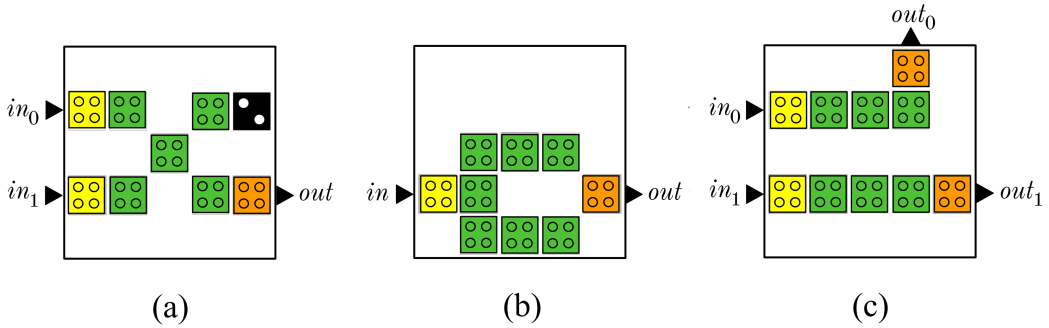


Figure 20 – Examples of QCA cells adopted in our model. (a) Majority-based gate implementing an *and* function. (b) Inverter gate. (c) Two wires in a USE cell.

4.3.1 Fiction Framework (WALTER et al., 2019)

Fiction is a framework for the physical design of field-coupled nanocomputing circuits with a focus on the QCA technology (WALTER et al., 2019). This tool supports different clocking schemes (including USE) and it is composed of two main modules: the first (WALTER et al., 2018) is based on Satisfiability Modulo Theories (SMT) and it is capable of handling small instances of the circuit, while the second (WALTER et al., 2019) relies on Orthogonal Graph Drawing (OGD) and provides scalability at the cost of some area. This section will focus on the SMT-based method.

As detailed in (WALTER et al., 2018), the method receives an input netlist and the design constraints (e.g., available area). Based on that, a symbolic formulation is defined to represent the problem and the design restrictions. This formulation is passed to an SMT solver responsible for checking the feasibility of the circuit.

The main difference between the aforementioned SMT-based method and the approach proposed in this thesis is that the design constraints defined in (WALTER et al., 2018) potentially lead to non-synchronized solutions where the frequency of data input on the circuit is dependent on the latency of each input. As discussed in (TORRES et al., 2018), this produces area-optimized circuits at the price of a smaller throughput and the need for controlling the frequency of data input, which is not convenient for creating cell libraries, for example. Along with that, (WALTER et al., 2018) supports single-wire cells, a design style that limits the routing paths available when compared to the multi-wire cell style adopted in the proposed approach.

4.3.2 Trindade et al. (TRINDADE et al., 2016)

This method consists of a greedy breadth-first approach and is one of the first automated solutions proposed for the physical design of QCA circuits with the USE clocking scheme.

As detailed in (TRINDADE et al., 2016), the algorithm receives a Directed Acyclic Graph (DAG) representing the input netlist. For each level of the DAG (defined a priori), all the nodes are simultaneously placed and routed. The placement is feasible if there is a free cell in the USE grid that can allocate all the nodes of a given level in a way that all of them have the same latency for the connections to the upper level. If there is no available USE grid cell for placement, then the latency is increased in one unit and the procedure restarts. The routing task is done similarly: if no routing paths are available for a given latency between levels, then the latency is increased and the process restarts.

The main aspects that differentiate (TRINDADE et al., 2016) and the proposed approach are: (1) the levels of the input graph are defined a priori, which already limits the search space; (2) the latency between levels is the same for each node located in these levels, thus having an impact in the area of the circuit, as it will be shown in the

experiments of this thesis; (3) it is a greedy approach, so local minima are often achieved; (4) only single-wire cells are allowed, limiting the routing paths on the circuit. On the other hand, it is important to notice that all the solutions produced by (TRINDADE et al., 2016) are synchronized, as the ones obtained with the proposed methodology. Because of this, a fair comparison is possible as later shown in the experiments of this thesis (Section 5.4.2).

4.3.3 Fontes et al. (FONTES et al., 2018)

This method consists of two steps: the first is responsible for the placement and routing of netlists containing a small number of gates, whereas the second connects the subcircuits generated in the first step to create the final circuit (providing scalability). The discussion will focus on the first step.

As described in (FONTES et al., 2018), the method is based on the approach presented in (TRINDADE et al., 2016). The algorithm receives an input netlist organized on different levels (defined a priori as well). The placement and routing procedures are responsible for assigning the locations of each element by traversing the input netlist in breadth-first order. After assigning an initial latency to each level, the gates are placed and routed in such a way that, if there are no available positions in the USE grid for all the cells in a given level, the latency is increased by one unit. The maximum latency between levels $l_{\max}$ is defined by $l_{\max} = \max(4, d)$, where d is the input depth. If the maximum latency of a given level is exceeded without finding any feasible solution, then the algorithm backtracks and the latency of the upper level is increased. The main difference between (FONTES et al., 2018) and (TRINDADE et al., 2016) relies on the fact that the first explores simultaneously multiple placement and routing solutions for a given level in the search-space tree in order to assess the impact in later decisions of the algorithm, thus choosing the path that results in a minimum-area circuit.

The aspects of (FONTES et al., 2018) that differ from the proposed methodology are: (1) the levels of the input netlist are defined a priori; (2) the latency between levels is the same for each node located in these levels; (3) it is a greedy algorithm, so local minima are frequently achieved; (4) the maximum latency between two connected nodes is limited by the depth of the input graph. However, as in (TRINDADE et al., 2016), the solutions obtained by (FONTES et al., 2018) are synchronized, hence being suitable for the comparisons with the proposed methodology (Section 5.4.2).

4.3.4 Formigoni et al. (FORMIGONI et al., 2021)

The method proposed in (FORMIGONI et al., 2021) is based on a multilevel graph partitioning algorithm.

"Firstly, several hypergraphs are generated, indexed from 0 to n, where n is the number of the input nodes. Then, a graph maximal matching algorithm is applied as

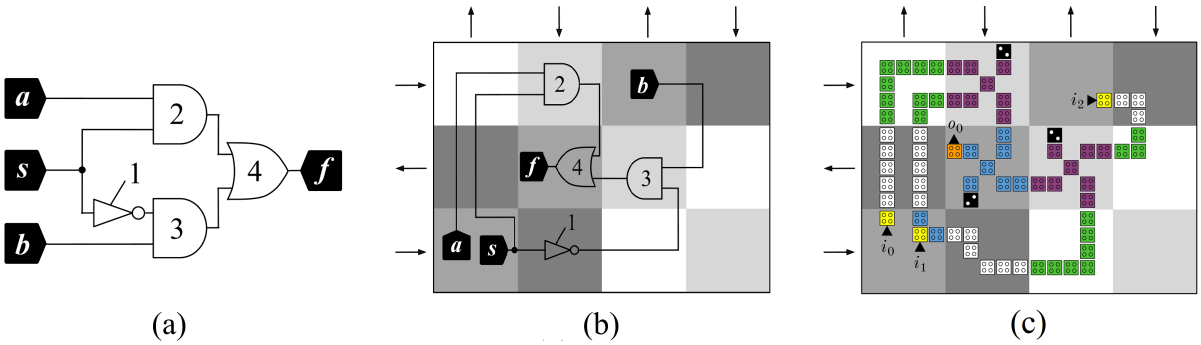


Figure 21 – Implementation of a 2:1 MUX in QCA. (a) Logic gates implementing the 2:1 MUX. (b) Gate-level solution obtained by the proposed methodology. (c) QCA circuit.

the criteria to collapse adjacent graph vertices. The algorithm repeats this process until it creates a hypergraph composed of a single vertex. For the second phase of the strategy, it is performed the placement of the hypergraph indexed by n, followed by on-grid uncoarsening for layout expansion. The algorithm generates the final layout when the process expands the original base graph. Area overhead translates to the number of generated hypergraphs, which the algorithm should minimize" (FORMIGONI et al., 2021).

Finally, the aspects of (FORMIGONI et al., 2021) that are divergent of the proposed methodology are: (1) the latency between levels is the same for each node located in these levels; (2) it is a greedy algorithm, so local minima can be obtained; As in (TRINDADE et al., 2016) and (FONTES et al., 2018), the solutions obtained by (FORMIGONI et al., 2021) are synchronized, so comparitons with the proposed methodology can be conducted (Section 5.4.2).

4.4 Method Overview

This section illustrates how our approach works. As we will see in Section 4.5.3, there are several constraints that must be satisfied for a QCA circuit to be valid (i.e., functional). These constraints are classified into three categories:

- Placement constraints (Section 4.5.3.1), responsible for defining valid positions for each gate;
- Synchronicity constraints (Section 4.5.3.2), responsible for ensuring the global synchronicity of the circuit and the local synchronicity of each individual gate;
- Routing constraints (Section 4.5.3.3), necessary for designing the connections between each gate.

We next describe the synthesis of the 2:1 multiplexer (2:1 MUX) circuit shown in Figure 21 (a), which results in the solution presented in (b) and (c).

4.4.1 Placement

The placement task is responsible for guaranteeing the following two conditions: (1) each gate, input, and output must be located in only one position (avoiding multiples instances of these components); (2) each grid cell should not exceeds the limit of one gate, two wires, or (exclusively) two inputs. Considering the solution previously presented in Figure 21 (b), we can notice that these two constraints are met.

4.4.2 Synchronization

The synchronization task is responsible for guaranteeing the following conditions: (1) each signal must traverse the circuit according to the directions imposed by the arrows located on the border of the grid; (2) all inputs must be assigned to the same clock phase (similarly for the outputs); (3) the signals arriving at a gate should have the same latency. Again, all these constraints are honored in Figure 21 (b).

4.4.3 Routing

The routing task guarantees the following conditions: (1) each input is followed by a wire or by the gates connected to this input; (2) each gate is preceded by its inputs (primary inputs of the circuit or other gates) or by a wire coming from these inputs; (3) each gate is followed by its outputs (primary outputs of the circuit or other gates) or by its own wire.

4.4.4 SAT-based Procedure

After defining the placement ($\mathcal{P}$), synchronization ($\mathcal{S}$), and routing ($\mathcal{R}$) constraints for a given USE grid, a Boolean formula

$$\mathcal{F} \equiv \mathcal{P} \wedge \mathcal{S} \wedge \mathcal{R} \tag{14}$$

is created such that, if $\mathcal{F}$ is satisfiable, then a valid solution is obtained. If unsatisfiable, then the search space is expanded (e.g., by rotating or flipping the USE grid, changing the USE grid configurations, or increasing the area) and $\mathcal{F}$ is redefined. The details of the algorithm are discussed in Section 4.5.4.

4.5 SAT-based Method for QCA Design

This section is responsible for the formalization of the QCA synthesis method and it is organized as follows: Section 4.5.1 presents the input of the method; Section 4.5.2 defines the Boolean variables that represent the devices and wires placed in the circuit layout; Section 4.5.3 presents the formulas that model the design constraints; finally, Section 4.5.4 describes the proposed algorithm.

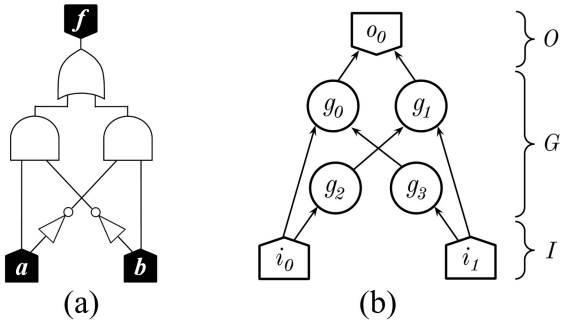


Figure 22 – A 2-input XOR gate (XOR2). (a) Gate-level netlist. (b) Corresponding graph.

4.5.1 Input

Let N be the netlist circuit represented as a hypergraph $N = (V, E)$, such that V is the set of vertices, representing the gates, and E is the set of hyperedges, representing the wires. Figure 22 depicts the hypergraph of a 2-input XOR gate (XOR2). Furthermore, the set of vertices of N can be partitioned into three disjoint subsets: input nodes $I = \{i_0, i_1, ..., i_n\}$, gate nodes $G = \{g_0, g_1, ..., g_m\}$, and output nodes $O = \{o_0, o_1, ..., o_p\}$, such that $V = I \cup G \cup O$.

4.5.2 Boolean Variables

There are two types of variables in our model: device variables and interface variables. These sets of variables are detailed below.

4.5.2.1 Device Variables

Device variables represent the devices on the circuit, i.e., the gates (majority-gates and inverters) and wires of the QCA design, as illustrated in Figure 20. These devices can be inputs, gates, or outputs.

To define the device variables used in our formulation, consider (x, y) as a pair of coordinates of the USE grid. Thus, the gate variable $g_i(x, y)$ denotes that the logic gate or wire represented by node g_i is in the location (x, y) of the grid. The same applies to the input variable $i_i(x, y)$ and the output variable $o_i(x, y)$[1].

Considering that device variables represent gates or wires, it is necessary to define driver cells and wire cells. Formally,

$$g_i(x, y) \iff g_{i(d)}(x, y) \vee g_{i(w)}(x, y), \tag{15}$$

where $g_{i(d)}$ and $g_{i(w)}$ represent a driver (i.e., an input, a majority gate, or an inverter) and a wire of g_i, respectively. Proposition (15) indicates that the presence of g_i in (x, y)

[1]For the sake of simplicity, the formulas in this thesis are defined in terms of the gate variables. However, it is important to notice that similar expressions are also defined for the input and output variables. All the exceptions for the previous statement are explicit in the text.

implies the presence of a driver or wire of g_i in the same location (and vice versa).

Since the phase of the cell at (x, y) in the USE grid is determined by the grid configuration, a new variable is necessary to represent the cycle each gate operates. Considering $C = \{0, 1, ..., cycles - 1\}$, where $c \in C$, we can represent driver and wire cells with their associated cycles c as $g^c_{i(d)}(x, y)$ and $g^c_{i(w)}(x, y)$, respectively. Formally, we have

$$g_{i(d)}(x, y) \iff \bigvee_{c \in C} g^c_{i(d)}(x, y), \tag{16}$$

$$g_{i(w)}(x, y) \iff \bigvee_{c \in C} g^c_{i(w)}(x, y), \tag{17}$$

$$g^c_i(x, y) \iff g^c_{i(d)}(x, y) \vee g^c_{i(w)}(x, y). \tag{18}$$

4.5.2.2 Interface Variables

Interface variables are responsible for representing the connections between the cells of the USE grid, i.e., the interface ports of each USE cell.

Let us consider the USE cell of the coordinates (x, y) represented in Figure 23. This cell contains two interfaces on each one of its frontiers (west, north, east, and south) as shown in the respective figure. Let us define $R = \{ws, wn, ne, nw\}$ as the set of port locations on the west and north side of the cell, where $r \in R$. Thus, we can represent an interface variable as $r(x, y)$. Notice that the interface variables of the eastern and southern frontiers are shared with the neighbors USE cells placed in the right and the bottom of the given (x, y) cell, respectively.

Since it is possible to have two different signals traversing a USE cell formed by a 5x5 structure of QCA devices (as Figure 20 (c) illustrates) it is necessary to identify which signal is linked to each port. This way, we can define a set of two signals $S = \{0, 1\}$, such that $s \in S$ and

$$r(x, y) \iff \bigvee_{s=0..1} r_{(s)}(x, y). \tag{19}$$

4.5.2.3 Variables Relationships

After defining the Boolean variables of our method, it is crucial to create the logical relationships between the device and interface variables in order to be able to define the design constraints.

a. *Driver variables and interface variables:* a driver cell $g_{i(d)}(x, y)$ containing n interfaces (considering $n = degree(g_i)$, i.e., the degree of the node g_i) should have at least n interface ports connected if $n = 1$ (inputs) and exactly n interface ports connected if $n = 2$ (inverters), $n = 3$ (and or or gates), or $n = 4$ (3-input majority gates).

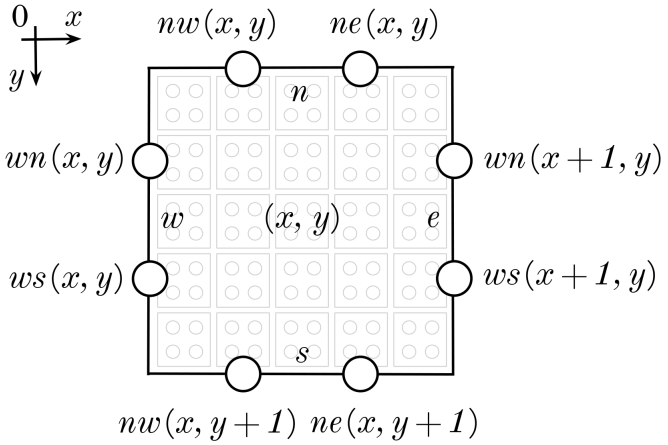


Figure 23 – Interfaces of a USE cell.

In order to formulate this relationship, let us assume $B(x,y) = \{ws(x,y), wn(x,y), nw(x,y), ne(x,y), ws(x+1,y), wn(x+1,y), nw(x,y+1), ne(x,y+1)\}$ as the set of interface variables containing the ports placed on the borders of the cell (x,y). This way, we can derive

$$i_{i(d)}(x,y) \implies atleast(1, B(x,y)), \tag{20}$$

for the case of inputs, and

$$g_{i(d)}(x,y) \implies exactly(n, B(x,y)), \tag{21}$$

for the case of gates and outputs.

b. *Wire variables and interface variables:* if a wire is placed in a USE cell, then at least two interface ports should be connected, i.e., two ports should share the same signal.

To formalize this, consider $B_{(s)}(x,y) = \{ws_{(s)}(x,y), wn_{(s)}(x,y), nw_{(s)}(x,y), ne_{(s)}(x,y), ws_{(s)}(x+1,y), wn_{(s)}(x+1,y), nw_{(s)}(x,y+1), ne_{(s)}(x,y+1)\}$ as a set similar to $B(x,y)$ previous defined but adding the signal information to each element.

Formally,

$$g_{i(w)}(x,y) \implies \bigvee_{s=0..1} atleast(2, B_{(s)}(x,y)). \tag{22}$$

4.5.2.4 Table of Variables

Table 3 summarizes the set of variables defined in our method[2].

[2]The input and output variables are omitted in Table 3 for simplicity reasons.

Table 3 – Table of variables used in the SAT-based method for QCA.

Type	Variables	Description
Device	$g_i(x,y)$	Gate i is placed in the coordinates (x,y).
	$g_{i(d)}(x,y)$	Driver of the gate i is placed in (x,y).
	$g_{i(w)}(x,y)$	Wire of the gate i is placed in (x,y).
	$g_i^c(x,y)$	Gate i is placed in (x,y) and it operates in cycle c.
	$g_{i(d)}^c(x,y)$	Driver of the gate i is in (x,y) and it operates in cycle c.
	$g_{i(w)}^c(x,y)$	Wire of the gate i is in (x,y) and it operates in cycle c.
Interface	$r(x,y)$	Port r of the (x,y) cell is connected.
	$r_{(s)}(x,y)$	Port r of (x,y) is connected and it contains the signal s.

4.5.2.5 Number of Variables

The set of Boolean variables created through our modeling is composed of the device variables and the interface variables. We enumerate these sets as follow:

a. The number of device variables is given by

$$\#D = \#cols * \#rows * \#nodes * (3 + 3 * \#cycles), \tag{23}$$

where $\#cols$ and $\#rows$ are the number of columns and rows of the USE grid, respectively, $\#nodes$ is the number of nodes on the input hypergraph N, and $\#cycles$ corresponds to the number of cycles available for each cell.

b. The number of interface variables is given by

$$\#I = 12 * \#cols * \#rows, \tag{24}$$

where $\#cols$ and $\#rows$ are the number of columns and rows of the USE grid, respectively.

4.5.3 Design Constraints

This section presents the design constraints[3] for the synthesis of QCA circuits through the proposed approach. The constraints were derived from empirical observations (mostly from QCA circuits available in the literature such as in (FONTES et al., 2018; TRINDADE et al., 2016; FORMIGONI et al., 2021) presented in the related work section), so there is no formal guarantee that the following constraints lead to an optimal solution in terms of area. However, as supported by the results (Section 4.6), the area of the circuits generated via the proposed methodology is optimized when compared to other well-known approaches.

[3]Appendix B presents an example of how the following constraints are applied.

The constraints are divided into three categories: the placement constraints (Section 4.5.3.1), the synchronization constraints (Section 4.5.3.2), and the routing constraints (Section 4.5.3.3). This subsection will describe more deeply these restrictions presenting the general rules to be satisfied and the modeling equations associated with each one of the constraints.

4.5.3.1 Placement Constraints

The placement constraints are responsible to avoid invalid placement configurations of the cells in the circuit. Notice that only device variables are used for generating the placement restrictions ahead.

a. *Single driver constraint:* each driver should be placed at exactly one location of the grid, i.e., multiples drivers are not allowed. Formally, we can define

$$exactly(1, \{g_{i(d)}(x_0, y_0), g_{i(d)}(x_1, y_1), ..., g_{i(d)}(x_{max}, y_{max})\}), \tag{25}$$

such that x_{max} and y_{max} are the maximum values for the coordinates x and y of the USE grid, respectively.

b. *Overlapping constraint:* each location contains at most one gate or (exclusively) at most two wires containing at most two input drivers. To define these constraints, consider the following equations:

$$\begin{aligned}
&i_{i(d)}(x, y) \implies \\
&atmost(2, \{i_{0(d)}(x, y), i_{1(d)}(x, y), ...\}) \wedge \\
&exactly(0, \{g_{0(d)}(x, y), g_{1(d)}(x, y), ..., o_{0(d)}(x, y), o_{1(d)}(x, y), ...\}) \wedge \\
&atmost(2, \{i_{0(w)}(x, y), i_{1(w)}(x, y), ..., g_{0(w)}(x, y), g_{1(w)}(x, y), ..., o_{0(w)}(x, y), o_{1(w)}(x, y), ...\}),
\end{aligned} \tag{26}$$

for the case of the input cells,

$$\begin{aligned}
&g_{i(d)}(x, y) \implies \\
&exactly(1, \{g_{0(d)}(x, y), g_{1(d)}(x, y), ..., o_{0(d)}(x, y), o_{1(d)}(x, y), ...\}) \wedge \\
&exactly(0, \{i_{0(w)}(x, y), i_{1(w)}(x, y), ..., g_{0(w)}(x, y), g_{1(w)}(x, y), ..., o_{0(w)}(x, y), o_{1(w)}(x, y), ...\}),
\end{aligned} \tag{27}$$

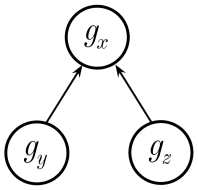


Figure 24 – An instance of a gate with two inputs.

for the case of gate and output cells, and

$$g_{i(w)}(x,y) \implies$$
$$exactly(0, \{i_{0(d)}(x,y), i_{1(d)}(x,y), ..., g_{0(d)}(x,y), g_{1(d)}(x,y), ..., o_{0(d)}(x,y), o_{1(d)}(x,y), ...\}) \wedge$$
$$atmost(2, \{i_{0(w)}(x,y), i_{1(w)}(x,y), ..., g_{0(w)}(x,y), g_{1(w)}(x,y), ..., o_{0(w)}(x,y), o_{1(w)}(x,y), ...\}), \quad (28)$$

for the case of any wire.

4.5.3.2 Synchronization Constraints

The synchronization constraints are responsible to ensure that all the inputs of a given cell arrive at the same time, as well as to synchronize the input and output of the data in the circuit. It is important to notice that only device variables are used for generating the following set of synchronization restrictions.

a. *Driver synchronization constraint:* for each gate or output driver, its inputs should be submitted to the same clock cycle if the cell is placed on locations upon phases 1, 2, or 3, and to the previous clock cycle if the cell is placed upon phase 0.

   To define the equations that implement this constraint, some auxiliary definitions are necessary:

   i. Consider a given gate $g_{(x)}$ such that its inputs are $g_{(y)}$ and $g_{(z)}$ as Figure 24 shows;

   ii. Consider the different USE cell configurations presented in Figure 25 in which we can notice the different cell orientations (denoted by the arrows) and the neighborhood of (x, y);

   iii. Finally, consider $phase(x, y)$ as a function that returns the phase of the cell located in the coordinates (x, y) of the USE grid.

This way, we have

$$\begin{aligned}
g^c_{x(d)}(x,y) &\implies (g^c_y(x,y-1) \vee g^c_y(x-1,y)) \wedge (g^c_z(x,y-1) \vee g^c_z(x-1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^c_y(x,y-1) \vee g^c_y(x+1,y)) \wedge (g^c_z(x,y-1) \vee g^c_z(x+1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^c_y(x,y+1) \vee g^c_y(x-1,y)) \wedge (g^c_z(x,y+1) \vee g^c_z(x-1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^c_y(x,y+1) \vee g^c_y(x+1,y)) \wedge (g^c_z(x,y+1) \vee g^c_z(x+1,y)),
\end{aligned} \tag{29}$$

for the cases presented in Figure25 (a), (b), (c), and (d), respectively, and for $phase(x,y) \geq 1$. Similarly, we have

$$\begin{aligned}
g^c_{x(d)}(x,y) &\implies (g^{c-1}_y(x,y-1) \vee g^{c-1}_y(x-1,y)) \wedge (g^{c-1}_z(x,y-1) \vee g^{c-1}_z(x-1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^{c-1}_y(x,y-1) \vee g^{c-1}_y(x+1,y)) \wedge (g^{c-1}_z(x,y-1) \vee g^{c-1}_z(x+1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^{c-1}_y(x,y+1) \vee g^{c-1}_y(x-1,y)) \wedge (g^{c-1}_z(x,y+1) \vee g^{c-1}_z(x-1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^{c-1}_y(x,y+1) \vee g^{c-1}_y(x+1,y)) \wedge (g^{c-1}_z(x,y+1) \vee g^{c-1}_z(x+1,y)),
\end{aligned} \tag{30}$$

for the cases presented in Figure25 (a), (b), (c), and (d), respectively, but now con-

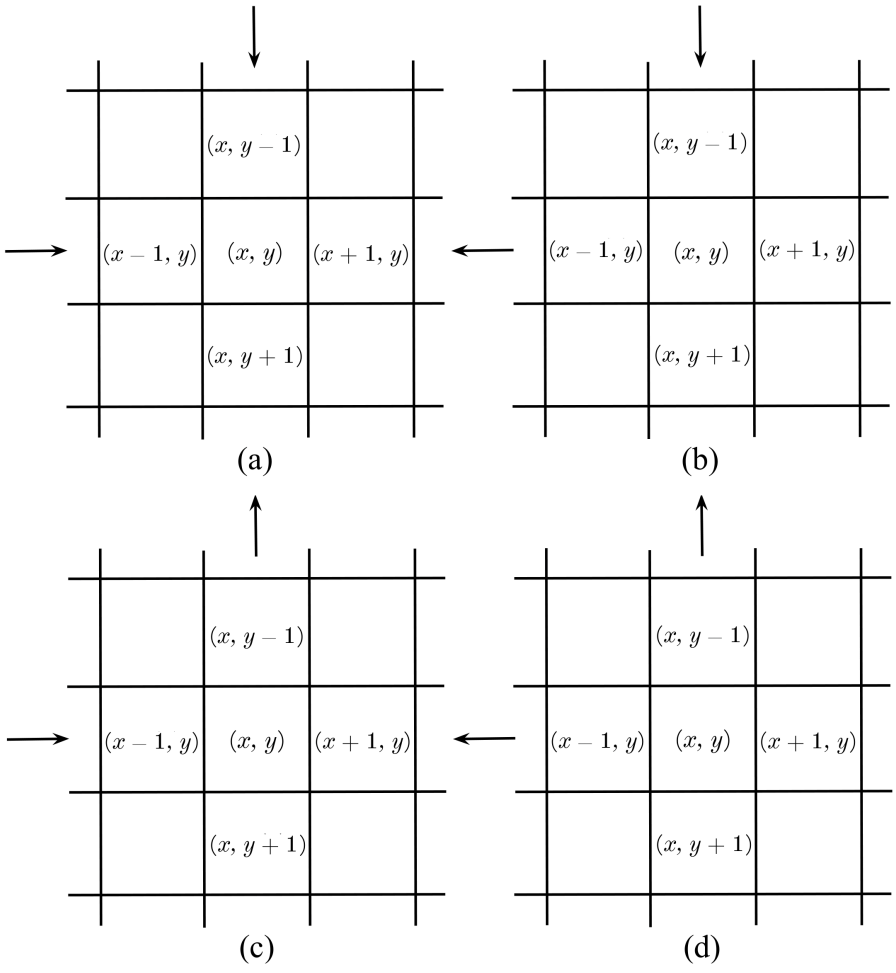


Figure 25 – Neighbors of the (x,y) USE cell.

sidering $phase(x, y) = 0$.

b. *Wire synchronization constraint:* for each wire cell, its input (the driver that originates this wire or another instance of this wire) should be submitted to the same clock if the cell is submitted to the phase 1, 2, or 3, and to the previous clock cycle if the cell is submitted to the phase 0.

Similarly to the driver synchronization constraint, we make use of the auxiliary definitions previously presented. This way, we have

$$\begin{aligned} g^c_{x(w)}(x,y) &\implies g^c_x(x,y-1) \vee g^c_x(x-1,y)), \\ g^c_{x(w)}(x,y) &\implies g^c_x(x,y-1) \vee g^c_x(x+1,y)), \\ g^c_{x(w)}(x,y) &\implies g^c_x(x,y+1) \vee g^c_x(x-1,y)), \\ g^c_{x(w)}(x,y) &\implies g^c_x(x,y+1) \vee g^c_x(x+1,y)), \end{aligned} \tag{31}$$

for the cases presented in Figure25 (a), (b), (c), and (d), respectively, and for $phase(x, y) \geq 1$. Similarly, we have

$$\begin{aligned} g^c_{x(w)}(x,y) &\implies g^{c-1}_x(x,y-1) \vee g^{c-1}_x(x-1,y)), \\ g^c_{x(w)}(x,y) &\implies g^{c-1}_x(x,y-1) \vee g^{c-1}_x(x+1,y)), \\ g^c_{x(w)}(x,y) &\implies g^{c-1}_x(x,y+1) \vee g^{c-1}_x(x-1,y)), \\ g^c_{x(w)}(x,y) &\implies g^{c-1}_x(x,y+1) \vee g^{c-1}_x(x+1,y)), \end{aligned} \tag{32}$$

for the cases presented in Figure25 (a), (b), (c), and (d), respectively, such that $phase(x, y) = 0$.

c. *Single-cycle constraint:* each cell should operate under exactly one cycle, i.e., multiples cycles for a single cell are not allowed. Formally,

$$g_i(x,y) \implies exactly(1, \{g^0_i(x,y), g^1_i(x,y), ..., g^{\#cycles-1}_i(x,y)\}). \tag{33}$$

d. *Input/output phase synchronization constraint:* each input should be submitted to the same phase. The same is valid for each output.

To define this constraint, let us consider the following:

i. A vector $[(x^0_0, y^0_0), (x^0_1, y^0_1), ..., (x^0_k, y^0_k)]$ containing all the k coordinates submitted to phase 0;

ii. A vector $[(x^1_0, y^1_0), (x^1_1, y^1_1), ..., (x^1_l, y^1_l)]$ containing all the l coordinates submitted to phase 1;

iii. A vector $[(x_0^2, y_0^2), (x_1^2, y_1^2), ..., (x_m^2, y_m^2)]$ containing all the m coordinates submitted to phase 2;

iv. A vector $[(x_0^3, y_0^3), (x_1^3, y_1^3), ..., (x_n^3, y_n^3)]$ containing all the n coordinates submitted to phase 3.

Then, we have

$$i_{a(d)}(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{j=0..k} i_{b(d)}(x_j^0, y_j^0), \text{if } phase(x,y) = 0, \tag{34}$$

$$i_{a(d)}(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{j=0..l} i_{b(d)}(x_j^1, y_j^1), \text{if } phase(x,y) = 1, \tag{35}$$

$$i_{a(d)}(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{j=0..m} i_{b(d)}(x_j^2, y_j^2), \text{if } phase(x,y) = 2, \tag{36}$$

$$i_{a(d)}(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{j=0..n} i_{b(d)}(x_j^3, y_j^3), \text{if } phase(x,y) = 3, \tag{37}$$

such that $\#i$ denotes the number of input nodes in the netlist. Furthermore, similar formulas are also defined for the case of the outputs.

e. *Input/output cycle synchronization constraint:* each input should be submitted to the same clock cycle. The same is valid for each output. Formally, we have

$$i^c_{a(d)}(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{\substack{x=0..x_{max} \\ y=0..y_{max}}} i^c_{b(d)}(x,y), \tag{38}$$

where $\#i$ denotes the number of input nodes in the netlist. A similar formula is also defined for the outputs.

4.5.3.3 Routing Constraints

The routing constraints are responsible for connecting the elements in the USE grid. Differing from the other placement and synchronization constraints, the routing restrictions mostly make use of the interface variables previously defined in Section 4.5.2.2.

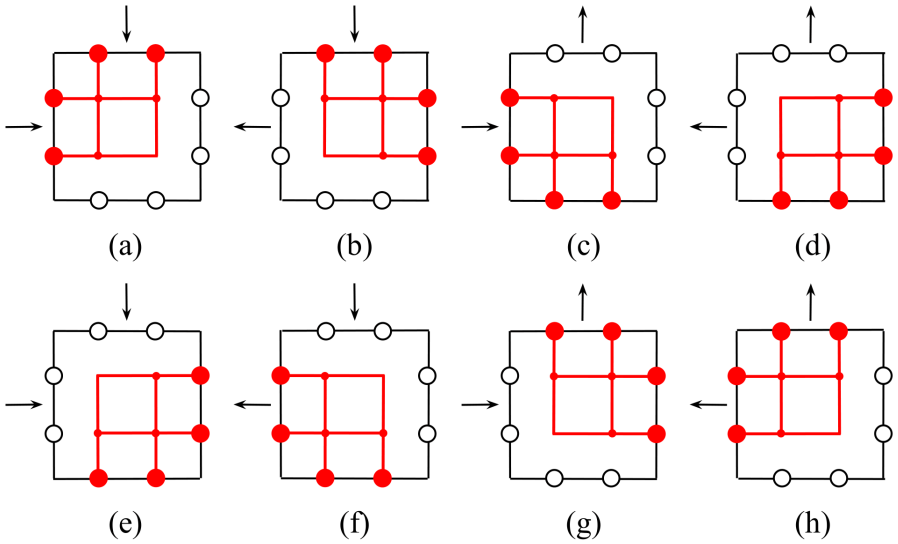


Figure 26 – Invalid paths of information in a cell accordingly with its orientation (denoted by the arrows in the USE grid).

a. *No return constraint:* each signal should not enter and exit the cell by the same side. Formally,

$$\bigwedge_{\substack{x=0..x_{max}\\y=0..y_{max}\\s=0..1}} \neg(ws_{(s)}(x,y) \wedge wn_{(s)}(x,y)) \ \wedge \bigwedge_{\substack{x=0..x_{max}\\y=0..y_{max}\\s=0..1}} \neg(nw_{(s)}(x,y) \wedge ne_{(s)}(x,y)). \tag{39}$$

b. *Border constraint:* ports located on the border of the routing area should not be connected. Formally,

$$\bigwedge_{x=0..x_{max}} \neg nw(x,0) \wedge \neg ne(x,0) \ \wedge \bigwedge_{y=0..y_{max}} \neg ws(0,y) \wedge \neg wn(0,y). \tag{40}$$

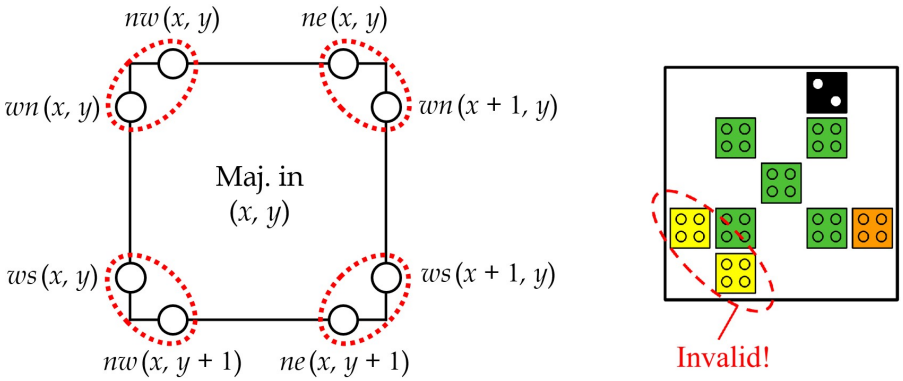
c. *Invalid corners constraint:* since each grid cell has some specific signal flow direction assigned to it (accordingly with the arrows located on the borders of the USE scheme), then all the routing paths presented in Figure 26 are forbidden, i.e., the information could not propagate through the red paths shown in the figure.

   Following what is defined in Figure 26, we can formulate this restriction as follows:

$$\bigwedge_{s=0..1} \neg((nw_{(s)}(x,y+1) \vee ne_{(s)}(x,y+1)) \wedge (ws_{(s)}(x+1,y) \vee wn_{(s)}(x+1,y))) \tag{41}$$

   prevents the situation presented in (a) and (h),

$$\bigwedge_{s=0..1} \neg((nw_{(s)}(x,y) \vee ne_{(s)}(x,y)) \wedge (ws_{(s)}(x+1,y) \vee wn_{(s)}(x+1,y))) \tag{42}$$



(a) (b)

Figure 27 – (a) Pairs of ports that should not be simultaneously connected. (b) Example of an invalid majority gate configuration.

avoids the cases (b) and (g),

$$\bigwedge_{s=0..1} \neg((nw_{(s)}(x,y+1) \vee ne_{(s)}(x,y+1)) \wedge (ws_{(s)}(x,y) \vee wn_{(s)}(x,y))) \tag{43}$$

deals with the cases (c) and (f), while

$$\bigwedge_{s=0..1} \neg((nw_{(s)}(x,y) \vee ne_{(s)}(x,y)) \wedge (ws_{(s)}(x,y) \vee wn_{(s)}(x,y))) \tag{44}$$

implements the constraints for preventing (d) and (e) to happen.

d. *Majority access constraint:* in a cell containing a majority gate (or any and or or gate based on the majority topology), two neighbor ports should not be connected if they are located in the corners of the USE cell. Figure 27 (a) presents the forbidden pairs of ports simultaneously connected, while Figure 27 (b) shows an instance of an invalid majority gate configuration.

   Considering a given node g_i of the input netlist, if $degree(g_i) \geq 2$ then g_i is implementing a majority-gate. This way, we can define

$$\begin{aligned} g_{i(d)}(x,y) \implies & \neg(wn(x,y) \wedge nw(x,y)) \wedge \neg(ne(x,y) \wedge wn(x+1,y)) \wedge \\ & \neg(ws(x,y) \wedge nw(x,y+1)) \wedge \neg(ws(x+1,y) \wedge ne(x,y+1)). \end{aligned} \tag{45}$$

e. *Gate driver adjacency constraint:* each gate (i.e., a majority or an inverter) should be followed (accordingly with the orientation of the USE cell containing this device) by the cells connected to its output or by its wires and should be preceded by the cells connected to its inputs.

   For this constraint, consider a given gate g_x such that its inputs are connected to g_y

and g_z and its output is the gate g_a as presented in Figure 28. This way, we have

$$\begin{aligned}
g_{x(d)}(x,y) \implies & exactly(1,\{g_y(x,y-1),g_y(x-1,y)\}) \wedge \\
& exactly(1,\{g_z(x,y-1),g_z(x-1,y)\}), \\
g_{x(d)}(x,y) \implies & exactly(1,\{g_y(x,y-1),g_y(x+1,y)\}) \wedge \\
& exactly(1,\{g_z(x,y-1),g_z(x+1,y)\}), \\
g_{x(d)}(x,y) \implies & exactly(1,\{g_y(x,y+1),g_y(x-1,y)\}) \wedge \\
& exactly(1,\{g_z(x,y+1),g_z(x-1,y)\}), \\
g_{x(d)}(x,y) \implies & exactly(1,\{g_y(x,y+1),g_y(x+1,y)\}) \wedge \\
& exactly(1,\{g_z(x,y+1),g_z(x+1,y)\}),
\end{aligned} \tag{46}$$

for the grid configurations presented in Figure 25 (a), (b), (c), and (d), respectively. Along with that, we have

$$\begin{aligned}
g_{x(d)}(x,y) \implies & exactly(1,\{g_{x(w)}(x,y+1),g_{x(w)}(x+1,y)\}) \wedge \\
& exactly(1,\{g_a(x,y+1),g_a(x+1,y)\}), \\
g_{x(d)}(x,y) \implies & exactly(1,\{g_{x(w)}(x,y+1),g_{x(w)}(x-1,y)\}) \wedge \\
& exactly(1,\{g_a(x,y+1),g_a(x-1,y)\}), \\
g_{x(d)}(x,y) \implies & exactly(1,\{g_{x(w)}(x,y-1),g_{x(w)}(x+1,y)\}) \wedge \\
& exactly(1,\{g_a(x,y-1),g_a(x+1,y)\}), \\
g_{x(d)}(x,y) \implies & exactly(1,\{g_{x(w)}(x,y-1),g_{x(w)}(x-1,y)\}) \wedge \\
& exactly(1,\{g_a(x,y-1),g_a(x-1,y)\}),
\end{aligned} \tag{47}$$

also for the cases presented in Figure 25 (a), (b), (c), and (d), respectively.

f. *Input driver adjacency constraint:* an input driver connected to a given gate should be followed (accordingly with the orientation of the USE grid) by one of its own wires or by this gate. Furthermore, if the input is connected to more than one gate, then it should be followed by one of its own wires or by at least one of these gates.

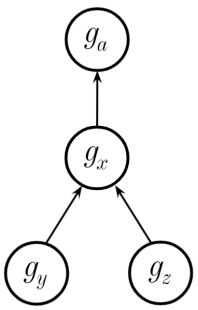


Figure 28 – An instance of a gate with two inputs and one output.

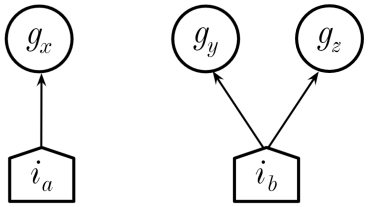


Figure 29 – Instances of gates with one (a) and two (b) inputs.

To define this constraint, let us consider an input i_a connected to g_x and an input i_b connected to g_y and g_z as shown in Figure 29. This way, considering Figure 29 (a), i.e., $degree(i_a) = 1$, we have

$$\begin{aligned}
i_{a(d)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x+1,y), g_{x(d)}(x,y+1), g_{x(d)}(x+1,y)\}),\\
i_{a(d)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x-1,y), g_{x(d)}(x,y+1), g_{x(d)}(x-1,y)\}),\\
i_{a(d)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x+1,y), g_{x(d)}(x,y-1), g_{x(d)}(x+1,y)\}),\\
i_{a(d)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x-1,y), g_{x(d)}(x,y-1), g_{x(d)}(x-1,y)\}),
\end{aligned} \tag{48}$$

for USE grid configurations presented in Figure 25 (a), (b), (c), and (d), respectively. Along with that, considering Figure 29 (b), i.e., $degree(i_a) \geq 2$, we have

$$\begin{aligned}
i_{a(d)}(x,y) &\implies i_{a(w)}(x,y+1) \vee i_{a(w)}(x+1,y) \vee g_{y(d)}(x,y+1) \vee g_{y(d)}(x+1,y) \vee\\
&\quad g_{z(d)}(x,y+1) \vee g_{z(d)}(x+1,y),\\
i_{a(d)}(x,y) &\implies i_{a(w)}(x,y+1) \vee i_{a(w)}(x-1,y) \vee g_{y(d)}(x,y+1) \vee g_{y(d)}(x-1,y) \vee\\
&\quad g_{z(d)}(x,y+1) \vee g_{z(d)}(x-1,y),\\
i_{a(d)}(x,y) &\implies i_{a(w)}(x,y-1) \vee i_{a(w)}(x+1,y) \vee g_{y(d)}(x,y-1) \vee g_{y(d)}(x+1,y) \vee\\
&\quad g_{z(d)}(x,y-1) \vee g_{z(d)}(x+1,y),\\
i_{a(d)}(x,y) &\implies i_{a(w)}(x,y-1) \vee i_{a(w)}(x-1,y) \vee g_{y(d)}(x,y-1) \vee g_{y(d)}(x-1,y) \vee\\
&\quad g_{z(d)}(x,y-1) \vee g_{x(d)}(x-1,y),
\end{aligned} \tag{49}$$

also for the cases shown in Figure 25 (a), (b), (c), and (d), respectively.

g. *Wire adjacency constraint:* a wire should be followed (accordingly with the orientation of the USE cell containing this device) by another instance of the same wire or by the gates connected to its output. Along with that, the wire should be preceded by another instance of the same wire or by the gate that originates this wire.

To define this constraint, consider what was proposed previously in Figure 29 (a),

where the input i_a is connected to the gate g_x. This way, we have

$$\begin{aligned}
i_{a(w)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x+1,y), g_{x(d)}(x,y+1), g_{x(d)}(x+1,y)\}),\\
i_{a(w)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x-1,y), g_{x(d)}(x,y+1), g_{x(d)}(x-1,y)\}),\\
i_{a(w)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x+1,y), g_{x(d)}(x,y-1), g_{x(d)}(x+1,y)\}),\\
i_{a(w)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x-1,y), g_{x(d)}(x,y-1), g_{x(d)}(x-1,y)\}),
\end{aligned} \tag{50}$$

for the cases shown in Figure 25 (a), (b), (c), and (d), respectively. Along with that, we have

$$\begin{aligned}
i_{a(w)}(x,y) &\implies atleast(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x-1,y), i_{a(d)}(x,y-1), i_{a(d)}(x-1,y)\}),\\
i_{a(w)}(x,y) &\implies atleast(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x+1,y), i_{a(d)}(x,y-1), i_{a(d)}(x+1,y)\}),\\
i_{a(w)}(x,y) &\implies atleast(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x-1,y), i_{a(d)}(x,y+1), i_{a(d)}(x-1,y)\}),\\
i_{a(w)}(x,y) &\implies atleast(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x+1,y), i_{a(d)}(x,y+1), i_{a(d)}(x+1,y)\}),
\end{aligned} \tag{51}$$

also considering the cases defined in Figure 25 (a), (b), (c), and (d), respectively.

4.5.3.4 Table of Design Constraints

Table 4 summarizes the constraints defined in our formulation for the cell synthesis in QCA.

4.5.4 Algorithm

This section presents the proposed method (Algorithm 4) for generating synchronized QCA circuits with optimizations in area. The algorithm is based on the variables and constraints defined in the previous section of this chapter[4].

In this algorithm, consider $orientation$ as a vector containing the possible orientations for the origin (coordinate $(0,0)$ of the USE grid) and $phases$ as a vector containing the phases of the origin. Along with that, let us consider `Factor(`$area$`)` as a function that returns a vector $[(r_0,c_0),(r_1,c_1),...,(r_n,c_n)]$ which contains all the n combinations of r and c (rows and columns) such that $r_i * c_i = area$ (e.g., if $area = 7$, then $combinations = [(7,1),(1,7)]$; if $area = 12$, then $combinations = [(1,12),(2,6),(3,4),(4,3),\ (6,2),(12,1)]$). Moreover, the functions `Placement(...)`, `Synchronization(...)`, and `Routing(...)` are responsible for creating the clauses described in Sections 4.5.3.1, 4.5.3.2, and 4.5.3.3, respectively. Finally, the function

[4]As in the previous chapter, the set of formulas presented are not necessarily in the CNF format widely adopted for most of the SAT solvers available. Thus, we employ the Tseitin transformation (TSEITIN, 1983) to convert to the CNF format each one of the elements created by the equations.

Table 4 – Table of design constraints: QCA synthesis.

Type	Constraint	Description
Plac.	Single driver	Each driver should be placed at exactly one grid location.
	Overlapping	Each grid location contains at most one gate or (exclusively) at most two wire cells containing at most two input drivers.
Sync.	Driver sync.	For each gate or output driver, its inputs should be submitted to the same clock cycle if the cell is placed on locations upon phases 1, 2, or 3, and to the previous clock cycle if the cell is placed upon phase 0.
	Wire sync.	For each wire cell, its input should be submitted to the same clock if the cell is submitted to the phase 1, 2, or 3, and to the previous clock cycle if the cell is submitted to the phase 0.
	Single-cycle	Each cell should operate under exactly one cycle.
	I/O phase sync.	Each input should be submitted to the same phase. The same is valid for each output.
	I/O cycle sync.	Each input should start operating on the same clock cycle. The same is valid for each output.
Rout.	No return	Each signal should not enter and exit the cell on the same side.
	Border	Ports located on the border of the routing area should not be connected.
	Invalid corners	Some routing paths are not allowed (see Figure 26).
	Majority access	In a cell containing a majority gate, two neighbor ports should not be connected if they are located in the corners of this USE cell.
	Gate driver adj.	Each gate should be followed by the cells connected to its output or by its wires and should be preceded by the cells connected to its inputs.
	Input driver adj.	Each input driver connected to one gate should be followed by one of its own wires or by the gate. Furthermore, if the input is connected to more than one gate, then it should be followed by one of its own wires or by at least one of the gates connected to its output.
	Wire adj.	Each wire should be followed by the cells connected to its output or by other wire of the same signal and should be preceded by a wire or by a driver of the same signal.

Algorithm 4: SAT-based pseudo-code for the synthesis of QCA circuits

```
input  : input hypergraph N, number of cycles c
output: QCA-USE design circuit
orientation ← (⇂, ↿, ↾, ↿)
phases ← (0, 1, 2, 3)
area ← Nodes(N)
while True do
    combinations ← Factor(area)
    for i ← 0 to combinations.size do
        rows, cols ← combinations(i)
        for o ← 0 to orientation.size do
            for p ← 0 to phases.size do
                V ← CreateBooleanVariables(N, c, rows, cols)
                placement ← Placement(N, V, c, rows, cols, orientation(o), phases(p))
                sync ← Synchronization(N, V, c, rows, cols, orientation(o), phases(p))
                routing ← Routing(N, c, V, rows, cols, orientation(o), phases(p))
                SAT ← Solver(placement, sync, routing)
                if SAT ≠ False then
                    circuit ← Translate (SAT)
                    return circuit
    area ← area + 1
```

`Translate(...)` is in charge of transforming the result obtained by the SAT solver (the set of variables assigned to $True$ on the solution) into the QCA circuit.

In order to exemplify the solutions obtained through the presented method, consider the QCA circuit illustrated in Figure 30 which implements the XOR2 function described by the input netlist shown in Figure 22.

4.6 Experiments

The experiments presented in this section were conducted under a benchmark of well-known digital circuits and are divided into two parts: the first consists of an assessment of the solutions produced through the proposed method considering different synchronization profiles, while the second is a comparison with solutions generated through different methodologies. It is important to notice that the QCADesigner (WALUS et al., 2004) was adopted to validate and test the circuits here presented.

4.6.1 Assessment of Synchronization Profiles

The first experiment defines two synchronization profiles for testing (where we ensure the global synchronicity of the circuit based on the design principles described in (TORRES et al., 2018)): (i) synchronization at clock phase level and (ii) synchronization at clock cycle level. These different profiles are detailed below:

i. Synchronization at clock phase level: in this scenario, all the inputs of the circuit are submitted to the same phase and to the same cycle (the same is valid for the

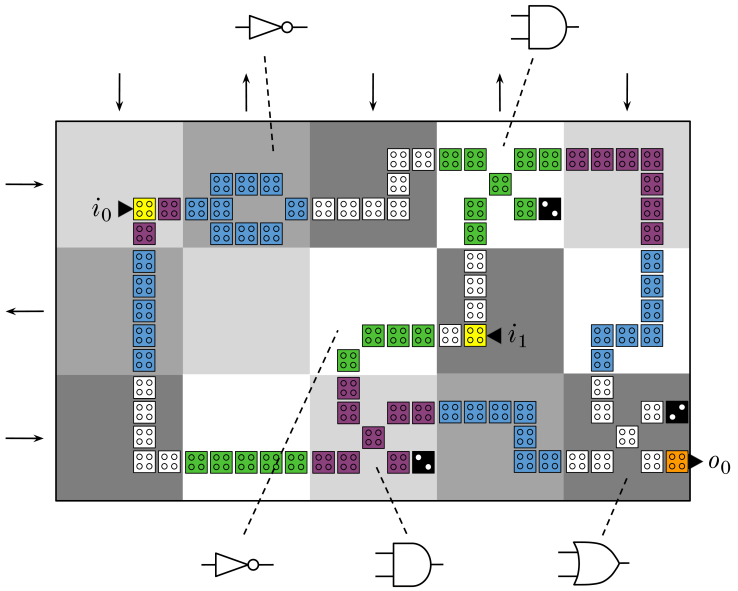


Figure 30 – QCA circuit obtained through the proposed SAT-based approach implementing the XOR2 function.

outputs). This enables to the circuit receives a new data on each clock phase (i.e., achieving the maximum throughput). For generating synchronized circuits at clock phase level we include in our formulation all the synchronization restrictions defined in Section 4.5.3.2;

ii. Synchronization at clock cycle level: in this case, each input of the circuit is submitted to the same cycle on the USE grid (the same is valid for the outputs), i.e. there are no restrictions regarding the phase of each input and output. This scenario ensures that new data can be computed on each clock cycle (composed of four clock phases as shown in Section 4.2.3). For generating synchronized circuits at clock cycle level we did not include in our formulation the restriction defined in Section 4.5.3.2d.

The results of the first experiment are summarized in Table 5, where *#G*, *#I*, and *#O* refers to the number of gates, inputs, and outputs of the netlists, respectively, *Area* is the area of the resulting layout in terms of USE cells (i.e., each unit of area comprises a 5x5 QCA structure as shown in Figure 19), *Lat.* is the latency of the critical path of the circuit (relative to the number of clock phases), and *Occ. (%)* is the occupancy rate of the QCA design.

Considering the data presented in Table 5, we can notice that the solutions with synchronization at clock cycle level present optimizations in area for most of the cases. However, as pointed before, these circuits present a smaller throughput compared to the circuits with clock phase synchronization since the data input is limited by the clock cycle in the former, while by the clock phase in the latter (this tradeoff is expected as discussed in (TORRES et al., 2018)). In this scenario, the circuit designer can choose

Table 5 – Comparison between different synchronicity profiles: QCA synthesis.

Circuit	#G	#I	#O	Phase synchronicity			Cycle synchronicity		
				Area	Lat.	Occ. (%)	Area	Lat.	Occ. (%)
2:1 MUX	3	3	1	4×3	4	91.7	4×3	5	75.0
XOR2	4	2	1	4×4	7	100.0	5×3	6	93.3
XNOR2	5	2	1	4×4	7	100.0	4×4	7	87.5
Full adder	7	3	2	7×7	12	79.6	8×5	14	90.0
C17	5	5	2	6×4	7	87.5	7×3	8	95.2
t	13	5	2	7×5	6	80.0	7×4	7	82.1
newtag	14	8	1	8×5	9	85.0	7×5	9	94.3
Par. gen.	9	3	1	7×5	10	88.6	7×5	14	97.1
Par. ch.	14	4	1	7×7	12	87.8	7×7	14	91.8

which version fits well accordingly with the design specifications. Besides that, we can notice that the occupancy rate of both synchronization profiles assessed are high, meaning that our method was able to achieve solutions that minimize the number of blank spaces in the layout.

In order to illustrate the difference between the two approaches, consider the 2-input XOR (XOR2) circuit presented in Figure 31, where (a) presents the solution with clock phase synchronicity and (b) shows the QCA circuit with clock cycle synchronicity. Notice that (b) presents an optimization in area compared to (a). However, (a) possess a higher throughput given that new inputs can be applied in each clock phase.

4.6.2 Comparison with other Methodologies

The second experiment consists of a comparison between the solutions produced through the proposed methodology and other methods available in the literature. In this scenario, (FONTES et al., 2018), (TRINDADE et al., 2016), and (FORMIGONI et al., 2021) were employed for this assessment since all the approaches generate

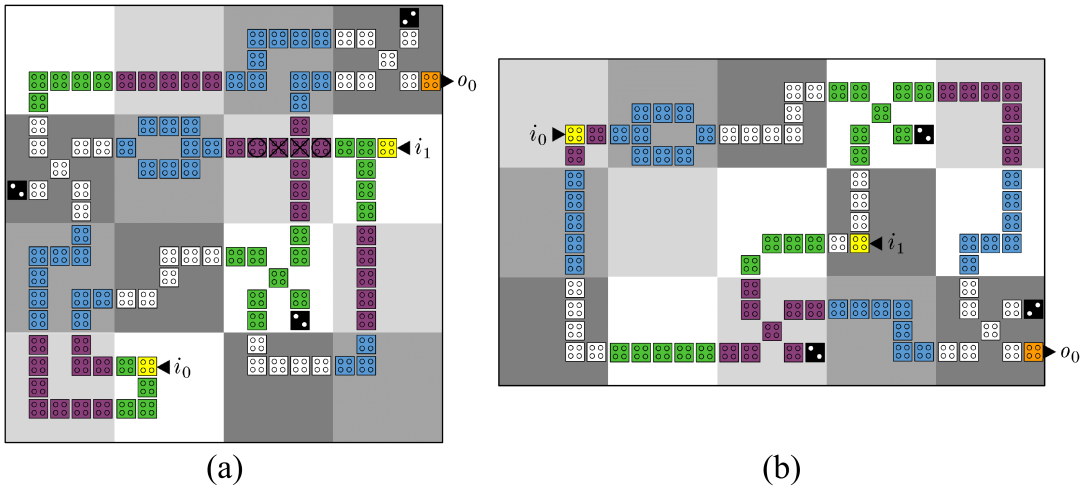


Figure 31 – XOR2 circuits. (a) Solution with clock phase synchronicity. (b) Solution with clock cycle synchronicity.

Table 6 – Comparison with other methodologies: QCA synthesis.

Circuit	Proposed method			Fontes *et al.*			Trindade *et al.*			Formigoni *et al.*		
	Area	L.	Occ.	Area	L.	Occ.	Area	L.	Occ.	Area	L.	Occ.
2:1 MUX	4×3	4	91.7	5×3	4	80.0	5×4	5	75.0	6×5	6	56.6
XOR2	4×4	7	100.0	4×4	4	81.2	7×4	7	71.4	9×7	12	46.0
XNOR2	4×4	7	100.0	4×6	8	75.0	5×6	8	73.3	-	-	-
F. adder	7×7	12	79.6	7×8	12	64.3	-	-	-	-	-	-
C17	6×4	7	87.5	8×6	8	66.7	-	-	-	-	-	-
t	7×5	6	80.0	60	10	-	-	-	-	-	-	-
newtag	8×5	9	85.0	80	12	-	-	-	-	-	-	-
P. gen.	7×5	10	88.6	7×6	12	88.1	10×9	14	50.0	16×6	14	55.2
P. ch.	7×7	12	87.8	6×10	13	73.3	16×6	14	58.3	16×11	22	39.2

synchronized solutions at clock phase level (as detailed in Sections 4.3.3, 4.3.2, and 4.3.4, respectively). Thus, in this experiment, we also ensure the same synchronicity profile for the circuits obtained through our method in order to provide a fair comparison. The results of the second experiment are summarized in Table 6.

Considering the data presented in Table 6, we can notice that the proposed approach was able to generate circuits with optimizations in area for all the cases in comparison with (FONTES et al., 2018), (TRINDADE et al., 2016), and (FORMIGONI et al., 2021) methods (with the exception of the XOR2 layout which has presented the same area of the (FONTES et al., 2018) solution). Along with that, the occupancy rate achieved was considerably higher for the entire benchmark, meaning that the SAT-based approach was able to minimize the blank spaces in the layout in comparison with the solutions produced by (FONTES et al., 2018), (TRINDADE et al., 2016), and (FORMIGONI et al., 2021). Finally, as a side effect of the area minimization, most of the solutions also presented a smaller latency since this parameter is usually related to the size of the circuit.

Finally, Figure 32 illustrates the 2:1 multiplexer (2:1 MUX) circuits generated through the proposed methodology (a), via the (FONTES et al., 2018) methodology (b), by the (TRINDADE et al., 2016) methodology (c), and through the (FORMIGONI et al., 2021) methodology.

4.6.2.1 Case Study: Full Adder

In order to highlight the differences between the proposed approach and the method presented in (FONTES et al., 2018), let us consider the full adder presented in Figure 33 designed through these two different methodologies[5].

As presented in (FONTES et al., 2018) and in Section 4.3.3 of this chapter, the

[5]In this case, the (TRINDADE et al., 2016; FORMIGONI et al., 2021) methods were not considered for this analysis because it present worst results for every circuit compared to (FONTES et al., 2018) as Table 6 shown.

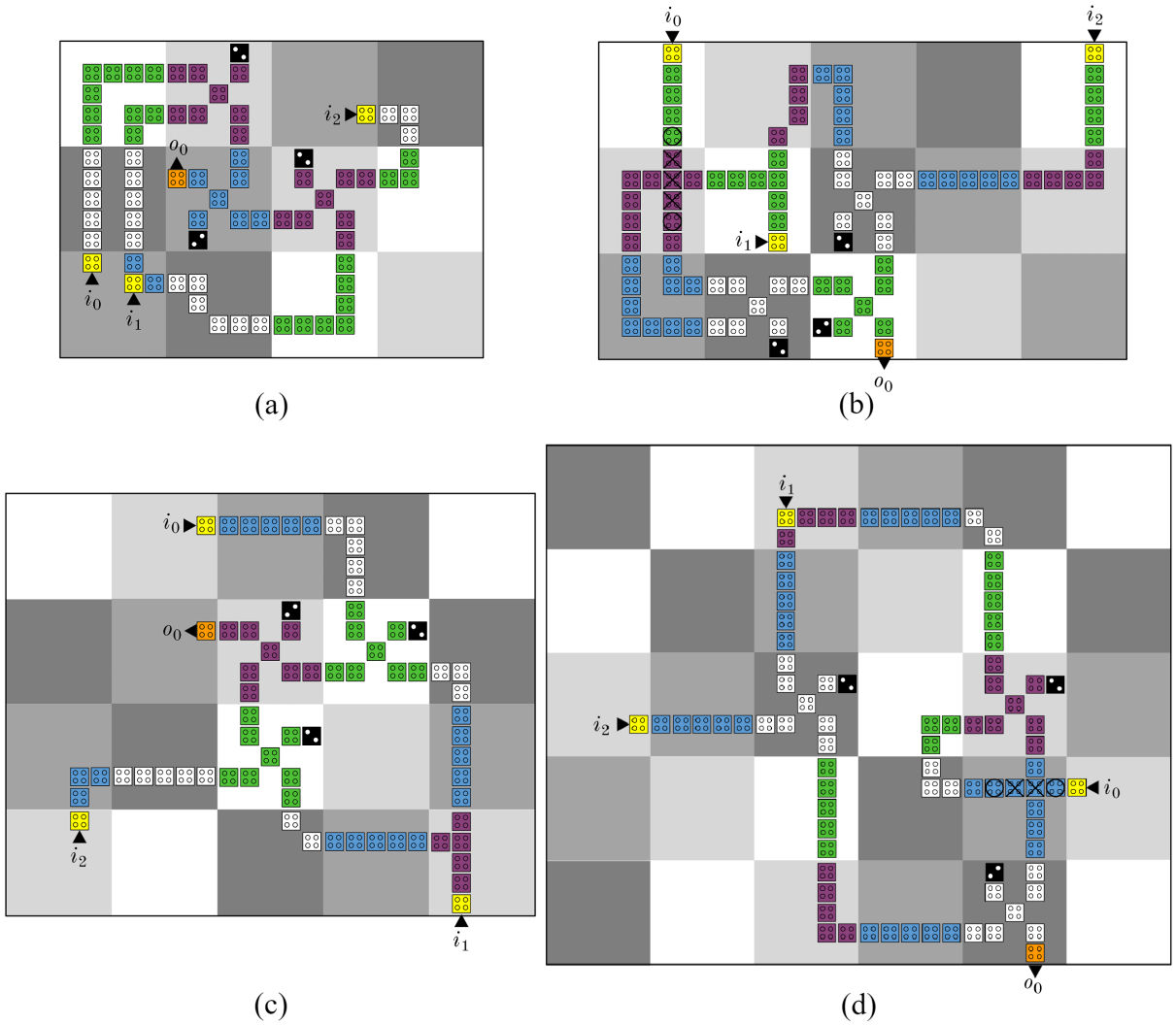


Figure 32 – Different versions of the 2:1 MUX. (a) Proposed solution. (b) Solution obtained through (FONTES et al., 2018) methodology. (c) Solution generated through (TRINDADE et al., 2016). (d) Solution obtained through (FORMIGONI et al., 2021).

circuit design algorithm starts defining how the levels of the input graph are organized. Right after that, a fixed latency value of each level is defined. Finally, the search for the solution begins in breadth-first order. If the placement and routing of all the nodes of a given level are not feasible, then the latency of this level is increased in one unity (where the maximum latency l_{max} is given by $l_{max} = max(4, d)$, where d is the depth of the input graph). If the maximum latency of a given level is exceeded without finding any feasible solution, then the algorithm backtracks, the latency of the upper level is increased, and the algorithm proceeds from that. Figure 33 (a) presents the input netlist with its respective level latencies computed from the (FONTES et al., 2018) algorithm in order to find the solution (b).

Our method acts in a dramatically different way. Firstly, we do not define levels for the input netlist since the synchronization constraints are modeled in the formulation and it is only based on the connections of the cells (as presented in Section 4.5.3.2 of

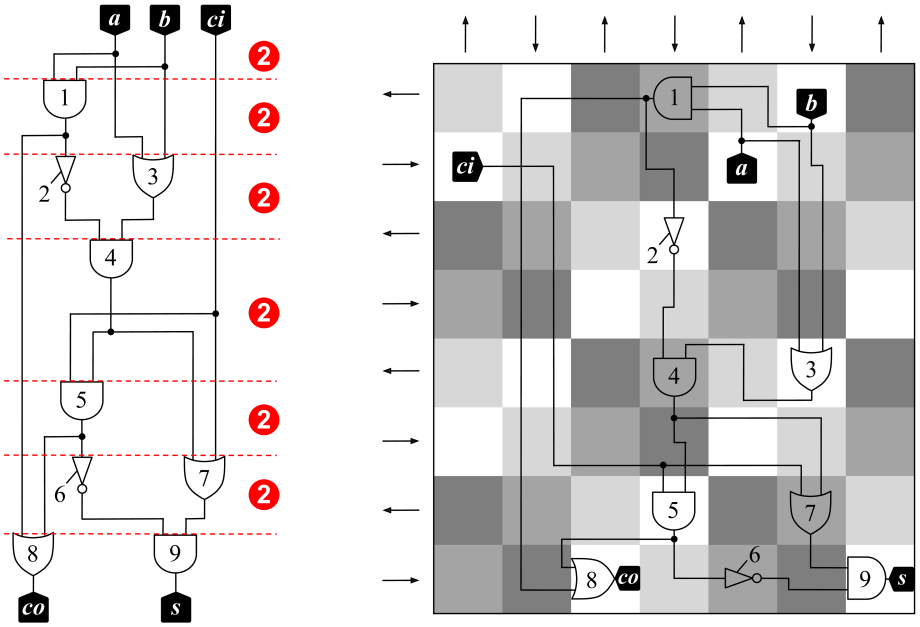


Figure 33 – Full adder synthesis performed through (FONTES et al., 2018). (a) Netlist with latencies proposed in (FONTES et al., 2018). (b) QCA circuit generated from the netlist presented in (a).

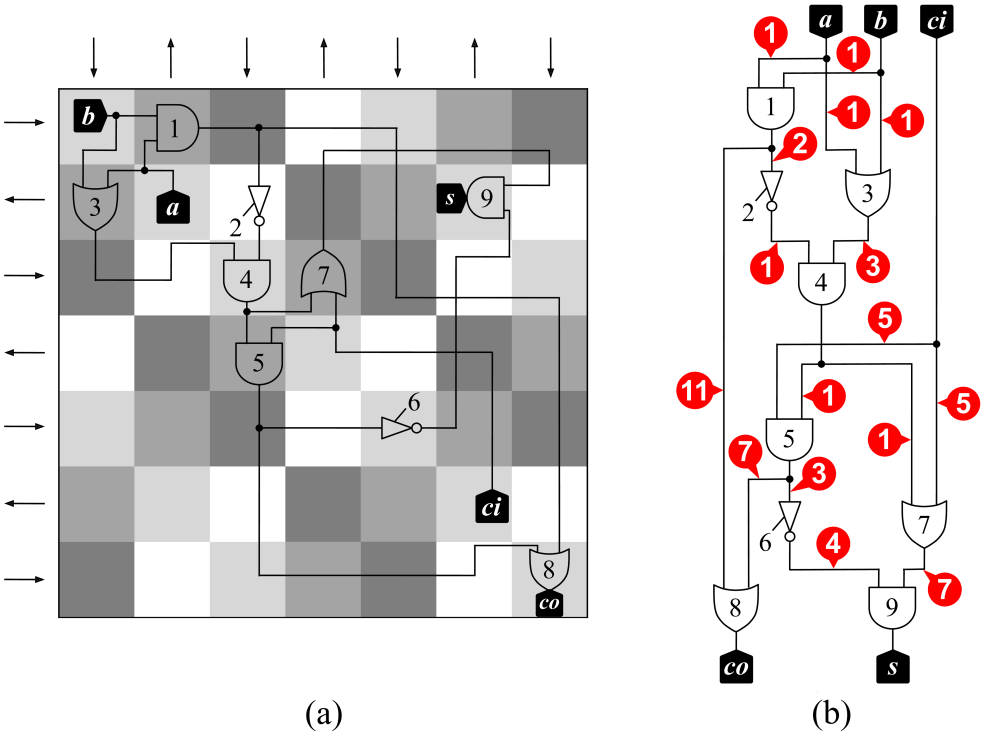


Figure 34 – Full adder synthesis performed through the SAT-based method. (a) Circuit generated via the proposed method. (b) Netlist with latencies obtained from the solution (a).

this chapter). This way, each connection (wire) of the circuit contains its own latency. Notice that, even with this characteristic, the local synchronicity of all the gates on the circuit is ensured. In order to illustrate this, from the circuit presented in Figure 34 (a) generated through the proposed method, we derive the netlist with latencies

shown in (b). It can be seen that, for any gate of the circuit, each path connecting the inputs of this given gate to the inputs of the circuit contains the same latency (i.e., the synchronicity at clock phase level is ensured).

4.7 Chapter Conclusions

In this chapter we presented a new SAT-based placement and routing algorithm for generating synchronized QCA circuits. A set of Boolean variables to represent the problem and formulas to describe the restrictions imposed by the designing rules were created. The method relies on the USE clocking scheme, a structure responsible for encoding the several design restrictions intrinsic to the QCA design.

The results of the experiments have shown that the proposed methodology was able to deliver area optimized cells when compared to well-known methods available in the literature. Besides that, an exploration of different synchronization profiles was made, where a tradeoff between area and throughput was observed on the generated circuits.

Finally, the proposed methodology can be applied for several purposes such as on the design of QCA standard cell libraries, for the on-the-fly design of QCA circuits, for the integration with divide-and-conquer strategies to provide scalability, for validation and comparison with other new methodologies, among others.

5 NANOMAGNETIC LOGIC

Besides all the advances proposed by the quantum-dot cellular automata technology (QCA), the process of encoding information through a specific local position of charged particles in a confined area introduces new technological barriers in manufacturing and design tasks. In this scenario, the nanomagnetic logic (NML) (previously known as magnetic QCA, or MQCA) was proposed, in which some of these obstacles around controlling quantum particles are avoided by using nanomagnets as its core device.

As the QCA, the NML is a field-coupled nanotechnology where the binary information is represented through the polarity of nanomagnetic cells and it propagates to adjacent devices through magnetic coupling (CSABA et al., 2002; SOARES et al., 2018). Thus, the idea is to use the magnetic phenomena - widely adopted for data storage - to perform logic as well. In this scenario, its nonvolatile nature enables solutions where no energy is consumed in the standby operation, differently of charge-based technologies (such as the CMOS), where the static and dynamic power have similar levels nowadays (ANDERSON; BHANJA, 2014). This is the basis for the prototypes presented in (IMRE et al., 2006; VARGA et al., 2010, 2011), where proofs-of-concept were presented and drive years of research and development in the area.

Another similarity between QCA and NML is the clocking system, responsible for controlling the flow of information in the circuit and ensuring its expected behavior. Although it is possible to use the same four-phase clocking schemes as in QCA - i.e., USE (CAMPOS et al., 2016), RES (GOSWAMI et al., 2019), and 2DDWave (VANKAMAMIDI; OTTAVI; LOMBARDI, 2008), as cited in the previous chapter -, recently a new clocking scheme focused in the NML technology were proposed: the BANCS (FORMIGONI; VILELA NETO; NACIF, 2018). BANCS is a three-phase scheme that takes into account the design restrictions of NML, providing an environment for the implementation of placement and routing algorithms. Besides that, the use of clocking schemes enables scalability, among others advantages as discussed in the last chapter.

In this scenario, we present a new SAT-based method for placement and routing of NML circuits using BANCS as our clocking scheme. As in the previous chapter,

besides focusing on area optimization, our solution also provided good results in terms of latency and can be used for many purposes such as in the design of NML cell libraries, in the on-the-fly synthesis of NML circuits, among others.

5.1 Chapter Organization

The sections in this chapter are organized as follows: Section 5.2 presents concepts regarding the NML technology; Section 5.3 introduces the proposed methodology; Section 5.4 presents the results of our experiments; finally, Section 5.5 concludes this chapter[1].

5.2 Preliminaries

Nanomagnetic logic (NML) is a field-coupled nanocomputing (FCN) technology (ANDERSON; BHANJA, 2014). As in QCA, NML is capable of storing, transferring, and performing logic through local field interactions that occur between devices located close to each other. Since the phenomena of magnetic coupling applied in the NML functioning do not directly involve electric current flowing, this approach potentially leads to ultra-low power consumption (ANDERSON; BHANJA, 2014). Ahead in this section, some basic concepts related to NML are presented in order to provide a useful background for the remainder of this chapter.

5.2.1 NML Basics

The fundamental unit of an NML circuit is a rectangular-shaped nanomagnet - although other geometries are also possible (NIEMIER et al., 2012). The nanomagnet must be sufficiently small to present only one natural magnetic domain, where this magnetization is likely to lie along its longer axis - known as the easy axis - so it minimizes the shape energy. Therefore, there are two stable configurations, where the magnetization vector points in one of the two possible directions along the length of the rectangle. Based on this, we arbitrarily defined the logic values 0 and 1 when the magnetization vector points $down$ and up, respectively. Figure 35 (a) illustrates these states, where the blue part of the magnet is its south pole, where the red area is its north pole. Additionally, an external magnetic field can be applied to bring the targeted nanomagnets into a metastable configuration called $null$ state (GRAZIANO et al., 2011) which is useful for the clocking system. Along with that, the magnetic coupling between neighbors devices can occur in two different ways: antiferromagnetic and ferromagnetic. As shown in Figure 35 (b), the former configuration, antiferromagnetic, presents an antiparallel direction of the magnetization vectors, while the latter, ferro-

[1] It is important to point out that the related work and method overview sections are not presented since both are similar to what is shown in the QCA chapter of this thesis.

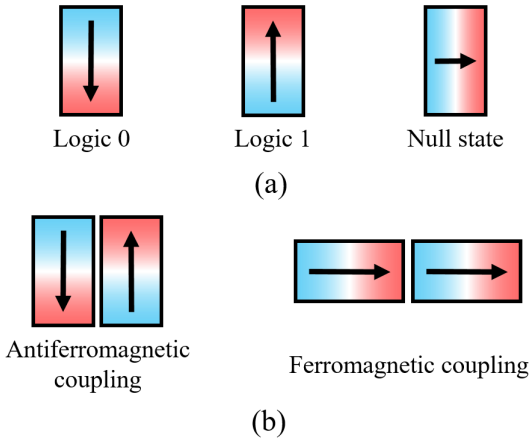


Figure 35 – NML cells. (a) NML cells encoding the logic 0, 1, and the $null$ states. (b) Coupling patterns.
Source: adapted from (SOARES et al., 2018).

magnetic, presents a parallel orientation. This way, by positioning the nanomagnets in one of these two patterns, it is possible to implement NML wires and perform logic operations (as detailed next in Section 5.2.4).

5.2.2 NML Clocking System

As in the QCA paradigm, the clocking system of the NML technology is responsible for inducing smooth changes in the magnetization profiles of the devices. This way, sudden changes on the devices do not lead the system to an invalid configuration. Along with that, the clocking is responsible for controlling the flow of information through the circuit, this way avoiding signals errors in long arrays of magnets. Figure 36 shows the data transfer in an NML array of cells with antiferromagnetic coupling.

The implementation of the clocking system is based on an external magnetic field that forces the cell to the $null$ state. By removing this external field, each cell assumes a configuration of a minimum state of energy accordingly with its neighbors. This

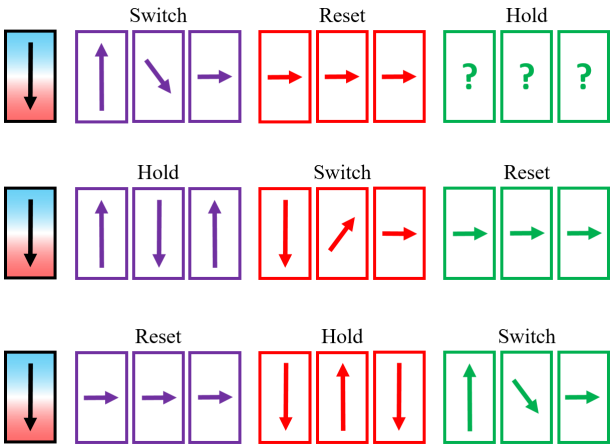


Figure 36 – Data transfer in an array of NML devices.
Source: adapted from (SOARES et al., 2018).

process, different from the one employed in QCA, is composed of three phases:

1. Reset: this phase is responsible for inducing the $null$ state in the cells. This is performed by applying an external magnetic field strong enough to force the cells into its metastable configuration (where its magnetization vector is perpendicular to the device longer axis);

2. Switch: this phase is characterized by gradual removal of the external magnetic field applied in the reset phase. This way, each device assume a minimum energy state based on the interaction with its neighbors;

3. Hold: this phase is responsible for keeping the cells in their current states, i.e., there is no external magnetic field applied.

The engineering of the clocking system in NML represents one of the main challenges for the adoption of this emerging technology. A broad discussion about this topic, along with potential candidates to solve this issue, is presented in (ANDERSON; BHANJA, 2014).

5.2.3 NML Clocking Schemes

As in the QCA circuit design, clocking schemes are also adopted for the NML circuit design since it ensures that all the clocking constraints are satisfied during the project phase. Thus, besides providing scalability for the system, the regular structures of the clocking schemes are a useful tool for enabling the implementation of automated synthesis methods for the placement and routing of NML circuits such as the one proposed in this chapter.

Since the NML clocking contains three-phase, any four-phased clocking scheme can be adopted as long as we avoid the placement of devices in the positions corresponding to the last (i.e., the fourth) clocking zone (FORMIGONI et al., 2021). Recently, a three-phased clocking scheme called BANCS (FORMIGONI; VILELA NETO; NACIF, 2018) was introduced with a focus on the design of NML circuits. Besides managing the correct synchronization of the circuit, BANCS also deals with the signal errors that can happen in long arrays of nanomagnets. As shown in (CSABA; POROD, 2010), wires with more than five nanomagnets present a high error rate due to thermal noise (FORMIGONI; VILELA NETO; NACIF, 2018). To avoid that, BANCS imposes that structures submitted to the same clocking zone must be smaller or equal to five magnets arranged in the ferromagnetic or antiferromagnetic coupling configurations. The BANCS design grid is illustrated in Figure 37, where cells located on a clock zone receive data from the neighbors located on a preceding zone and send data to the neighbors located on a succeeding zone, thus guaranteeing a correct flow of information within the circuit.

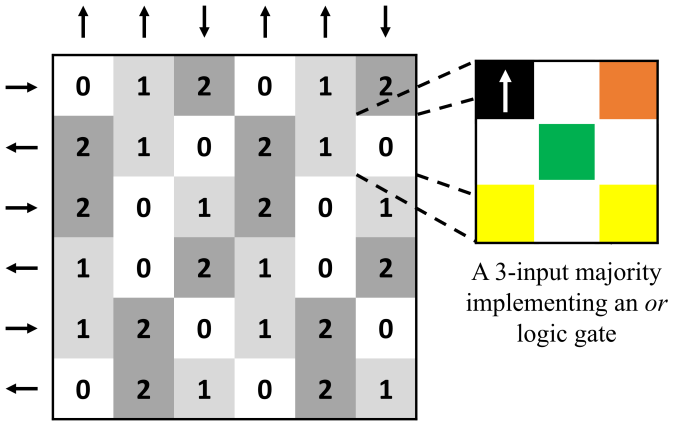


Figure 37 – BANCS grid and a BANCS cell containing a 3-input majority gate implementing an or logic gate.

5.2.4 NML Cells

Following what was done in the QCA chapter, in this section, some basic gate implementations are presented so they can be used as the basis for the solutions designed through our proposed approach. All the implementations were previously defined in (FORMIGONI; VILELA NETO; NACIF, 2018) and some of them were also physically implemented in the prototypes presented in (IMRE et al., 2006; VARGA et al., 2010, 2011).

Considering this, in Figure 38 we present instances of some NML cells: the 3-input majority gate (which is the basis for implementing the or and and gates) (a) and wires (b). Notice that the inverter gate is trivially implemented by positioning two devices with antiferromagnetic coupling.

5.3 SAT-based Method for NML Design

This section presents the formal method responsible for the NML circuit synthesis and it is organized as follows: Section 5.3.1 presents the input of the method; Section 5.3.2 defines the Boolean variables that represent the devices and wires placed in the layout; Section 5.3.3 presents the formulas for modeling the design constraints;

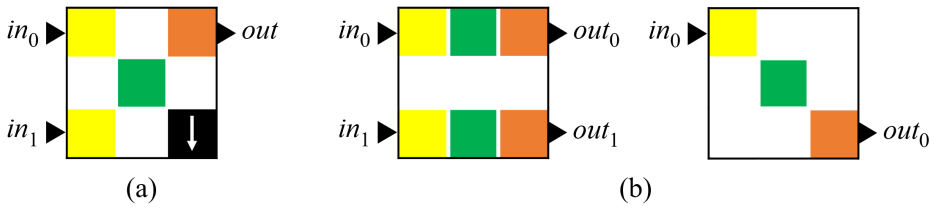


Figure 38 – Examples of NML gates adopted in our model. (a) 3-input majority gate implementing an and function. (b) Instances of wires in NML.

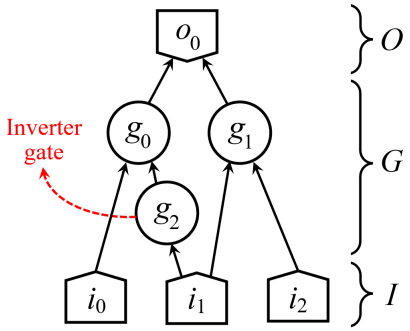


Figure 39 – A 2:1 multiplexer (2:1 MUX).

finally, Section 5.3.4 describes the proposed algorithm[2].

5.3.1 Input

Let N be the netlist circuit represented as a hypergraph $N = (V, E)$, such that V is the set of vertices, representing the gates, and E is the set of hyperedges, representing the wires. Figure 39 shows the hypergraph of a 2:1 multiplexer (2:1 MUX). Furthermore, the set of vertices of N can be partitioned into three disjoint subsets: input nodes $I = \{i_0, i_1, ..., i_n\}$, gate nodes $G = \{g_0, g_1, ..., g_m\}$, and output nodes $O = \{o_0, o_1, ..., o_p\}$, such that $V = I \cup G \cup O$.

Additionally to these definitions, the gate and output nodes are composed by majorty-based cells and inverter cells. Considering this, let us define the the set of majority gates as $M = \{m_0, m_1, ..., m_q\}$, such that $M \subseteq (G \cup O)$. Similarly to that, consider the set of inverter gates as $B = \{b_0, b_1, ..., b_r\}$, such that $B \subseteq (G \cup O)$. Notice that $M \cup B = G \cup O$. Some formulas presented in this section will make use of these definitions.

5.3.2 Boolean Variables

Two categories of Boolean variables are defined in our method: device variables and interface variables. In the next subsections we described these two types of variables.

5.3.2.1 Device Variables

Device variables represent the gates and wires positioned in the BANCS grid. These devices are part of the input, gate, or output sets.

In order to define the device variables, let us first consider (x, y) as a pair of coordinates of the BANCS grid. Thus, a gate variable $g_i(x, y)$ is responsible to represent a logic gate or wire of the node g_i and positioned in the coordinates (x, y) of the grid,

[2]Since there are some resemblances between the circuit modeling for NML and QCA, part of this section are similar to the Section 4.5 of this thesis. We decided to keep it as it is for a better and complete understanding of the proposed NML circuit modeling independently of the previous QCA synthesis chapter.

such that $g \in G$. Similar to that, we have the device variables $i_i(x, y)$ and $o_i(x, y)$ for representing the nodes of the input and output sets, such that $i \in I$ and $o \in O$[3].

Since device variables represent gates and wires in the circuit, then we have

$$g_i(x, y) \iff g_{i(d)}(x, y) \vee g_{i(w)}(x, y), \tag{52}$$

where $g_{i(d)}$ and $g_{i(w)}$ represent a driver (i.e., an input, a majority gate, or an inverter) and a wire of the circuit, respectively.

Additionally, a driver is either a majority or an inverter. This way, we have

$$g_{i(d)}(x, y) \iff m_{i(d)}(x, y) \vee b_{i(w)}(x, y), \tag{53}$$

such that

$$m_{i(d)}(x, y) \implies \neg b_{i(w)}(x, y) \tag{54}$$

and

$$b_{i(d)}(x, y) \implies \neg m_{i(w)}(x, y), \tag{55}$$

where $m \in M$ and $b \in B$.

Along with that, it is also important to mention that a gate $g_{i(d)}$ or a wire $g_{i(w)}$ operate under an specific clock cycle c. Thus, we have

$$g_{i(d)}(x, y) \iff \bigvee_{c \in C} g^c_{i(d)}(x, y), \tag{56}$$

$$g_{i(w)}(x, y) \iff \bigvee_{c \in C} g^c_{i(w)}(x, y), \tag{57}$$

$$g^c_i(x, y) \iff g^c_{i(d)}(x, y) \vee g^c_{i(w)}(x, y), \tag{58}$$

where $c \in C$ and $C = \{0, 1, ..., cycles - 1\}$.

5.3.2.2 Interface Variables

Interface variables are responsible to represent the connections between the cells of the BANCS grid, i.e., the interface ports of each USE cell.

Let us consider the BANCS cell of the coordinates (x, y) represented in Figure 40. This cell contains two interfaces on each one of its frontiers (west, north, east, and south) as shown in the figure. Let us define $R = \{ws, wn, ne, nw\}$ as the set of port locations on the west and north side of the cell, where $r \in R$. Thus, we can represent an interface variable as $r(x, y)$. Notice that the interface variables of the eastern and

[3]All the formulas presented in this chapter will be defined in terms of the gate variables in order to keep the modeling as simple as possible. However, it is important to notice that similar expressions are also defined for the input and output variables. All the exceptions for the previous statement are explicit in the text.

southern frontiers are shared with the neighbors BANCS cells placed in the right and the bottom of the given (x, y) cell, respectively.

Along with that, it is possible to have two different signals in a single BANCS cell, so we define a set of two signals $S = \{0, 1\}$, such that $s \in S$ and

$$r(x, y) \iff \bigvee_{s=0..1} r_{(s)}(x, y). \tag{59}$$

5.3.2.3 Variables Relationships

After defining the Boolean variables of our method, it is necessary to create the logical relationships between the device and interface variables.

a. *Driver variables and interface variables:* a driver cell $g_{i(d)}(x, y)$ containing n interfaces (considering $n = degree(g_i)$, i.e., the degree of the node g_i) should have at least n interface ports connected if $n = 1$ (inputs) and exactly n interface ports connected if $n = 2$ (inverters), $n = 3$ (and or or gates), or $n = 4$ (3-input majority gates).

   In order to formulate this relationship, let us assume $Z(x, y) = \{ws(x, y), wn(x, y), nw(x, y), ne(x, y), ws(x + 1, y), wn(x + 1, y), nw(x, y + 1), ne(x, y + 1)\}$ as the set of interface variables containing the ports placed on the borders of the cell (x, y). This way, we can derive

$$i_i(d)(x, y) \implies atleast(1, Z(x, y)), \tag{60}$$

   for the case of inputs,

$$m_i(d)(x, y) \implies exactly(n, Z(x, y)), \tag{61}$$

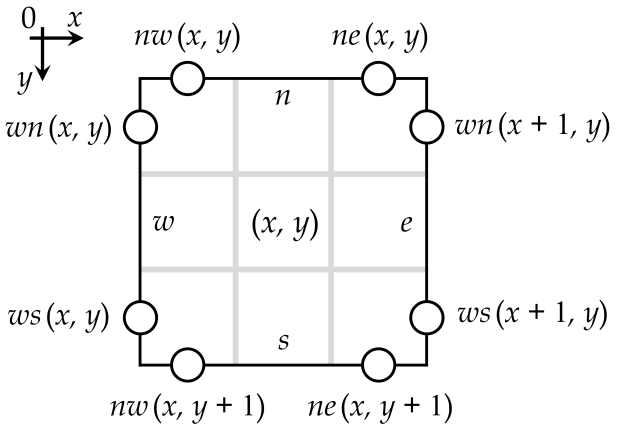


Figure 40 – Interfaces of a BANCS cell.

for the case of majority-based nodes, and

$$b_i(d)(x,y) \implies atleast(2, Z(x,y)), \tag{62}$$

for the case of the inverter nodes.

b. *Wire variables and interface variables:* if a wire is placed in a BANCS cell, then at least two interface ports should be connected, i.e., two ports should share the same signal.

To formalize this, consider $Z_{(s)}(x,y) = \{ws_{(s)}(x,y), wn_{(s)}(x,y), nw_{(s)}(x,y), ne_{(s)}(x,y), ws_{(s)}(x+1,y), wn_{(s)}(x+1,y), nw_{(s)}(x,y+1), ne_{(s)}(x,y+1)\}$ as a set similar to $Z(x,y)$ previous defined but adding the signal information to each element.

Formally,

$$g_{i(w)}(x,y) \implies \bigvee_{s=0..1} atleast(2, B_{(s)}(x,y)). \tag{63}$$

5.3.2.4 Table of Variables

Table 7 summarizes the variables defined in our model[4].

5.3.2.5 Number of Variables

The set of Boolean variables defined in our model is composed of device and interface variables. We enumerate these sets as follow:

a. The number of device variables is given by

$$\#D = \#cols * \#rows * \#nodes * (5 + 3 * \#cycles), \tag{64}$$

where $\#cols$ and $\#rows$ are the number of columns and rows of the BANCS grid, respectively, $\#nodes$ is the number of nodes on the input hypergraph N, and $\#cycles$ corresponds to the number of cycles of the solution attempt.

b. The number of interface variables is given by

$$\#I = 12 * \#cols * \#rows, \tag{65}$$

where $\#cols$ and $\#rows$ are the number of columns and rows of the BANCS grid, respectively.

[4]The input and output variables are omitted in Table 7 for simplicity reasons.

Table 7 – Table of variables used in the SAT-based method for NML.

Type	Variables	Description
Device	$g_i(x,y)$	Gate i is placed in the coordinates (x,y).
	$g_{i(d)}(x,y)$	Driver of the gate i is placed in (x,y).
	$g_{i(w)}(x,y)$	Wire of the gate i is placed in (x,y).
	$g_i^c(x,y)$	Gate i is placed in (x,y) and it operates in cycle c.
	$g_{i(d)}^c(x,y)$	Driver of the gate i is in (x,y) and it operates in cycle c.
	$g_{i(w)}^c(x,y)$	Wire of the gate i is in (x,y) and it operates in cycle c.
	$m_{i(d)}(x,y)$	Majority gate i is placed in (x,y).
	$b_{i(d)}(x,y)$	Inverter gate i is placed in (x,y).
Interface	$r(x,y)$	Port r of the (x,y) cell is connected.
	$r_{(s)}(x,y)$	Port r of (x,y) is connected and it contains the signal s.

5.3.3 Design Constraints

In this section we present the design constraints[5] for the synthesis of NML circuits through the SAT-based approach proposed. The constraints were derived from empirical observations based on what was presented in (FORMIGONI; VILELA NETO; NACIF, 2018), so there is no formal guarantee that the following model leads to an optimal solution in terms of area. These constraints are divided into three categories: placement (Section 5.3.3.1), synchronization (Section 5.3.3.2), and the routing constraints (Section 5.3.3.3). This subsection will detail these restrictions presenting the formulas that must be satisfied in order to have a valid functional circuit.

5.3.3.1 Placement Constraints

The placement constraints are responsible to define the restrictions for placing the cells (gates and wires) in the layout, not allowing invalid configurations regarding the placement.

a. *Single driver constraint:* each driver should be placed at exactly one location of the grid. Formally, we can define

$$exactly(1, \{g_{i(d)}(x_0, y_0), g_{i(d)}(x_1, y_1), ..., g_{i(d)}(x_{max}, y_{max})\}), \tag{66}$$

such that x_{max} and y_{max} are the maximum values for the coordinates x and y of the BANCS grid, respectively.

b. *Overlapping constraint:* if a majority-based gate is placed in a given BANCS cell, then no inverters or wires are allowed in this cell. Besides that, if an inverter-based gate is located in a given BANCS cell, then this cell can also allocate another inverter or wire. Finally, if a wire is in a given BANCS cell, then this cell can have another

[5] Appendix C presents an example of how the following constraints are applied.

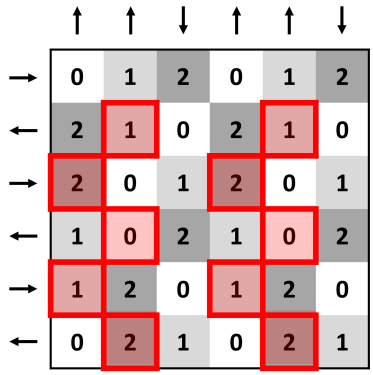


Figure 41 – Invalid coordinates for majority gate placement (red areas).

wire or inverter-based gate.

To define these constraints, we have

$$m_{i(d)}(x,y) \implies$$
$$exactly(1, \{g_{0(d)}(x,y), g_{1(d)}(x,y), ..., o_{0(d)}(x,y), o_{1(d)}(x,y), ...\}) \wedge$$
$$exactly(0, \{i_{0(w)}(x,y), i_{1(w)}(x,y), ..., g_{0(w)}(x,y), g_{1(w)}(x,y), ..., o_{0(w)}(x,y), o_{1(w)}(x,y), ...\}), \tag{67}$$

for the cases of the majority-based cells,

$$b_{i(d)}(x,y) \implies atmost(2, \{i_{0(d)}(x,y), i_{1(d)}(x,y), ..., b_{0(d)}(x,y), b_{1(d)}(x,y), ..., i_{0(w)}(x,y), i_{1(w)}(x,y), ..., g_{0(w)}(x,y), g_{1(w)}(x,y), ...\}), \tag{68}$$

for the cases of the inverter cells, and, finally,

$$i_{i(d)}(x,y) \implies atmost(2, \{i_{0(d)}(x,y), i_{1(d)}(x,y), ...\}), \tag{69}$$

for the case of input cells.

c. *Bottom majority constraint:* in order to define this constraint, let us first consider $phase(x,y)$ as a function that returns the phase of the cell located in the coordinates (x,y) of the BANCS grid. Based on this, a majority gate should not be placed in locations containing the same phase of the cell above. Fig 41 illustrates this restriction. The formula below creates this constraint:

$$\text{if } phase(x,y) = phase(x,y-1) \implies exactly(0, \{m_{0(d)}(x,y), m_{1(d)}(x,y), ...\}, \tag{70}$$

such that this is valid only for majority-based nodes (i.e., inverters are not included in this definition).

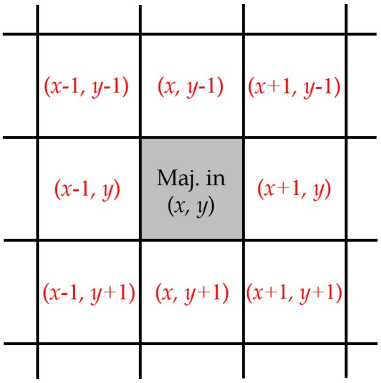


Figure 42 – Invalid positions (red coordinates) for majority gate placement around a majority-based gate located in (x, y).

d. *Majority near majority constraint:* if a majority gate is located in the BANCS coordinates (x, y), then all its neighbors should not contain another majority gate. Fig 42 shows the invalid positions nearby a majority-based gate placed in (x, y).

   To define this, we have

$$m_{i(d)}(x,y) \implies \bigwedge_{j=0..\#m,\ m_i \neq m_j} exactly(0, \{m_{j(d)}(x-1,y-1), m_{j(d)}(x,y-1), m_{j(d)}(x+1,y-1), m_{j(d)}(x-1,y), m_{j(d)}(x+1,y), m_{j(d)}(x-1,y+1), m_{j(d)}(x,y+1), m_{j(d)}(x+1,y+1)\}) \quad (71)$$

e. *Wire near majority constraint:* if a majority gate is located in (x, y), then the wires illustrated in Figure 43 are not allowed.

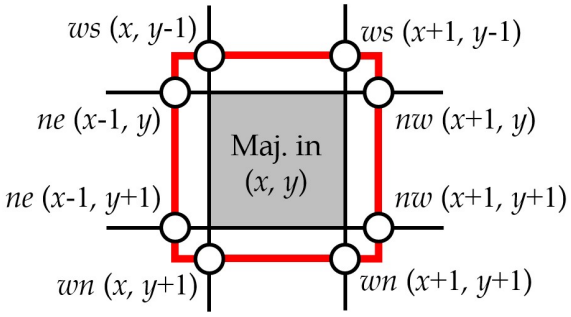


Figure 43 – Invalid wires around a majority gate placed in (x, y).

To define this, we have

$$
\begin{aligned}
m_{i(d)}(x,y) &\implies \bigwedge_{s=0..1} \neg(ws_{(s)}(x,y-1) \wedge ws_{(s)}(x+1,y-1)),\\
m_{i(d)}(x,y) &\implies \bigwedge_{s=0..1} \neg(ws_{(s)}(x+1,y-1) \wedge nw_{(s)}(x+1,y)),\\
m_{i(d)}(x,y) &\implies \bigwedge_{s=0..1} \neg(nw_{(s)}(x+1,y) \wedge nw_{(s)}(x+1,y+1)),\\
m_{i(d)}(x,y) &\implies \bigwedge_{s=0..1} \neg(nw_{(s)}(x+1,y+1) \wedge wn_{(s)}(x+1,y+1)),\\
m_{i(d)}(x,y) &\implies \bigwedge_{s=0..1} \neg(wn_{(s)}(x+1,y+1) \wedge wn_{(s)}(x,y+1)),\\
m_{i(d)}(x,y) &\implies \bigwedge_{s=0..1} \neg(wn_{(s)}(x,y+1) \wedge ne_{(s)}(x-1,y+1)),\\
m_{i(d)}(x,y) &\implies \bigwedge_{s=0..1} \neg(ne_{(s)}(x-1,y+1) \wedge ne_{(s)}(x-1,y)),\\
m_{i(d)}(x,y) &\implies \bigwedge_{s=0..1} \neg(ne_{(s)}(x-1,y) \wedge ws_{(s)}(x,y-1)).
\end{aligned}
\tag{72}
$$

f. *Wire near wire constraint:* a pair of wires should not be placed in the positions presented in Figure 44.

In order to formalize this constraint, consider the following equation for modelling the restriction presented in Figure 44 (a)

$$\neg(\bigwedge_{s=0..1} (ne_{(s)}(x,y) \wedge ne_{(s)}(x,y+1)) \wedge (nw_{(s')}(x+1,y) \wedge nw_{(s')}(x+1,y+1))), \tag{73}$$

such that $s' = 1$ if $s = 0$ and $s' = 0$ if $s = 1$, and

$$\neg(\bigwedge_{s=0..1} (ws_{(s)}(x,y-1) \wedge ws_{(s)}(x+1,y-1)) \wedge (wn_{(s')}(x,y) \wedge wn_{(s')}(x+1,y))), \tag{74}$$

for modeling the case presented in Figure 44 (b).

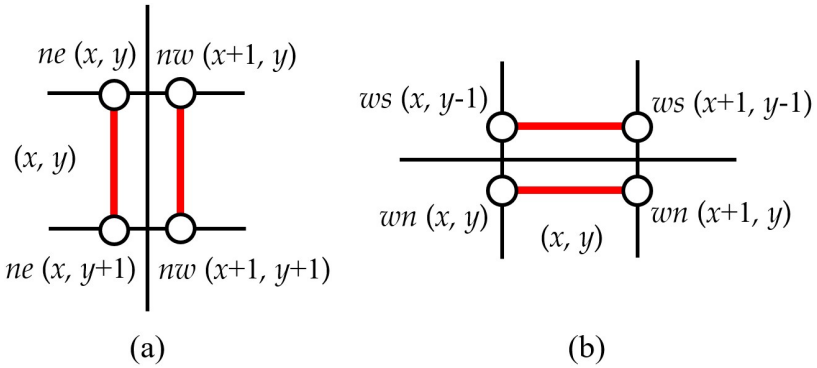


Figure 44 – Invalid pair of wires.

5.3.3.2 Synchronization Constraints

The synchronization constraints are responsible to ensure that all the inputs of a given cell arrive at the same time, as well as to synchronize the input and output of the data in the circuit.

a. *Driver synchronization constraint:* there are two scenarios to be defined: the first consists of drivers located in cells with the same phase of its southern neighbor; the second consists of drivers located in cells with a different phase of its southern neighbor. In other words, we have the following constraints:

   i. In the first scenario, where $phase(x, y) = phase(x, y+1)$, the inputs of the drivers should be submitted to the same clock cycle if the cell is placed on locations upon phases 1 or 2 and to the previous clock cycle if the cell is placed upon phase 0.

   ii. In the second scenario, where $phase(x, y) \neq phase(x, y + 1)$, the inputs of the drivers should be submitted to the same clock cycle if the cell is placed on locations upon phases 1 or 2 and to the previous or the same clock cycle if the cell is placed upon phase 0.

   To define the equations that implement the driver synchronization constraint, some auxiliary definitions are necessary:

   i. Consider a given gate $g_{(x)}$ such that its inputs are $g_{(y)}$ and $g_{(z)}$ (as shown in Figure 24 of the previous chapter);

   ii. Consider the different BANCS cell configurations presented in Figure 45 in which we can notice the different cell orientations (denoted by the arrows) and the neighborhood of the (x, y) location.

   Thus, considering $phase(x, y) \geq 1$, we have

$$\begin{aligned}
g^c_{x(d)}(x,y) &\implies (g^c_y(x,y-1) \vee g^c_y(x-1,y)) \wedge (g^c_z(x,y-1) \vee g^c_z(x-1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^c_y(x,y-1) \vee g^c_y(x+1,y)) \wedge (g^c_z(x,y-1) \vee g^c_z(x+1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^c_y(x,y+1) \vee g^c_y(x-1,y)) \wedge (g^c_z(x,y+1) \vee g^c_z(x-1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^c_y(x,y+1) \vee g^c_y(x+1,y)) \wedge (g^c_z(x,y+1) \vee g^c_z(x+1,y)),
\end{aligned} \tag{75}$$

   for the cases presented in Figure 45 (a), (b), (c), and (d), respectively. Similarly, considering $phase(x, y) = 0$, we have

$$\begin{aligned}
g^c_{x(d)}(x,y) &\implies (g^{c-1}_y(x,y-1) \vee g^{c-1}_y(x-1,y)) \wedge (g^{c-1}_z(x,y-1) \vee g^{c-1}_z(x-1,y)),\\
g^c_{x(d)}(x,y) &\implies (g^{c-1}_y(x,y-1) \vee g^{c-1}_y(x+1,y)) \wedge (g^{c-1}_z(x,y-1) \vee g^{c-1}_z(x+1,y)),
\end{aligned} \tag{76}$$

for the cases presented in Figure 45 (a) and (b), respectively. Along with that, we have

$$\begin{aligned} g^c_{x(d)}(x,y) &\implies (g^{c-1}_y(x,y+1) \vee g^{c-1}_y(x-1,y)) \wedge (g^c_z(x,y+1) \vee g^{c-1}_z(x-1,y)), \\ g^c_{x(d)}(x,y) &\implies (g^{c-1}_y(x,y+1) \vee g^{c-1}_y(x+1,y)) \wedge (g^c_z(x,y+1) \vee g^{c-1}_z(x+1,y)), \end{aligned} \tag{77}$$

for the cases shown in Figure 45 (c) and (d), such that $phase(x,y) \neq phase(x,y+1)$ (scenario i presented above). Finally, we define

$$\begin{aligned} g^c_{x(d)}(x,y) &\implies (g^c_y(x,y+1) \vee g^{c-1}_y(x-1,y)) \wedge (g^c_z(x,y+1) \vee g^{c-1}_z(x-1,y)), \\ g^c_{x(d)}(x,y) &\implies (g^c_y(x,y+1) \vee g^{c-1}_y(x+1,y)) \wedge (g^c_z(x,y+1) \vee g^{c-1}_z(x+1,y)), \end{aligned} \tag{78}$$

for the cases shown in Figure 45 (c) and (d), such that $phase(x,y) = phase(x,y+1)$ (scenario ii presented before).

b. *Wire synchronization constraint:* for the wire synchronization constraint, we make use of the same idea presented above. Thus, two scenarios should be considered

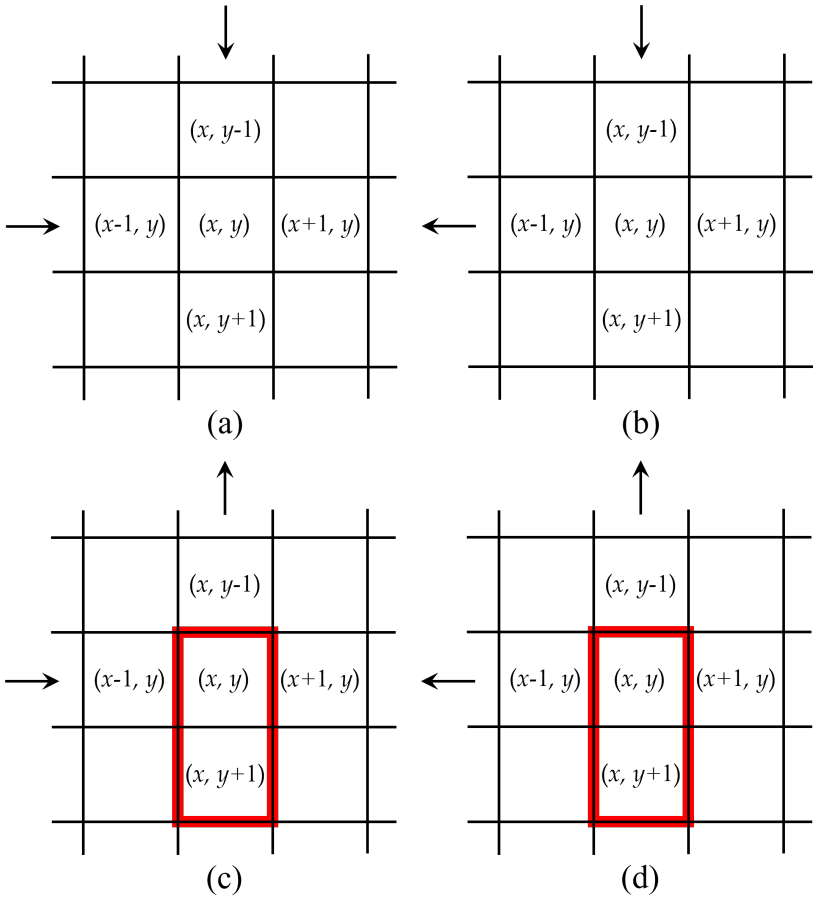


Figure 45 – Neighbors of the (x,y) BANCS cell, where the red areas can be submitted to the same or to different clock phases.

for the modeling:

i. In the first scenario, where $phase(x, y) = phase(x, y + 1)$, the inputs of a given wire (i.e., the driver that originates the wire or another instance of the same wire) should be submitted to the same clock cycle if the cell is placed on locations upon phases 1 or 2 and to the previous clock cycle if the cell is placed upon phase 0.

ii. In the second scenario, where $phase(x, y) \neq phase(x, y + 1)$, the inputs of a given wire should be submitted to the same clock cycle if the cell is placed on locations upon phases 1 or 2 and to the previous or the same clock cycle if the cell is placed upon phase 0.

To formalize this constraint, considering $phase(x, y) \geq 1$, we have

$$\begin{aligned} g^c_{x(w)}(x, y) &\implies g^c_x(x, y - 1) \vee g^c_x(x - 1, y)), \\ g^c_{x(w)}(x, y) &\implies g^c_x(x, y - 1) \vee g^c_x(x + 1, y)), \\ g^c_{x(w)}(x, y) &\implies g^c_x(x, y + 1) \vee g^c_x(x - 1, y)), \\ g^c_{x(w)}(x, y) &\implies g^c_x(x, y + 1) \vee g^c_x(x + 1, y)), \end{aligned} \tag{79}$$

for the cases presented in Figure 25 (a), (b), (c), and (d), respectively. Similarly, considering $phase(x, y) = 0$, we have

$$\begin{aligned} g^c_{x(w)}(x, y) &\implies g^{c-1}_x(x, y - 1) \vee g^{c-1}_x(x - 1, y)), \\ g^c_{x(w)}(x, y) &\implies g^{c-1}_x(x, y - 1) \vee g^{c-1}_x(x + 1, y)), \end{aligned} \tag{80}$$

for the cases presented in Figure 25 (a), (b), respectively. Along with that, we have

$$\begin{aligned} g^c_{x(w)}(x, y) &\implies g^{c-1}_x(x, y + 1) \vee g^{c-1}_x(x - 1, y)), \\ g^c_{x(w)}(x, y) &\implies g^{c-1}_x(x, y + 1) \vee g^{c-1}_x(x + 1, y)), \end{aligned} \tag{81}$$

for the cases shown in Figure 45 (c) and (d), such that $phase(x, y) \neq phase(x, y + 1)$ (scenario i presented above). Finally, we define

$$\begin{aligned} g^c_{x(w)}(x, y) &\implies g^c_x(x, y + 1) \vee g^{c-1}_x(x - 1, y)), \\ g^c_{x(w)}(x, y) &\implies g^c_x(x, y + 1) \vee g^{c-1}_x(x + 1, y)), \end{aligned} \tag{82}$$

for the cases shown in Figure 45 (c) and (d), such that $phase(x, y) = phase(x, y + 1)$ (scenario ii presented before).

c. *Single-cycle constraint:* each cell (driver or wire) should operate under exactly one

cycle. Thus, we have

$$g_i(x,y) \implies exactly(1, \{g_i^0(x,y), g_i^1(x,y), ..., g_i^{\#cycles-1}(x,y)\}). \tag{83}$$

d. *Input/output phase synchronization constraint:* each input should be submitted to the same phase. The same is valid for each output.

To define this constraint, first let us consider the following:

i. A vector $[(x_0^0, y_0^0), (x_1^0, y_1^0), ..., (x_k^0, y_k^0)]$ containing all the k coordinates submitted to phase 0;

ii. A vector $[(x_0^1, y_0^1), (x_1^1, y_1^1), ..., (x_l^1, y_l^1)]$ containing all the l coordinates submitted to phase 1;

iii. A vector $[(x_0^2, y_0^2), (x_1^2, y_1^2), ..., (x_m^2, y_m^2)]$ containing all the m coordinates submitted to phase 2.

Then, we have

$$i_{a(d)}(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{j=0..k} i_{b(d)}(x_j^0, y_j^0), \text{if } phase(x,y) = 0, \tag{84}$$

$$i_{a(d)}(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{j=0..l} i_{b(d)}(x_j^1, y_j^1), \text{if } phase(x,y) = 1, \tag{85}$$

$$i_{a(d)}(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{j=0..m} i_{b(d)}(x_j^2, y_j^2), \text{if } phase(x,y) = 2, \tag{86}$$

such that $\#i$ denotes the number of input nodes in the netlist. Furthermore, similar formulas are also defined for the case of the output nodes.

e. *Input/output cycle synchronization constraint:* each input should be submitted to the same clock cycle. The same is valid for each output. Formally, we have

$$i_{a(d)}^c(x,y) \implies \bigwedge_{\substack{b=0..\#i \\ i_a \neq i_b}} \bigvee_{\substack{x=0..x_{max} \\ y=0..y_{max}}} i_{b(d)}^c(x,y), \tag{87}$$

where $\#i$ denotes the number of input nodes in the netlist. A similar formula is also defined for the outputs.

5.3.3.3 Routing Constraints

The routing constraints are responsible for connecting all the elements placed in the BANCS grid. The following constraints describe the routing model designed.

a. *No return constraint:* a signal should not enter and exit the cell by the same side. Formally, we have

$$\bigwedge_{\substack{x=0..x_{max}\\y=0..y_{max}\\s=0..1}} \neg(ws_{(s)}(x,y) \wedge wn_{(s)}(x,y)) \ \wedge \bigwedge_{\substack{x=0..x_{max}\\y=0..y_{max}\\s=0..1}} \neg(nw_{(s)}(x,y) \wedge ne_{(s)}(x,y)). \tag{88}$$

b. *Border constraint:* ports located on the border of the routing area should not be connected. Formally,

$$\bigwedge_{x=0..x_{max}} \neg nw(x,0) \wedge \neg ne(x,0) \ \wedge \bigwedge_{y=0..y_{max}} \neg ws(0,y) \wedge \neg wn(0,y). \tag{89}$$

c. *Invalid corners constraint:* since each cell has a specific signal flow direction assigned to it (accordingly with the arrows located on the borders of the BANCS grid), then all the routing paths presented in the last chapter in Figure 26 are forbidden, i.e., the information could not propagate through the red paths shown in the figure.

   Following what is defined in Figure 26, we can formulate this restriction as follows:

$$\bigwedge_{s=0..1} \neg((nw_{(s)}(x,y+1) \vee ne_{(s)}(x,y+1)) \wedge (ws_{(s)}(x+1,y) \vee wn_{(s)}(x+1,y))) \tag{90}$$

   prevents the situation presented in (a) and (h),

$$\bigwedge_{s=0..1} \neg((nw_{(s)}(x,y) \vee ne_{(s)}(x,y)) \wedge (ws_{(s)}(x+1,y) \vee wn_{(s)}(x+1,y))) \tag{91}$$

   avoids the cases (b) and (g),

$$\bigwedge_{s=0..1} \neg((nw_{(s)}(x,y+1) \vee ne_{(s)}(x,y+1)) \wedge (ws_{(s)}(x,y) \vee wn_{(s)}(x,y))) \tag{92}$$

   deals with the cases (c) and (f), while

$$\bigwedge_{s=0..1} \neg((nw_{(s)}(x,y) \vee ne_{(s)}(x,y)) \wedge (ws_{(s)}(x,y) \vee wn_{(s)}(x,y))) \tag{93}$$

   implements the constraints for preventing (d) and (e) to happen.

d. *Majority access constraint:* in a cell containing a majority-based gate, two neighbor ports should not be connected if they are located in the corners of the BANCS cell. We presented the cases where this occurs in Figure 27 (a).

   To implement this constraint, we have

$$\begin{aligned} m_{i(d)}(x,y) \implies & \neg(wn(x,y) \wedge nw(x,y)) \wedge \neg(ne(x,y) \wedge wn(x+1,y)) \wedge \\ & \neg(ws(x,y) \wedge nw(x,y+1)) \ \wedge \neg(ws(x+1,y) \wedge ne(x,y+1)). \end{aligned} \tag{94}$$

e. *Gate driver adjacency constraint:* each gate (majority or an inverter) should be followed (accordingly with the orientation of the BANCS grid) by the cells connected to its output or by its wires and should be preceded by the cells connected to its inputs.

   For this constraint, consider what was presented in Figure 28, where g_y and g_z are the inputs of g_x and g_a is the output of g_x. This way, we have

$$\begin{aligned} g_{x(d)}(x,y) \implies & exactly(1, \{g_y(x,y-1), g_y(x-1,y)\}) \wedge \\ & exactly(1, \{g_z(x,y-1), g_z(x-1,y)\}), \\ g_{x(d)}(x,y) \implies & exactly(1, \{g_y(x,y-1), g_y(x+1,y)\}) \wedge \\ & exactly(1, \{g_z(x,y-1), g_z(x+1,y)\}), \\ g_{x(d)}(x,y) \implies & exactly(1, \{g_y(x,y+1), g_y(x-1,y)\}) \wedge \\ & exactly(1, \{g_z(x,y+1), g_z(x-1,y)\}), \\ g_{x(d)}(x,y) \implies & exactly(1, \{g_y(x,y+1), g_y(x+1,y)\}) \wedge \\ & exactly(1, \{g_z(x,y+1), g_z(x+1,y)\}), \end{aligned} \tag{95}$$

   for the grid configurations presented in Figure 45 (a), (b), (c), and (d), respectively. Along with that, we have

$$\begin{aligned} g_{x(d)}(x,y) \implies & exactly(1, \{g_{x(w)}(x,y+1), g_{x(w)}(x+1,y)\}) \wedge \\ & exactly(1, \{g_a(x,y+1), g_a(x+1,y)\}), \\ g_{x(d)}(x,y) \implies & exactly(1, \{g_{x(w)}(x,y+1), g_{x(w)}(x-1,y)\}) \wedge \\ & exactly(1, \{g_a(x,y+1), g_a(x-1,y)\}), \\ g_{x(d)}(x,y) \implies & exactly(1, \{g_{x(w)}(x,y-1), g_{x(w)}(x+1,y)\}) \wedge \\ & exactly(1, \{g_a(x,y-1), g_a(x+1,y)\}), \\ g_{x(d)}(x,y) \implies & exactly(1, \{g_{x(w)}(x,y-1), g_{x(w)}(x-1,y)\}) \wedge \\ & exactly(1, \{g_a(x,y-1), g_a(x-1,y)\}), \end{aligned} \tag{96}$$

   also for the cases presented in Figure 45 (a), (b), (c), and (d), respectively.

f. *Input driver adjacency constraint:* an input driver connected to a given gate should be followed (accordingly with the orientation of the BANCS grid) by one of its own wires or by this gate. Furthermore, if the input is connected to more than one gate, then it should be followed by one of its own wires or by at least one of these gates.

To define this constraint, consider an input i_a connected to g_x and an input i_b connected to g_y and g_z as previously presented in Figure 29. This way, considering $degree(i_a) = 1$, we have

$$\begin{aligned}
i_{a(d)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x+1,y), g_{x(d)}(x,y+1), g_{x(d)}(x+1,y)\}),\\
i_{a(d)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x-1,y), g_{x(d)}(x,y+1), g_{x(d)}(x-1,y)\}),\\
i_{a(d)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x+1,y), g_{x(d)}(x,y-1), g_{x(d)}(x+1,y)\}),\\
i_{a(d)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x-1,y), g_{x(d)}(x,y-1), g_{x(d)}(x-1,y)\}),
\end{aligned} \tag{97}$$

for the BANCS grid configurations presented in Figure 45 (a), (b), (c), and (d), respectively. Along with that, considering $degree(i_a) \geq 2$, we have

$$\begin{aligned}
i_{a(d)}(x,y) \implies & i_{a(w)}(x,y+1) \vee i_{a(w)}(x+1,y) \vee g_{y(d)}(x,y+1) \vee g_{y(d)}(x+1,y) \vee \\
& g_{z(d)}(x,y+1) \vee g_{z(d)}(x+1,y),\\
i_{a(d)}(x,y) \implies & i_{a(w)}(x,y+1) \vee i_{a(w)}(x-1,y) \vee g_{y(d)}(x,y+1) \vee g_{y(d)}(x-1,y) \vee \\
& g_{z(d)}(x,y+1) \vee g_{z(d)}(x-1,y),\\
i_{a(d)}(x,y) \implies & i_{a(w)}(x,y-1) \vee i_{a(w)}(x+1,y) \vee g_{y(d)}(x,y-1) \vee g_{y(d)}(x+1,y) \vee \\
& g_{z(d)}(x,y-1) \vee g_{z(d)}(x+1,y),\\
i_{a(d)}(x,y) \implies & i_{a(w)}(x,y-1) \vee i_{a(w)}(x-1,y) \vee g_{y(d)}(x,y-1) \vee g_{y(d)}(x-1,y) \vee \\
& g_{z(d)}(x,y-1) \vee g_{x(d)}(x-1,y),
\end{aligned} \tag{98}$$

also for the cases shown in Figure 45 (a), (b), (c), and (d), respectively.

g. *Wire adjacency constraint:* a wire should be followed (accordingly with the orientation of the BANCS grid) by another instance of the same wire or by the gates connected to its output. Along with that, the wire should be preceded by another instance of the same wire or by the gate that originates this wire.

To define this constraint, consider an input i_a connected to the gate g_x. This way,

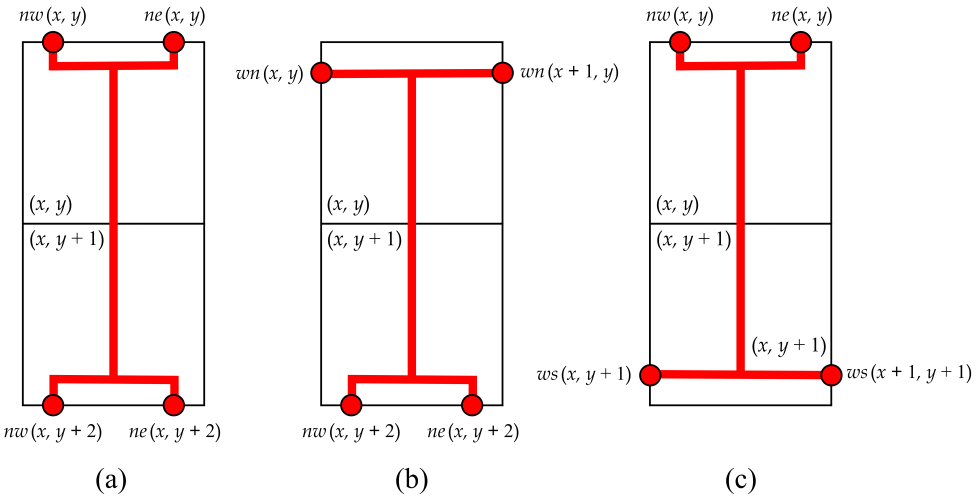


Figure 46 – Invalid routing scenarios: long wires (more then five magnets) submitted to the same clock phase.

we have

$$
\begin{aligned}
i_{a(w)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x+1,y), g_{x(d)}(x,y+1), g_{x(d)}(x+1,y)\}),\\
i_{a(w)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x-1,y), g_{x(d)}(x,y+1), g_{x(d)}(x-1,y)\}),\\
i_{a(w)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x+1,y), g_{x(d)}(x,y-1), g_{x(d)}(x+1,y)\}),\\
i_{a(w)}(x,y) &\implies exactly(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x-1,y), g_{x(d)}(x,y-1), g_{x(d)}(x-1,y)\}),
\end{aligned}
\tag{99}
$$

for the cases shown in Figure 45 (a), (b), (c), and (d), respectively. Along with that, we have

$$
\begin{aligned}
i_{a(w)}(x,y) &\implies atleast(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x-1,y), i_{a(d)}(x,y-1), i_{a(d)}(x-1,y)\}),\\
i_{a(w)}(x,y) &\implies atleast(1, \{i_{a(w)}(x,y-1), i_{a(w)}(x+1,y), i_{a(d)}(x,y-1), i_{a(d)}(x+1,y)\}),\\
i_{a(w)}(x,y) &\implies atleast(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x-1,y), i_{a(d)}(x,y+1), i_{a(d)}(x-1,y)\}),\\
i_{a(w)}(x,y) &\implies atleast(1, \{i_{a(w)}(x,y+1), i_{a(w)}(x+1,y), i_{a(d)}(x,y+1), i_{a(d)}(x+1,y)\}),
\end{aligned}
\tag{100}
$$

also considering the cases defined in Figure 45 (a), (b), (c), and (d), respectively.

h. *Long wire constraint:* wires should not be composed of more than five magnets submitted to the same phase in order to prevent errors caused by thermal noise. Figure 46 presents the problematic scenarios.

   To implement this constraint, we first consider the vertical neighbor cells with same

phase, i.e., $phase(x, y) = phase(x, y + 1)$. Thus, we have

$$g_{i(w)}(x, y) \wedge g_{i(w)}(x, y + 1) \implies \bigwedge_{s=0..1} \neg((nw_{(s)}(x, y) \vee ne_{(s)}(x, y)) \wedge (nw_{(s)}(x, y + 2) \vee ne_{(s)}(x, y + 2))) \quad (101)$$

for modeling the restriction presented in Figure 46 (a),

$$g_{i(w)}(x, y) \wedge g_{i(w)}(x, y + 1) \implies \bigwedge_{s=0..1} \neg((wn_{(s)}(x, y) \vee wn_{(s)}(x + 1, y)) \wedge (nw_{(s)}(x, y + 2) \vee ne_{(s)}(x, y + 2))) \quad (102)$$

for modeling the restriction (b), and

$$g_{i(w)}(x, y) \wedge g_{i(w)}(x, y + 1) \implies \bigwedge_{s=0..1} \neg((nw_{(s)}(x, y) \vee ne_{(s)}(x, y)) \wedge (ws_{(s)}(x, y + 1) \vee ws_{(s)}(x + 1, y + 1))) \quad (103)$$

for modeling the restriction (c).

5.3.3.4 Table of Design Constraints

Table 8 summarizes the set of constraints for NML synthesis.

5.3.4 Algorithm

This section presents the proposed method (Algorithm 5) for generating area-optimized synchronized NML circuits. The algorithm is based on the variables and constraints defined in the previous section of this chapter[6].

In this algorithm, consider $orientation$ as a vector containing the possible orientations for the grid origin (coordinate $(0, 0)$) and $phases$ as a vector containing the phases of the origin. Along with that, consider `Factor(`$area$`)` as a function that returns a vector $[(r_0, c_0), (r_1, c_1), ..., (r_n, c_n)]$ which contains all the n combinations of r and c (rows and columns) such that $r_i * c_i = area$. Moreover, the functions `Placement(...)`, `Synchronization(...)`, and `Routing(...)` are responsible for creating the clauses described in Sections 5.3.3.1, 5.3.3.2, and 5.3.3.3, respectively. Finally, the function `Translate(...)` transforms the result obtained by the SAT solver (the set of variables assigned to $True$ on the solution) into the NML circuit.

[6] As in the previous chapters, the set of formulas presented are not necessarily in the CNF format, so it is necessary to apply the Tseitin transformation (TSEITIN, 1983) in order to convert the formulas to CNF.

Table 8 – Table of design constraints: NML synthesis.

Type	Constraint	Description
Plac.	Single driver	Each driver should be placed at exactly one location.
	Overlapping	Each grid cell contain at most one majority or (exclusively) at most two inverters/wires.
	Bottom majority	Majority gates should not be placed in locations containing the same phase of the cell above.
	Maj. near maj.	Majority gates should not be placed alongside other majorities.
	Wire near maj.	Wires should not be placed in some tracks alongside a majority gate (see Figure 43).
	Wire near wire	Wires should not be placed in some tracks alongside other wires (see Figure 44).
Sync.	Driver sync.	For each gate or output driver, its inputs should be submitted to the same clock cycle if the cell is on phases 1 or 2 and to the previous or the same clock cycle if the cell is on phase 0 (depending on the position of the driver).
	Wire sync.	For each wire cell, its input should be submitted to the same clock if the cell is submitted to phase 1 or 2 and to the previous or the same clock cycle if the cell is submitted to phase 0 (depending on the position of the driver).
	Single-cycle	Each cell should operate under exactly one cycle.
	I/O phase sync.	Each input should be submitted to the same phase. The same is valid for each output.
	I/O cycle sync.	Each input should operate on the same clock cycle. The same is valid for each output.
Rout.	No return	A signal should not enter and exit a cell by the same side.
	Border	Ports located on the border of the routing area should not be connected.
	Invalid corners	Some routing paths are not allowed (see Figure 26).
	Majority access	In a cell containing a majority gate, two neighbor ports should not be connected if they are located in the corners of this BANCS cell.
	Gate driver adj.	Each gate should be followed by the cells connected to its output or by its wires and should be preceded by the cells connected to its inputs.
	Input driver adj.	Each input driver connected to one gate should be followed by one of its own wires or by the gate. Furthermore, if the input is connected to more than one gate, then it should be followed by one of its own wires or by at least one of the gates connected to its output.
	Wire adj.	Each wire should be followed by the cells connected to its output or by other wire of the same signal and should be preceded by a wire or by a driver of the same signal.
	Long wire	Wires should not be composed of more than five magnets on the same phase.

Algorithm 5: SAT-based pseudo-code for the synthesis of NML circuits

```
input  : input hypergraph N, number of cycles c
output: NML-BANCS design circuit
orientation ← (⇩,⇦,⇧,⇨)
phases ← (0, 1, 2)
area ← Nodes(N)
while True do
    combinations ← Factor(area)
    for i ← 0 to combinations.size do
        rows, cols ← combinations(i)
        for o ← 0 to orientation.size do
            for p ← 0 to phases.size do
                V ← CreateBooleanVariables(N, c, rows, cols)
                placement ← Placement(N, V, c, rows, cols, orientation(o), phases(p))
                sync ← Synchronization(N, V, c, rows, cols, orientation(o), phases(p))
                routing ← Routing(N, V, c, rows, cols, orientation(o), phases(p))
                SAT ← Solver(placement, sync, routing)
                if SAT ≠ False then
                    circuit ← Translate (SAT)
                    return circuit
    area ← area + 1
```

5.4 Experiments

The experiments presented in this section follow the same idea of the previous chapter (Section 4.6): two assessments were conducted over a well-known benchmark of digital circuits, where the first consists of exploring different synchronization profiles, while the second aims to compare the solutions obtained with (FORMIGONI et al., 2021). It is important to point out that the NMLSim (SOARES et al., 2018) tool was used to validate all the circuits designed.

5.4.1 Assessment of Synchronization Profiles

The first experiment explores two synchronization profiles (such that both ensure the global synchronicity of the circuit based on the design principles described in (TORRES et al., 2018)): (i) synchronization at clock phase level and (ii) synchronization at clock cycle level. These different profiles are detailed below:

i. Synchronization at clock phase level: in this scenario, all the inputs are submitted to the same phase and to the same cycle (the same is valid for the outputs). This allows receiving new data on each clock phase (i.e., achieving the maximum throughput). For generating synchronized circuits at the clock phase level we include in our formulation all the synchronization restrictions defined in Section 5.3.3.2;

ii. Synchronization at clock cycle level: in this scenario, all the inputs and outputs of the circuit are submitted to the same cycle (the same is valid for the outputs), i.e. there are no restrictions regarding the phase in which the inputs and outputs

Table 9 – Comparison between different synchronicity profiles: NML synthesis.

Circuit	#G	#I	#O	Phase synchronicity			Cycle synchronicity		
				Area	Lat.	Occ. (%)	Area	Lat.	Occ. (%)
2:1 MUX	3	3	1	3×3	4	100.0	3×3	3	88.9
XOR2	4	2	1	3×3	3	100.0	3×3	3	100.0
Full adder	7	3	2	10×6	12	76.7	11×5	13	74.5
C17	5	5	2	9×5	8	87.5	7×5	10	88.6
Par. gen.	9	3	1	11×9	20	64.6	11×9	20	64.6
Par. ch.	14	4	1	8×7	12	78.6	8×7	12	78.6
Decoder	5	2	1	7×5	5	54.3	10×3	7	73.3

are submitted. This scenario ensures that new data can be computed on each clock cycle. For generating synchronized circuits at the clock cycle level we did not include in our formulation the restriction defined in Section 5.3.3.2d.

The results of the first experiment are summarized in Table 9, where *#G*, *#I*, and *#O* refers to the number of gates, inputs, and outputs of the netlists, respectively, *Area* is the area of the resulting layout in terms of BANCS cells, *Lat.* is the latency of the critical path of the circuit (in terms of the number of clock phases), and *Occ. (%)* is the occupancy rate of the BANCS grid.

As we can see in Table 9, the results corroborate with the ones presented in the previous chapter: the cells with synchronization at clock cycle level are (on average) optimized in terms of area in comparison with the cells with synchronization at clock phase level. However, it is important to notice that there is a tradeoff between area and throughput since the latter operates under the maximum throughput (as broadly discussed in (TORRES et al., 2018)). This way is up to the designer to decide which version of the cell is suitable for a given specification.

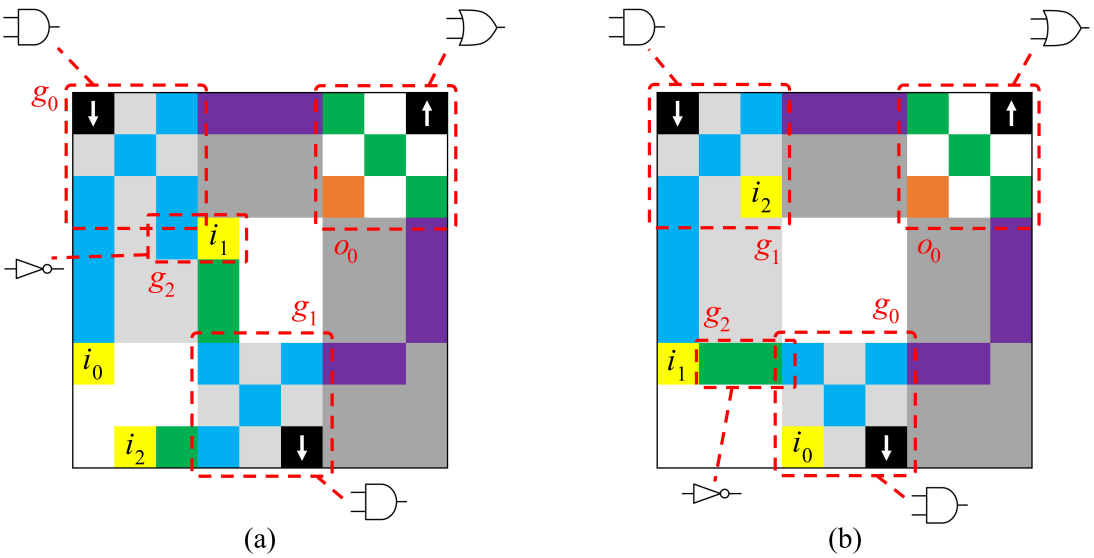


Figure 47 – 2:1 MUX circuits. (a) Solution with clock phase synchronicity. (b) Solution with clock cycle synchronicity

Table 10 – Comparison with (FORMIGONI et al., 2021) methodology.

Circuit	#G	#I	#O	Proposed method			Formigoni *et al.*		
				Area	Lat.	Occ. (%)	Area	Lat.	Occ. (%)
2:1 MUX	3	3	1	3×3	4	100.0	5×6	7	60.0
XOR2	4	2	1	3×3	3	100.0	6×7	9	56.7
Full adder	7	3	2	10×6	12	76.7	-	-	-
C17	5	5	2	9×5	8	87.5	-	-	-
P. gen.	9	3	1	11×9	20	64.6	15×12	24	60.0
P. ch.	14	4	1	8×7	12	78.6	12×15	21	51.6
Decoder	5	2	1	7×5	5	54.3	5×6	5	83.3

To illustrate these two approaches, consider the 2:1 MUX presented in Figure 47 where (a) is the solution with clock phase synchronization and (b) is the circuit with clock cycle synchronization. Notice that, besides achieving the same circuit area for both cases, (b) has 12.1% fewer magnets in comparison with (a). On the other hand, (a) achieves greater throughput than (b).

5.4.2 Comparison with another Methodology

The second experiment is a comparison between the solutions produced through the proposed methodology and the solutions produced in (FORMIGONI et al., 2021). It is important to notice that (FORMIGONI et al., 2021) presented only solutions with clock phase synchronicity, so we made the comparison of this profile in order to provide a fair assessment. The results of the second experiment are summarized in Table 10.

Considering the data presented in Table 10, the proposed approach can produce cells with less area in comparison with (FORMIGONI et al., 2021) for all cases except for the Decoder circuit. A side effect of this area optimization is the gain in latency and occupancy rate, producing faster and denser solutions in general.

To illustrate the differences between the solutions produced through the proposed methodology and the (FORMIGONI et al., 2021) approach, Figure 48 shows the different versions of the XOR2 gate. Furthermore, it is important to notice that our solution does not have any crossing wire, while (FORMIGONI et al., 2021) make use of one crossing segment.

5.5 Chapter Conclusions

In this chapter we present a new SAT-based methodology for generating NML circuits. A set of Boolean variables were defined to represent the states of the problem, while formulas were described to model the restrictions intrinsic to the NML design. All the design rules were derived from empirical observations of circuits designed in the BANCS clocking scheme, the structure responsible to encode the restrictions regarding the synchronization of the solution.

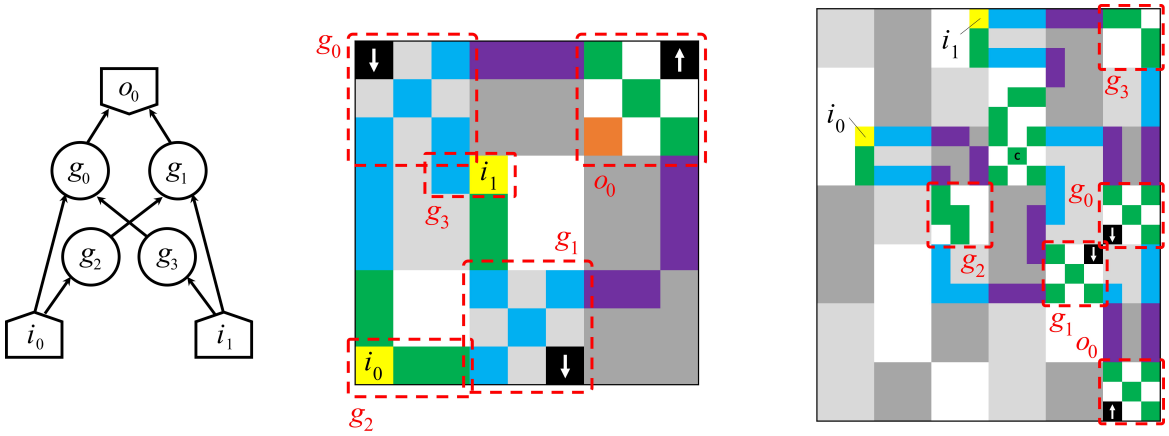


Figure 48 – Versions of the XOR2 gate in NML. (a) Input netlist. (b) Proposed solution. (c) Solution obtained through (FORMIGONI et al., 2021).
Source: (a) and (b) are both original, while (c) is adapted from (FORMIGONI et al., 2021).

The results of the experiments show that the proposed methodology can achieve better results in terms of circuit area when compared to a graph-based algorithm. Along with that, the latency of the solutions was also optimized, producing faster circuits in general. Another important result is related to the different synchronization profiles, where we described approaches with a tradeoff between area and throughput. Thus, is up to the designer which approach to take to meet their specifications.

Finally, the proposed methodology can be applied for several purposes such as the design of NML standard cell libraries, for the on-the-fly design of NML circuits, for the integration with divide-and-conquer strategies to provide scalability, for validation and comparison with other new methodologies, among others.

6 CONCLUSIONS

This thesis explores the capabilities of satisfiability solvers to manage the challenges inherent to the cell design task in different technologies. Nowadays, these tools can compute complex instances containing millions of variables and clauses, thus being an important asset to modern EDA solutions. In this scenario, we propose new satisfiability-based models to encode the design constraints of conventional and emerging technologies focusing on obtaining area-optimized solutions.

The first proposed method is focused on the transistor placement for static CMOS complex gates. The placement task plays a crucial role in cell synthesis since it is a process executed before the routing and compaction routines. Thus, obtaining a good placement potentially enables to have better layouts (in terms of area) at the end of the design flow. In this sense, the restrictions of the 65 nm technology node were formally defined and the satisfiability solver was able to find solutions with fewer columns in comparison with the placement routine implemented into ASTRAN, the automatic design tool used as the baseline for the experiments. The results also show that the layouts produced by the proposed design method not only presented optimization in area but also in the number of contacts, in general.

The second proposed method focus on the design for quantum-dot cellular automata, an emerging technology of the field-coupled nanotechnology family. The design in QCA must fulfill constraints in three domains: placement, routing, and synchronization. Thus, based on the clocking scheme called USE, we identified the design restrictions in QCA in order to formalize the discrete constraint model. Through a satisfiability solver, we were able to find solutions with optimization in area in comparison with other graph-based approaches. Furthermore, as a byproduct, the method also optimizes the latency of the cells. Finally, we proposed two design scenarios to cover different specification profiles, where a tradeoff between area and throughput is presented.

The third and last proposed approach performs cell synthesis for nanomagnetic logic, another field-coupled nanotechnology. As in the QCA design, placement, routing, and synchronization constraints must be taken into account for defining the restriction model. Thus, we employ the clocking scheme BANCS to formalize the design rules by

this clock template. Through a satisfiability solver, we find solutions that fulfill all the design restrictions in which, in general, the cells generated presented an optimization in area when compared to another design graph-based methodology. We also achieved faster solutions, i.e., the latency aspect was also optimized as a side effect of the area minimization. The tradeoff between area and delay was also introduced as in the QCA approach.

We can notice that the research questions raised in Section 1.1 of this thesis were successfully answered based on the hypotheses presented in the same section. We were able to model the design restrictions of each technology and make use of the satisfiability solvers capabilities to find good quality solutions in comparison with other traditional and well-established approaches. Moreover, the proposed methodologies can be used as a baseline in new researches in the area of the SCCG design as well as in the QCA and NML synthesis.

6.1 Future Work

Several new directions can be explored using the propositions, tools, and methods proposed in this thesis. We list them as follows:

- Adoption of newest technology nodes: even though the 65 nm technology is still adopted in the industry, new technologies present several design rules which are suitable to be formalized in a restriction model similar to what was done in this thesis.

- Improvements on ASTRAN: the design tool employed in this thesis presented several limitations, especially in its routing module. A satisfiability-based model which integrates the placement and routing procedures must improve the solutions found by ASTRAN in comparison with the proposed methodology.

- Assessment of clocking schemes: the proposed methods for QCA and NML can be adapted for different clocking schemes, thus being suitable for comparing these clocking schemes under a similar platform.

- Proposition of new clocking schemes: following the same principle of simplification on the design constraints, new clocking schemes may be a good alternative to provide relaxed restrictions, improving the capacities of the proposed approach.

- Design for fast QCA and NML cells: even though we were able to find solutions with optimizations in latency, a proper SAT-based method with a focus on this parameter may be useful when this circuit profile is desirable.

- Design of QCA and NML cell libraries: the design of cell libraries can be optimized using the proposed methods so they can be employed as core pieces of complex digital systems.
- Exploration of SAT in other emerging technologies: there are several emerging technologies - photonics and synthetic biological circuit, for instance - that can be explored using SAT to optimize its solutions in different aspects.

REFERENCES

AMD. **AMD Ryzen™ 9 5950X Desktop Processors**. Available: https://www.amd.com/en/products/cpu/amd-ryzen-9-5950x. Acessed in: Nov. 2021.

ANDERSON, N. G.; BHANJA, S. **Field-coupled Nanocomputing**: Paradigms, Progress, and Perspectives. first.ed. New York: Springer, 2014.

AQAJARI, S. A. H. et al. pyEDA: An Open-Source Python Toolkit for Pre-processing and Feature Extraction of Electrodermal Activity. **Procedia Computer Science**, [S.l.], v.184, p.99–106, 2021.

BIERE, A.; HEULE, M. **SAT Competition Winners on the SC2020 Benchmark Suite**. Available: http://fmv.jku.at/kissat/. Acessed in: Jan. 2022.

BROWN, C. E. Reducing Higher-Order Theorem Proving to a Sequence of SAT Problems. In: Proceedings of the 23rd International Conference on Automated Deduction, 2011. **Anais...** Springer International Publishing, 2011. p.147–161. (Lecture Notes in Computer Science, v.6803).

CAMPOS, C. A. T. **A Feasible Clocking Scheme (USE) and a Standard Cells Library (QCA ONE) for Future Quantum-dot Cellular Automata**. 2015. 101p. Dissertation — Federal University of Minas Gerais, Belo Horizonte, Minas Gerais, Brazil.

CAMPOS, C. A. T.; MARCIANO, A. L.; NETO, O. P. V.; TORRES, F. S. USE: A Universal, Scalable, and Efficient Clocking Scheme for QCA. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.35, n.3, p.513–517, 2016.

CARDOSO, M. S. et al. Transistor placement strategies for non-series-parallel cells. In: IEEE 60TH INTERNATIONAL MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS (MWSCAS), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.523–526.

CARDOSO, M. S. et al. Libra: An Automatic Design Methodology for CMOS Complex Gates. **IEEE Transactions on Circuits and Systems II: Express Briefs**, [S.l.], v.65, n.10, p.1345–1349, 2018.

CHEN, C.-A.; GUPTA, S. A satisfiability-based test generator for path delay faults in combinational circuits. In: DESIGN AUTOMATION CONFERENCE PROCEEDINGS, 1996, 33., 1996. **Anais...** [S.l.: s.n.], 1996. p.209–214.

CLARKE, E.; KROENING, D.; LERDA, F. A tool for checking ANSI-c programs. **Lecture Notes in Computer Science**, [S.l.], v.2988, p.168–176, 01 2004.

CORREIA, V. P.; REIS, A. I.; PORTO, C.; BRASIL, A. R. Classifying n-Input Boolean Functions. In: PROC. IWS, 2001. **Anais...** [S.l.: s.n.], 2001. p.58.

CORTADELLA, J. Area-Optimal Transistor Folding for 1-D Gridded Cell Design. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.32, n.11, p.1708–1721, 2013.

CORTADELLA, J.; PETIT, J.; GóMEZ, S.; MOLL, F. A Boolean Rule-Based Approach for Manufacturability-Aware Cell Routing. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.33, n.3, p.409–422, 2014.

CSABA, G. et al. Nanocomputing by field-coupled nanomagnets. **IEEE Transactions on Nanotechnology**, [S.l.], v.1, n.4, p.209–213, 2002.

CSABA, G.; POROD, W. Behavior of Nanomagnet Logic in the presence of thermal noise. In: INTERNATIONAL WORKSHOP ON COMPUTATIONAL ELECTRONICS, 2010., 2010. **Anais...** [S.l.: s.n.], 2010. p.1–4.

DEVADAS, S. Optimal layout via Boolean satisfiability. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN. DIGEST OF TECHNICAL PAPERS, 1989., 1989. **Anais...** [S.l.: s.n.], 1989. p.294–297.

DUECK, G.; SCHEUER, T. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. **Journal of Computational Physics**, [S.l.], v.90, n.1, p.161–175, 1990.

EéN, N.; SöRENSSON, N. An Extensible SAT-solver. In: SAT, 2003. **Anais...** Springer, 2003. p.502–518. (Lecture Notes in Computer Science, v.2919).

FONTES, G. et al. Placement and Routing by Overlapping and Merging QCA Gates. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–5.

FORMIGONI, R. E. et al. Evaluating nanomagnetic logic circuit layouts using different clock schemes. **Analog Integrated Circuits and Signal Processing**, [S.l.], v.106, n.1, p.205–218, Jan 2021.

FORMIGONI, R. E.; VILELA NETO, O. P.; NACIF, J. A. M. BANCS: Bidirectional Alternating Nanomagnetic Clocking Scheme. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–6.

FUJITA, M.; TAGUCHI, N.; IWATA, K.; MISHCHENKO, A. Incremental ATPG methods for multiple faults under multiple fault models. In: SIXTEENTH INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN, 2015. **Anais...** [S.l.: s.n.], 2015. p.177–180.

GANAI, M.; GUPTA, A. **SAT-Based Scalable Formal Verification Solutions** / **M. Ganai, A. Gupta.** [S.l.: s.n.], 2007.

GOLUMBIC, M.; MINTZ, A.; ROTICS, U. An improvement on the complexity of factoring read-once Boolean functions. **Discrete Applied Mathematics**, [S.l.], v.156, p.1633–1636, 05 2008.

GOMES, C. P.; KAUTZ, H.; SABHARWAL, A.; SELMAN, B. Chapter 2 Satisfiability Solvers. In: van Harmelen, F.; LIFSCHITZ, V.; PORTER, B. (Ed.). **Handbook of Knowledge Representation**. [S.l.]: Elsevier, 2008. p.89–134. (Foundations of Artificial Intelligence, v.3).

GOSWAMI, M. et al. An Efficient Clocking Scheme for Quantum-dot Cellular Automata. **International Journal of Electronics Letters**, [S.l.], 2019.

GRAZIANO, M.; VACCA, M.; CHIOLERIO, A.; ZAMBONI, M. An NCL-HDL Snake-Clock-Based Magnetic QCA Architecture. **IEEE Transactions on Nanotechnology**, [S.l.], v.10, n.5, p.1141–1149, 2011.

HANDBOOK OF SATISFIABILITY, 2009. **Anais...** IOS Press, 2009. (Frontiers in Artificial Intelligence and Applications, v.185).

IIZUKA, T.; IKEDA, M.; ASADA, K. High speed layout synthesis for minimum-width CMOS logic cells via Boolean satisfiability. In: ASP-DAC 2004: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE 2004 (IEEE CAT. NO.04EX753), 2004. **Anais...** [S.l.: s.n.], 2004. p.149–154.

IIZUKA, T.; IKEDA, M.; ASADA, K. Exact Minimum-Width Transistor Placement without Dual Constraint for CMOS Cells. In: ACM GREAT LAKES SYMPOSIUM ON VLSI, 15., 2005, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2005. p.74–77. (GLSVLSI '05).

IMRE, A. et al. Majority Logic Gate for Magnetic Quantum-Dot Cellular Automata. **Science**, [S.l.], v.311, n.5758, p.205–208, 2006.

INTEL. **Intel® Stratix® 10 GX 10M FPGA**. Available: https://ark.intel.com/content/www/br/pt/ark/products/210290/intel-stratix-10-gx-10m-fpga.html. Acessed in: Nov. 2021.

JACKSON, D.; SCHECHTER, I.; SHLYAKHTER, I. Alcoa: the Alloy constraint analyzer. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. ICSE 2000 THE NEW MILLENNIUM, 2000., 2000. **Proceedings...** [S.l.: s.n.], 2000. p.730–733.

JIANG, J.-H. R.; LEE, C.-C.; MISHCHENKO, A.; HUANG, C.-Y. To SAT or Not to SAT: Scalable Exploration of Functional Dependency. **IEEE Transactions on Computers**, [S.l.], v.59, n.4, p.457–467, 2010.

KAGARIS, D.; HANIOTAKIS, T. A Methodology for Transistor-Efficient Supergate Design. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [S.l.], v.15, n.4, p.488–492, 2007.

KARMAZIN, R.; OTERO, C. T. O.; MANOHAR, R. cellTK: Automated Layout for Asynchronous Circuits with Nonstandard Cells. In: IEEE 19TH INTERNATIONAL SYMPOSIUM ON ASYNCHRONOUS CIRCUITS AND SYSTEMS, 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p.58–66.

KAUTZ, H.; SELMAN, B. Planning as Satisfiability. In: 2013 , 1992. **Anais...** [S.l.: s.n.], 1992. p.359–363.

KHURSHID, S.; MARINOV, D. TestEra: Specification-Based Testing of Java Programs Using SAT. **Autom. Softw. Eng.**, [S.l.], v.11, p.403–434, 10 2004.

KIM, K.; WU, K.; KARRI, R. Quantum-Dot Cellular Automata Design Guideline. **IEICE Trans. Fundam. Electron. Commun. Comput. Sci.**, USA, v.E89-A, n.6, p.1607–1614, June 2006.

LANDAUER, R. **Is Quantum Mechanics Useful?** Dordrecht: Springer Netherlands, 1995. 237–240p.

LEE, R.-R.; JIANG, J.-H. R.; HUNG, W.-L. Bi-decomposing large Boolean functions via interpolation and satisfiability solving. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 2008., 2008. **Anais...** [S.l.: s.n.], 2008. p.636–641.

LENT, C. S.; SNIDER, G. L. **The Development of Quantum-Dot Cellular Automata**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. 3–20p.

LENT, C. S.; TOUGAW, P. D. A device architecture for computing with quantum dots. **Proceedings of the IEEE**, [S.l.], v.85, n.4, p.541–557, 1997.

LING, A.; SINGH, D.; BROWN, S. FPGA technology mapping: a study of optimality. In: DESIGN AUTOMATION CONFERENCE, 2005., 42., 2005. **Proceedings...** [S.l.: s.n.], 2005. p.427–432.

LIU, W.; O'NEILL, M.; SWARTZLANDER, E. **Design of Semiconductor QCA Systems**. [S.l.]: Artech House, 2013.

LOGICS. **Catalog of 53 handmade optimum switch networks**. Available: http://www.inf.ufrgs.br/logics/docman/53_NSP_Catalog.pdf. Acessed in: Nov. 2019.

LYNCE, I.; MARQUES-SILVA, J. Efficient Haplotype Inference with Boolean Satisfiability. In: AAAI, 2006. **Anais...** [S.l.: s.n.], 2006.

MARQUES-SILVA, J. a. P.; SAKALLAH, K. A. Boolean Satisfiability in Electronic Design Automation. In: ANNUAL DESIGN AUTOMATION CONFERENCE, 37., 2000, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2000. p.675–680. (DAC '00).

MARTINS, M. G. A. et al. Boolean factoring with multi-objective goals. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, 2010., 2010. **Anais...** [S.l.: s.n.], 2010. p.229–234.

MCMURCHIE, L.; EBELING, C. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In: THIRD INTERNATIONAL ACM SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS, 1995. **Anais...** [S.l.: s.n.], 1995. p.111–117.

MISHCHENKO, A.; CHATTERJEE, S.; BRAYTON, R.; EEN, N. Improvements to Combinational Equivalence Checking. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2006., 2006, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2006. p.836–843. (ICCAD '06).

NEIGENFIND, J. et al. Haplotype inference from unphased SNP data in heterozygous polyploids based on SAT. **BMC Genomics**, [S.l.], v.9, p.356 – 356, 2008.

NIEMIER, M. T. et al. Shape Engineering for Controlled Switching With Nanomagnet Logic. **IEEE Transactions on Nanotechnology**, [S.l.], v.11, n.2, p.220–230, 2012.

PETKOVSKA, A. Exploiting Satisfiability Solvers for Efficient Logic Synthesis. , Lausanne, p.153, 2017.

POSSANI, V. N. et al. Graph-Based Transistor Network Generation Method for Supergate Design. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [S.l.], v.24, n.2, p.692–705, 2016.

REIS, R. Design automation of transistor networks, a new challenge. In: IEEE INTERNATIONAL SYMPOSIUM OF CIRCUITS AND SYSTEMS (ISCAS), 2011., 2011. **Anais...** [S.l.: s.n.], 2011. p.2485–2488.

RINTANEN, J.; HELJANKO, K.; NIEMELä, I. Planning as satisfiability: parallel plans and algorithms for plan search. **Artificial Intelligence**, [S.l.], v.170, p.1031–1080, 09 2006.

ROSA, L. S. da; SCHNEIDER, F. R.; RIBAS, R. P.; REIS, A. I. Switch level optimization of digital CMOS gate networks. In: INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN, 2009., 2009. **Anais...** [S.l.: s.n.], 2009. p.324–329.

SILVA, L.; SILVA, J.; SILVEIRA, L.; SKALLAH, K. Timing analysis using propositional satisfiability. In: IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS. SURFING THE WAVES OF SCIENCE AND TECHNOLOGY (CAT. NO.98EX196), 1998., 1998. **Anais...** [S.l.: s.n.], 1998. v.3, p.95–98 vol.3.

SMANIOTTO, G. H. et al. Toward better layout design in ASTRAN CAD tool by using an efficient transistor folding. In: IEEE 59TH INTERNATIONAL MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS (MWSCAS), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.1–4.

SMITH, A.; VENERIS, A.; ALI, M.; VIGLAS, A. Fault diagnosis and logic debugging using Boolean satisfiability. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.24, n.10, p.1606–1621, 2005.

SOARES, T. R. B. S. et al. NMLSim: a Nanomagnetic Logic (NML) circuit designer and simulation tool. **Journal of Computational Electronics**, [S.l.], v.17, n.3, p.1370–1381, Sep 2018.

SOEKEN, M. et al. Heuristic NPN Classification for Large Functions Using AIGs and LEXSAT. In: SAT, 2016. **Anais...** [S.l.: s.n.], 2016.

SOOS, M.; NOHL, K.; CASTELLUCCIA, C. Extending SAT Solvers to Cryptographic Problems. In: THEORY AND APPLICATIONS OF SATISFIABILITY TESTING - SAT 2009, 12TH INTERNATIONAL CONFERENCE, SAT 2009, SWANSEA, UK, JUNE 30 - JULY 3, 2009. PROCEEDINGS, 2009. **Anais...** Springer, 2009. p.244–257. (Lecture Notes in Computer Science, v.5584).

SOROKIN, A.; RYZHENKO, N. SAT-Based Placement Adjustment of FinFETs inside Unroutable Standard Cells Targeting Feasible DRC-Clean Routing. In: GREAT LAKES SYMPOSIUM ON VLSI, 2019., 2019, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2019. p.159–164. (GLSVLSI '19).

TAYLOR, B.; PILEGGI, L. Exact Combinatorial Optimization Methods for Physical Design of Regular Logic Bricks. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 2007., 2007. **Anais...** [S.l.: s.n.], 2007. p.344–349.

TIMLER, J.; LENT, C. Power gain and dissipation in quantum-dot cellular automata. **Journal of Applied Physics**, [S.l.], v.91, p.823–831, 01 2002.

TORRES, F. S. et al. Exploration of the Synchronization Constraint in Quantum-dot Cellular Automata. In: EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN (DSD), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.642–648.

TRINDADE, A. et al. A Placement and routing algorithm for Quantum-dot Cellular Automata. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.1–6.

SIEKMANN, J. H.; WRIGHTSON, G. (Ed.). **On the Complexity of Derivation in Propositional Calculus**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983. 466–483p.

UEHARA; VANCLEEMPUT. Optimal Layout of CMOS Functional Arrays. **IEEE Transactions on Computers**, [S.l.], v.C-30, n.5, p.305–312, 1981.

VANKAMAMIDI, V.; OTTAVI, M.; LOMBARDI, F. Two-Dimensional Schemes for Clocking/Timing of QCA Circuits. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.27, n.1, p.34–44, 2008.

VARGA, E.; CSABA, G.; BERNSTEIN, G. H.; POROD, W. Implementation of a nanomagnetic full adder circuit. In: IEEE INTERNATIONAL CONFERENCE ON NANOTECHNOLOGY, 2011., 2011. **Anais...** [S.l.: s.n.], 2011. p.1244–1247.

VARGA, E. et al. Experimental Demonstration of Fanout for Nanomagnetic Logic. **IEEE Transactions on Nanotechnology**, [S.l.], v.9, n.6, p.668–670, 2010.

WALTER, M. et al. An exact method for design exploration of quantum-dot cellular automata. In: DESIGN, AUTOMATION TEST IN EUROPE CONFERENCE EXHIBITION (DATE), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.503–508.

WALTER, M. et al. fiction: An Open Source Framework for the Design of Field-coupled Nanocomputing Circuits. **CoRR**, [S.l.], v.abs/1905.02477, 2019.

WALTER, M. et al. Scalable Design for Field-Coupled Nanocomputing Circuits. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 24., 2019, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2019. p.197–202. (ASPDAC '19).

WALUS, K.; DYSART, T.; JULLIEN, G.; BUDIMAN, R. QCADesigner: a rapid design and Simulation tool for quantum-dot cellular automata. **IEEE Transactions on Nanotechnology**, [S.l.], v.3, n.1, p.26–31, 2004.

WANG, L.-T.; CHANG, Y.-W.; CHENG, K.-T. T. **Electronic Design Automation**: Synthesis, Verification, and Test. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009.

WOOD, R.; RUTENBAR, R. FPGA routing and routability estimation via Boolean satisfiability. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [S.l.], v.6, n.2, p.222–231, 1998.

WYNN, E. **A comparison of encodings for cardinality constraints in a SAT solver**.

ZIESEMER, A. M.; REIS, R. Physical design automation of transistor networks. **Microelectronic Engineering**, [S.l.], v.148, p.122–128, 2015. Micro/Nano Emerging Technologies 2015.

ZIESEMER JR, A. **Síntese Automática do Leiaute de Redes de Transistores**. 2014. 125p. Tese (Doutorado em Ciência da Computação) — .

Appendices

APPENDIX A – Example: SCCG design

A.1 Input netlist and design conditions

Let us consider the netlist illustrated in Figure 49 as the input of the proposed method. Along with that, consider that the number of columns available for placement is $C = 6$ in the current iteration of the proposed Algorithm 3, i.e., it is the second try after an unsatisfiable first iteration.

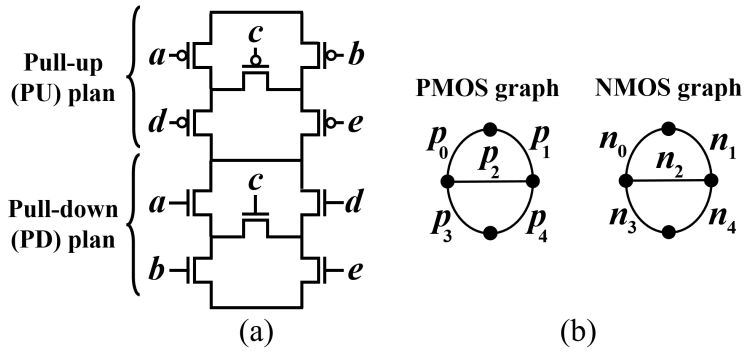


Figure 49 – Input netlist (a) with its equivalent graphs (b).

In this scenario, we have the placement constraints defined as follows.

A.2 Placement constraints

- *Transistor allocation constraint:* each transistor must be allocated at exactly one column. Thus, we have

$$
\begin{aligned}
p_0(1) &\implies \neg p_0(2) \wedge \neg p_0(3) \wedge ... \wedge \neg p_0(6), \\
p_0(2) &\implies \neg p_0(1) \wedge \neg p_0(3) \wedge ... \wedge \neg p_0(6), \\
&... \\
p_0(6) &\implies \neg p_0(1) \wedge \neg p_0(2) \wedge ... \wedge \neg p_0(5)
\end{aligned}
$$

  for the transistor p_0. The same is applied for every p_i and n_i.

- *Transistor overlapping constraint:* each position of the layout must contain at most

one transistor. Thus, we have

$$
\begin{gathered}
atmost(1, \{p_0(1), p_1(1), ...p_5(1)\}), \\
atmost(1, \{p_0(2), p_1(2), ...p_5(2)\}), \\
... \\
atmost(1, \{p_0(6), p_1(6), ...p_5(6)\}),
\end{gathered}
$$

such that the same is applied for the n set of variables.

- *Diffusion sharing constraint:* two transistors positioned laterally next to each other must share a node in the netlist. Thus, considering that in Figure 49 the drain of p_0 is shared with the drain of p_1 and the source of p_0 is shared with the sources of p_2 and p_3, we have

$$
\begin{aligned}
p_0(1,d) &\implies \neg p_1(2,s), \\
p_0(1,s) &\implies \neg p_1(2,d), \\
p_0(1,s) &\implies \neg p_1(2,s), \\
p_0(1,d) &\implies \neg p_2(2,d), \\
p_0(1,d) &\implies \neg p_2(2,s), \\
p_0(1,s) &\implies \neg p_2(2,d), \\
p_0(1,d) &\implies \neg p_3(2,d), \\
p_0(1,d) &\implies \neg p_3(2,s), \\
p_0(1,s) &\implies \neg p_3(2,d), \\
p_0(1,d) &\implies \neg p_4(2,d), \\
p_0(1,d) &\implies \neg p_4(2,s), \\
p_0(1,s) &\implies \neg p_4(2,d), \\
p_0(1,s) &\implies \neg p_4(2,s), \\
&... \\
p_0(5,d) &\implies \neg p_4(6,d), \\
p_0(5,d) &\implies \neg p_4(6,s), \\
p_0(5,s) &\implies \neg p_4(6,d), \\
p_0(5,s) &\implies \neg p_4(6,s)
\end{aligned}
$$

for p_0. The same is applied for every p_i and n_i.

- *Gate alignment constraint*: two transistors positioned vertically next to each other must share the gate signal in the netlist. Thus, considering that p_0 has the same

gate signal of only n_0, we have

$$
\begin{aligned}
p_0(1) &\implies \neg n_1(1),\\
p_0(1) &\implies \neg n_2(1),\\
p_0(1) &\implies \neg n_3(1),\\
p_0(1) &\implies \neg n_4(1),\\
p_0(2) &\implies \neg n_1(2),\\
&\dots\\
p_0(6) &\implies \neg n_4(6),
\end{aligned}
$$

for p_0. The same is applied for every p_i and n_i.

A.3 Satisfiable solution

Following the Algorithm 3, the conjunction of all formulas is computed through an SAT solver. In this case, this instance is satisfiable, i.e., there is a valid solution for the transistor placement using six layout columns ($C = 6$). Figure 50 shows the pseudo-layout of this solution.

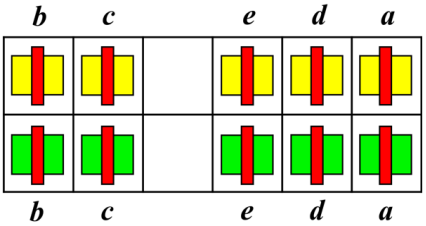


Figure 50 – Transistor placement solution for the circuit presented in Figure 49.

APPENDIX B – Example: QCA design

B.1 Input netlist and design conditions

Let us consider the 2:1 MUX circuit illustrated in Figure 51 (a) as the input of the proposed method. Along with that, consider the design space presented in Figure 51 (b) as the current state following the Algorithm 4. Furthermore, let us consider three cycles available for design.

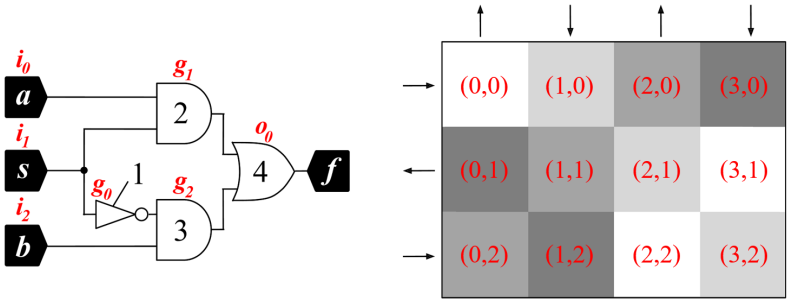


Figure 51 – 2:1 MUX (a) and design space available in the current algorithm iteration (b).

In this scenario, let us define the placement, routing, and synchronization constraints as follows.

B.2 Placement constraints

- *Single driver constraint*: each driver should be placed at exactly one grid location. Thus, we have

$$
\begin{aligned}
&exactly(1, \{i_{0(d)}(0,0), i_{0(d)}(0,1), ..., i_{0(d)}(3,2)\}),\\
&exactly(1, \{i_{1(d)}(0,0), i_{1(d)}(0,1), ..., i_{1(d)}(3,2)\}),\\
&exactly(1, \{i_{2(d)}(0,0), i_{2(d)}(0,1), ..., i_{2(d)}(3,2)\}),\\
&exactly(1, \{g_{0(d)}(0,0), g_{0(d)}(0,1), ..., g_{0(d)}(3,2)\}),\\
&exactly(1, \{g_{1(d)}(0,0), g_{1(d)}(0,1), ..., g_{1(d)}(3,2)\}),\\
&exactly(1, \{g_{2(d)}(0,0), g_{2(d)}(0,1), ..., g_{2(d)}(3,2)\}),\\
&exactly(1, \{o_{0(d)}(0,0), o_{0(d)}(0,1), ..., o_{0(d)}(3,2)\}).
\end{aligned}
$$

- *Overlapping constraint*: each grid location contains at most one gate or (exclusively) at most two wire cells containing at most two input drivers. Thus, we have

$$
\begin{aligned}
& i_{0(d)}(0,0) \implies \\
& atmost(2, \{i_{0(d)}(0,0), i_{1(d)}(0,0)\}) \wedge \\
& exactly(0, \{g_{0(d)}(0,0), g_{1(d)}(0,0), ..., o_{0(d)}(0,0)\}) \wedge \\
& atmost(2, \{i_{0(w)}(0,0), i_{1(w)}(0,0), g_{0(w)}(0,0), g_{1(w)}(0,0), ..., o_{0(w)}(0,0)\}), \\
& i_{1(d)}(0,0) \implies \\
& atmost(2, \{i_{0(d)}(0,0), i_{1(d)}(0,0)\}) \wedge \\
& exactly(0, \{g_{0(d)}(0,0), g_{1(d)}(0,0), ..., o_{0(d)}(0,0)\}) \wedge \\
& atmost(2, \{i_{0(w)}(0,0), i_{1(w)}(0,0), g_{0(w)}(0,0), g_{1(w)}(0,0), ..., o_{0(w)}(0,0)\}),
\end{aligned}
$$

  for the inputs,

$$
\begin{aligned}
& g_{0(d)}(0,0) \implies \\
& exactly(1, \{g_{0(d)}(0,0), g_{1(d)}(0,0), ..., o_{0(d)}(0,0)\}) \wedge \\
& exactly(0, \{i_{0(w)}(0,0), i_{1(w)}(0,0), g_{0(w)}(0,0), g_{1(w)}(0,0), ..., o_{0(w)}(0,0)\}), \\
& ... \\
& g_{3(d)}(0,0) \implies \\
& exactly(1, \{g_{0(d)}(0,0), g_{1(d)}(0,0), ..., o_{0(d)}(0,0)\}) \wedge \\
& exactly(0, \{i_{0(w)}(0,0), i_{1(w)}(0,0), g_{0(w)}(0,0), g_{1(w)}(0,0), ..., o_{0(w)}(0,0)\}), \\
& o_{0(d)}(0,0) \implies \\
& exactly(1, \{g_{0(d)}(0,0), g_{1(d)}(0,0), ..., o_{0(d)}(0,0)\}) \wedge \\
& exactly(0, \{i_{0(w)}(0,0), i_{1(w)}(0,0), g_{0(w)}(0,0), g_{1(w)}(0,0), ..., o_{0(w)}(0,0)\}),
\end{aligned}
$$

  for the gates and output, and,

$$
\begin{aligned}
& g_{0(w)}(0,0) \implies \\
& exactly(0, \{i_{0(d)}(0,0), i_{1(d)}(0,0), g_{0(d)}(0,0), g_{1(d)}(0,0), ..., o_{0(d)}(0,0)\}) \wedge \\
& atmost(2, \{i_{0(w)}(0,0), i_{1(w)}(0,0), g_{0(w)}(0,0), g_{1(w)}(0,0), ..., o_{0(w)}(0,0)\}),
\end{aligned}
$$

  for the wires. The same is valid for every coordinate of the USE grid.

B.3 Synchronization constraints

- *Driver synchronization constraint:* for each gate or output driver, its inputs should be submitted to the same clock cycle if the cell is placed on locations upon phases 1, 2, or 3, and to the previous clock cycle if the cell is placed upon phase 0. Thus,

expanding the equations for $c = 1$ (second cycle), we have

$$
\begin{aligned}
g^1_{1(d)}(0,0) &\implies i^0_0(0,1) \wedge i^0_1(0,1),\\
g^1_{1(d)}(1,0) &\implies i^1_0(0,0) \wedge i^1_1(0,0),\\
g^1_{1(d)}(2,0) &\implies (i^1_0(1,0) \vee i^1_0(2,1)) \wedge (i^1_1(1,0) \vee i^1_1(2,1)),\\
g^1_{1(d)}(3,0) &\implies i^1_0(2,0) \wedge i^1_1(2,0),\\
g^1_{1(d)}(0,1) &\implies (i^1_0(1,1) \vee i^1_0(0,2)) \wedge (i^1_1(1,1) \vee i^1_1(0,2)),\\
g^1_{1(d)}(1,1) &\implies (i^1_0(1,0) \vee i^1_0(0,1)) \wedge (i^1_1(1,0) \vee i^1_1(0,1)),\\
g^1_{1(d)}(2,1) &\implies (i^1_0(3,1) \vee i^1_0(2,2)) \wedge (i^1_1(3,1) \vee i^1_1(2,2)),\\
g^1_{1(d)}(3,1) &\implies i^0_0(3,0) \wedge i^0_1(3,0),\\
g^1_{1(d)}(1,2) &\implies (i^1_0(1,1) \vee i^1_0(0,2)) \wedge (i^1_1(1,1) \vee i^1_1(0,2)),\\
g^1_{1(d)}(2,2) &\implies i^1_0(1,2) \wedge i^1_1(1,2),\\
g^1_{1(d)}(3,2) &\implies (i^1_0(3,1) \vee i^1_0(2,2)) \wedge (i^1_1(3,1) \vee i^1_1(2,2)),
\end{aligned}
$$

such that the same is valid for all gates and outputs, as well as for all cycles.

- *Wire synchronization constraint:* for each wire cell, its input should be submitted to the same clock if the cell is submitted to the phase 1, 2, or 3, and to the previous clock cycle if the cell is submitted to the phase 0. Thus, expanding the equations for $c = 1$ (second cycle), we have

$$
\begin{aligned}
g^1_{1(w)}(0,0) &\implies g^0_1(0,1),\\
g^1_{1(w)}(1,0) &\implies g^1_1(0,0),\\
g^1_{1(w)}(2,0) &\implies g^1_1(1,0) \vee g^1_1(2,1),\\
g^1_{1(w)}(3,0) &\implies g^1_1(2,0),\\
g^1_{1(w)}(0,1) &\implies g^1_1(1,1) \vee g^1_1(0,2),\\
g^1_{1(w)}(1,1) &\implies g^1_1(1,0) \vee g^1_1(0,1),\\
g^1_{1(w)}(2,1) &\implies g^1_1(3,1) \vee g^1_1(2,2),\\
g^1_{1(w)}(3,1) &\implies g^0_1(3,0),\\
g^1_{1(w)}(1,2) &\implies g^1_1(1,1) \vee g^1_1(0,2),\\
g^1_{1(w)}(2,2) &\implies g^1_1(1,2),\\
g^1_{1(w)}(3,2) &\implies g^1_1(3,1) \vee g^1_1(2,2),
\end{aligned}
$$

  such that the same is valid for all inputs, gates, and outputs, as well as for all cycles.

- *Single-cycle constraint:* each cell should operate under exactly one cycle. Thus,

we have

$$
\begin{aligned}
g_0(0,0) &\implies exactly(1, \{g_0^0(0,0), g_0^1(0,0), g_0^2(0,0)\}),\\
g_0(1,0) &\implies exactly(1, \{g_0^0(1,0), g_0^1(1,0), g_0^2(1,0)\}),\\
&\ldots\\
g_0(3,2) &\implies exactly(1, \{g_0^0(3,2), g_0^1(3,2), g_0^2(3,2)\}),
\end{aligned}
$$

such that the same is valid for all inputs, gates, and outputs.

- *Input/output phase synchronization constraint:* each input should be submitted to the same phase. The same is valid for each output. Thus,

$$
\begin{aligned}
i_{0(d)}(0,0) &\implies (i_{1(d)}(0,0) \vee i_{1(d)}(3,1) \vee i_{1(d)}(2,2)) \wedge (i_{1(d)}(0,0) \vee i_{2(d)}(3,1) \vee i_{2(d)}(2,2)),\\
i_{0(d)}(1,0) &\implies (i_{1(d)}(1,0) \vee i_{1(d)}(2,1) \vee i_{1(d)}(3,2)) \wedge (i_{1(d)}(1,0) \vee i_{2(d)}(2,1) \vee i_{2(d)}(3,2)),\\
i_{0(d)}(2,0) &\implies (i_{1(d)}(2,0) \vee i_{1(d)}(1,1) \vee i_{1(d)}(0,2)) \wedge (i_{1(d)}(2,0) \vee i_{2(d)}(1,1) \vee i_{2(d)}(0,2)),\\
i_{0(d)}(3,0) &\implies (i_{1(d)}(3,0) \vee i_{1(d)}(0,1) \vee i_{1(d)}(1,2)) \wedge (i_{1(d)}(3,0) \vee i_{2(d)}(0,1) \vee i_{2(d)}(1,2)),\\
i_{0(d)}(0,1) &\implies (i_{1(d)}(3,0) \vee i_{1(d)}(0,1) \vee i_{1(d)}(1,2)) \wedge (i_{1(d)}(3,0) \vee i_{2(d)}(0,1) \vee i_{2(d)}(1,2)),\\
i_{0(d)}(1,1) &\implies (i_{1(d)}(2,0) \vee i_{1(d)}(1,1) \vee i_{1(d)}(0,2)) \wedge (i_{1(d)}(2,0) \vee i_{2(d)}(1,1) \vee i_{2(d)}(0,2)),\\
i_{0(d)}(2,1) &\implies (i_{1(d)}(1,0) \vee i_{1(d)}(2,1) \vee i_{1(d)}(3,2)) \wedge (i_{1(d)}(1,0) \vee i_{2(d)}(2,1) \vee i_{2(d)}(3,2)),\\
i_{0(d)}(3,1) &\implies (i_{1(d)}(0,0) \vee i_{1(d)}(3,1) \vee i_{1(d)}(2,2)) \wedge (i_{1(d)}(0,0) \vee i_{2(d)}(3,1) \vee i_{2(d)}(2,2)),\\
i_{0(d)}(0,2) &\implies (i_{1(d)}(2,0) \vee i_{1(d)}(1,1) \vee i_{1(d)}(0,2)) \wedge (i_{1(d)}(2,0) \vee i_{2(d)}(1,1) \vee i_{2(d)}(0,2)),\\
i_{0(d)}(1,2) &\implies (i_{1(d)}(3,0) \vee i_{1(d)}(0,1) \vee i_{1(d)}(1,2)) \wedge (i_{1(d)}(3,0) \vee i_{2(d)}(0,1) \vee i_{2(d)}(1,2)),\\
i_{0(d)}(2,2) &\implies (i_{1(d)}(0,0) \vee i_{1(d)}(3,1) \vee i_{1(d)}(2,2)) \wedge (i_{1(d)}(0,0) \vee i_{2(d)}(3,1) \vee i_{2(d)}(2,2)),\\
i_{0(d)}(3,2) &\implies (i_{1(d)}(1,0) \vee i_{1(d)}(2,1) \vee i_{1(d)}(3,2)) \wedge (i_{1(d)}(1,0) \vee i_{2(d)}(2,1) \vee i_{2(d)}(3,2)),
\end{aligned}
$$

such that the same is valid for all other inputs and for the output.

- *Input/output cycle synchronization constraint:* each input should start operating

on the same clock cycle. The same is valid for each output. Thus, we have

$$
\begin{aligned}
i^0_{0(d)}(0,0) \implies & (i^0_{1(d)}(0,0) \vee i^0_{1(d)}(1,0) \vee ... \vee i^0_{1(d)}(3,2)) \wedge \\
& (i^0_{2(d)}(0,0) \vee i^0_{2(d)}(1,0) \vee ... \vee i^0_{2(d)}(3,2)), \\
i^0_{0(d)}(1,0) \implies & (i^0_{1(d)}(0,0) \vee i^0_{1(d)}(1,0) \vee ... \vee i^0_{1(d)}(3,2)) \wedge \\
& (i^0_{2(d)}(0,0) \vee i^0_{2(d)}(1,0) \vee ... \vee i^0_{2(d)}(3,2)), \\
& ... \\
i^0_{0(d)}(3,2) \implies & (i^0_{1(d)}(0,0) \vee i^0_{1(d)}(1,0) \vee ... \vee i^0_{1(d)}(3,2)) \wedge \\
& (i^0_{2(d)}(0,0) \vee i^0_{2(d)}(1,0) \vee ... \vee i^0_{2(d)}(3,2)), \\
i^1_{0(d)}(0,0) \implies & (i^1_{1(d)}(0,0) \vee i^1_{1(d)}(1,0) \vee ... \vee i^1_{1(d)}(3,2)) \wedge \\
& (i^1_{2(d)}(0,0) \vee i^1_{2(d)}(1,0) \vee ... \vee i^1_{2(d)}(3,2)), \\
& ... \\
i^2_{0(d)}(3,2) \implies & (i^2_{1(d)}(0,0) \vee i^2_{1(d)}(1,0) \vee ... \vee i^2_{1(d)}(3,2)) \wedge \\
& (i^2_{2(d)}(0,0) \vee i^2_{2(d)}(1,0) \vee ... \vee i^2_{2(d)}(3,2)).
\end{aligned}
$$

B.4 Routing constraints

- *No return constraint:* each signal should not enter and exit the cell on the same side. Thus,

$$
\begin{aligned}
& \neg(ws_{(0)}(0,0) \wedge wn_{(0)}(0,0)) \wedge \neg(nw_{(0)}(0,0) \wedge ne_{(0)}(0,0)) \wedge \\
& \neg(ws_{(1)}(0,0) \wedge wn_{(1)}(0,0)) \wedge \neg(nw_{(1)}(0,0) \wedge ne_{(1)}(0,0)) \wedge \\
& \neg(ws_{(0)}(1,0) \wedge wn_{(0)}(1,0)) \wedge \neg(nw_{(0)}(1,0) \wedge ne_{(0)}(1,0)) \wedge \\
& \neg(ws_{(1)}(1,0) \wedge wn_{(1)}(1,0)) \wedge \neg(nw_{(1)}(1,0) \wedge ne_{(1)}(1,0)) \wedge \\
& ... \\
& \neg(ws_{(0)}(3,2) \wedge wn_{(0)}(3,2)) \wedge \neg(nw_{(0)}(3,2) \wedge ne_{(0)}(3,2)) \wedge \\
& \neg(ws_{(1)}(3,2) \wedge wn_{(1)}(3,2)) \wedge \neg(nw_{(1)}(3,2) \wedge ne_{(1)}(3,2)).
\end{aligned}
$$

- *Border constraint:* ports located on the border of the routing area should not be connected. Thus, we have

$$
\begin{aligned}
& \neg nw(0,0) \wedge \neg ne(0,0) \wedge \neg ws(0,0) \wedge \neg wn(0,0) \wedge \\
& \neg nw(1,0) \wedge \neg ne(1,0) \wedge \neg ws(0,1) \wedge \neg wn(0,1) \wedge \\
& \neg nw(2,0) \wedge \neg ne(2,0) \wedge \neg ws(0,2) \wedge \neg wn(0,2) \wedge \\
& \neg nw(3,0) \wedge \neg ne(3,0).
\end{aligned}
$$

- *Invalid corners constraint:* the routing paths presented in Figure 26 are not allowed. Thus, we have

$$\begin{aligned}
&\neg((nw_{(0)}(0,1) \vee ne_{(0)}(0,1)) \wedge (ws_{(0)}(0,0) \vee wn_{(0)}(0,0))),\\
&\neg((nw_{(0)}(2,1) \vee ne_{(0)}(2,1)) \wedge (ws_{(0)}(2,0) \vee wn_{(0)}(2,0))),\\
&\neg((nw_{(0)}(0,2) \vee ne_{(0)}(0,2)) \wedge (ws_{(0)}(1,1) \vee wn_{(0)}(1,1))),\\
&\neg((nw_{(0)}(1,1) \vee ne_{(0)}(1,1)) \wedge (ws_{(0)}(2,1) \vee wn_{(0)}(2,1))),\\
&\neg((nw_{(0)}(2,2) \vee ne_{(0)}(2,2)) \wedge (ws_{(0)}(3,1) \vee wn_{(0)}(3,1))),\\
&\neg((nw_{(0)}(1,2) \vee ne_{(0)}(1,2)) \wedge (ws_{(0)}(1,2) \vee wn_{(0)}(1,2))),\\
&\neg((nw_{(0)}(3,2) \vee ne_{(0)}(3,2)) \wedge (ws_{(0)}(3,2) \vee wn_{(0)}(3,2))),\\
&\neg((nw_{(1)}(0,1) \vee ne_{(1)}(0,1)) \wedge (ws_{(1)}(0,0) \vee wn_{(1)}(0,0))),\\
&\neg((nw_{(1)}(2,1) \vee ne_{(1)}(2,1)) \wedge (ws_{(1)}(2,0) \vee wn_{(1)}(2,0))),\\
&\neg((nw_{(1)}(0,2) \vee ne_{(1)}(0,2)) \wedge (ws_{(1)}(1,1) \vee wn_{(1)}(1,1))),\\
&\neg((nw_{(1)}(1,1) \vee ne_{(1)}(1,1)) \wedge (ws_{(1)}(2,1) \vee wn_{(1)}(2,1))),\\
&\neg((nw_{(1)}(2,2) \vee ne_{(1)}(2,2)) \wedge (ws_{(1)}(3,1) \vee wn_{(1)}(3,1))),\\
&\neg((nw_{(1)}(1,2) \vee ne_{(1)}(1,2)) \wedge (ws_{(1)}(1,2) \vee wn_{(1)}(1,2))),\\
&\neg((nw_{(1)}(3,2) \vee ne_{(1)}(3,2)) \wedge (ws_{(1)}(3,2) \vee wn_{(1)}(3,2))).
\end{aligned}$$

- *Majority access constraint:* in a cell containing a majority gate, two neighbor ports should not be connected if they are located in the corners of this USE cell. Thus, we have

$$\begin{aligned}
g_{1(d)}(0,0) \implies &\neg(wn(0,0) \wedge nw(0,0)) \wedge \neg(ne(0,0) \wedge wn(1,0)) \wedge\\
&\neg(ws(0,0) \wedge nw(0,1)) \wedge \neg(ws(1,0) \wedge ne(0,1)),\\
g_{1(d)}(1,0) \implies &\neg(wn(1,0) \wedge nw(1,0)) \wedge \neg(ne(1,0) \wedge wn(2,0)) \wedge\\
&\neg(ws(1,0) \wedge nw(1,1)) \wedge \neg(ws(2,0) \wedge ne(1,1)),\\
&\ldots\\
g_{2(d)}(3,2) \implies &\neg(wn(3,2) \wedge nw(3,2)),
\end{aligned}$$

  such that the same is applied for the output node.

- *Gate driver adjacency constraint:* each gate should be followed by the cells connected to its output or by its wires and should be preceded by the cells connected

to its inputs. Thus, we have

$$
\begin{aligned}
g_{0(d)}(0,0) &\implies exactly(1,\{i_1(0,1)\}),\\
g_{0(d)}(1,0) &\implies exactly(1,\{i_1(0,0)\}),\\
g_{0(d)}(2,0) &\implies exactly(1,\{i_1(1,0),i_1(2,1)\}),\\
g_{0(d)}(3,0) &\implies exactly(1,\{i_1(2,0)\}),\\
g_{0(d)}(0,1) &\implies exactly(1,\{i_1(0,2),i_1(1,1)\}),\\
g_{0(d)}(1,1) &\implies exactly(1,\{i_1(1,0),i_1(2,1)\}),\\
g_{0(d)}(2,1) &\implies exactly(1,\{i_1(2,2),i_1(3,1)\}),\\
g_{0(d)}(3,1) &\implies exactly(1,\{i_1(3,0)\}),\\
g_{0(d)}(1,2) &\implies exactly(1,\{i_1(1,1),i_1(0,2)\}),\\
g_{0(d)}(2,2) &\implies exactly(1,\{i_1(1,2)\}),\\
g_{0(d)}(3,2) &\implies exactly(1,\{i_1(3,1),i_1(2,2)\}),
\end{aligned}
$$

such that the same is applied for the other gates and output node, and

$$
\begin{aligned}
g_{0(d)}(0,0) &\implies exactly(1,\{g_2(1,0)\}),\\
g_{0(d)}(1,0) &\implies exactly(1,\{g_2(2,0),g_2(1,1)\}),\\
g_{0(d)}(2,0) &\implies exactly(1,\{g_2(3,0)\}),\\
g_{0(d)}(3,0) &\implies exactly(1,\{g_2(3,1)\}),\\
g_{0(d)}(0,1) &\implies exactly(1,\{g_2(0,0)\}),\\
g_{0(d)}(1,1) &\implies exactly(1,\{g_2(0,1),g_2(1,2)\}),\\
g_{0(d)}(2,1) &\implies exactly(1,\{g_2(2,0),g_2(1,1)\}),\\
g_{0(d)}(3,1) &\implies exactly(1,\{g_2(3,2),g_2(2,1)\}),\\
g_{0(d)}(0,2) &\implies exactly(1,\{g_2(1,2),g_2(0,1)\}),\\
g_{0(d)}(1,2) &\implies exactly(1,\{g_2(2,2)\}),\\
g_{0(d)}(2,2) &\implies exactly(1,\{g_2(2,1),g_2(3,2)\}),
\end{aligned}
$$

such that the same is also applied for the other gates and output node.

- *Input driver adjacency constraint:* each input driver connected to one gate should be followed by one of its own wires or by the gate. Furthermore, if the input is connected to more than one gate, then it should be followed by one of its own

wires or by at least one of the gates connected to its output. Thus, we have

$$\begin{aligned}
i_{0(d)}(0,0) &\implies exactly(1,\{i_{0(w)}(1,0), g_{1(d)}(1,0)\}),\\
i_{0(d)}(1,0) &\implies exactly(1,\{i_{0(w)}(2,0), i_{0(w)}(1,1), g_{1(d)}(2,0), g_{1(d)}(1,1)\}),\\
&\ldots\\
i_{0(d)}(2,2) &\implies exactly(1,\{i_{0(w)}(3,2), i_{0(w)}(2,1), g_{1(d)}(3,2), g_{1(d)}(2,1)\}),\\
i_{2(d)}(0,0) &\implies exactly(1,\{i_{2(w)}(1,0), g_{1(d)}(1,0)\}),\\
i_{2(d)}(1,0) &\implies exactly(1,\{i_{2(w)}(2,0), i_{2(w)}(1,1), g_{2(d)}(2,0), g_{2(d)}(1,1)\}),\\
&\ldots\\
i_{2(d)}(2,2) &\implies exactly(1,\{i_{2(w)}(3,2), i_{2(w)}(2,1), g_{2(d)}(3,2), g_{2(d)}(2,1)\})
\end{aligned}$$

for i_0 and i_2 ($degree(i_0) = degree(i_2) = 1$), and

$$\begin{aligned}
i_{1(d)}(0,0) &\implies i_{1(w)}(1,0) \vee g_{0(d)}(1,0) \vee g_{1(d)}(1,0),\\
i_{1(d)}(1,0) &\implies i_{1(w)}(1,1) \vee i_{1(w)}(2,0) \vee g_{0(d)}(1,1)\\
&\quad \vee g_{0(d)}(2,0) \vee g_{1(d)}(1,1) \vee g_{1(d)}(2,0),\\
&\ldots\\
i_{1(d)}(2,2) &\implies i_{1(w)}(3,2) \vee g_{0(d)}(3,2) \vee g_{1(d)}(3,2)
\end{aligned}$$

for i_1 ($degree(i_1) \geq 2$).

- *Wire adjacency constraint:* each wire should be followed by the cells connected to its output or by other wire of the same signal and should be preceded by a wire or by a driver of the same signal. Thus, we have

$$\begin{aligned}
i_{0(w)}(0,0) &\implies exactly(1,\{i_{0(w)}(1,0), g_{1(d)}(1,0)\}),\\
i_{0(w)}(1,0) &\implies exactly(1,\{i_{0(w)}(1,1), i_{0(w)}(2,0), g_{1(d)}(1,1), g_{1(d)}(2,0)\}),\\
&\ldots\\
i_{2(w)}(2,2) &\implies exactly(1,\{i_{0(w)}(3,2), i_{0(w)}(2,1), g_{1(d)}(3,2), g_{1(d)}(2,1)\})
\end{aligned}$$

and

$$\begin{aligned}
i_{0(w)}(0,0) &\implies atleast(1,\{i_{0(w)}(0,1), i_{0(d)}(0,1)\}),\\
i_{0(w)}(1,0) &\implies atleast(1,\{i_{0(w)}(0,0), i_{0(d)}(0,0)\}),\\
&\ldots\\
i_{2(w)}(3,2) &\implies atleast(1,\{i_{2(w)}(3,1), i_{2(w)}(2,2), i_{2(d)}(3,1), i_{2(d)}(2,2)\}).
\end{aligned}$$

B.5 Satisfiable solution

Following the Algorithm 4, the conjunction of all formulas is computed through an SAT solver. In this case, the instance shown in this chapter is satisfiable, i.e., there is a valid solution for the QCA design which fulfills all the design restrictions. Figure 52 presents the resulting circuit.

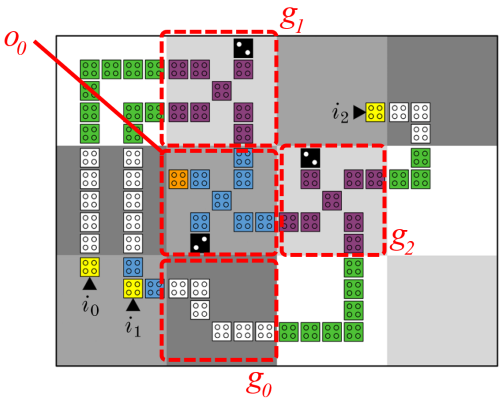
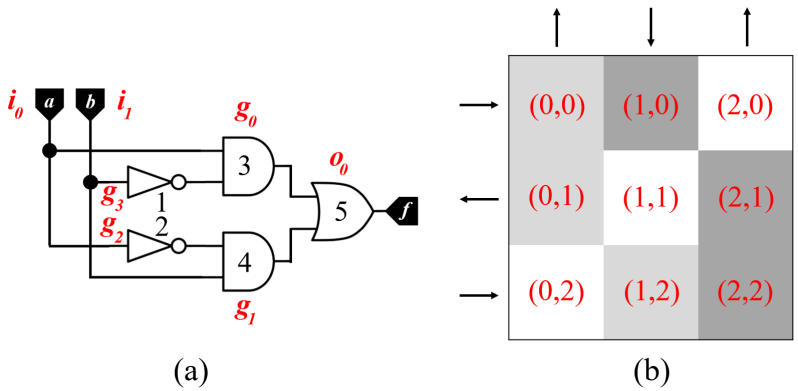


Figure 52 – Resulting QCA circuit implementing the 2:1 MUX circuit.

APPENDIX C – Example: NML design

C.1 Input netlist and design conditions

Let us consider the XOR2 netlist illustrated in Figure 53 (a) as the input of the proposed method. Along with that, consider the design space presented in Figure 53 (b) as the current state following the Algorithm 5.3.4. Furthermore, let us consider three two available for design.



(a) (b)

Figure 53 – XOR2 (a) and design space available in the current algorithm iteration (b).

In this scenario, let us define the placement, routing, and synchronization constraints as follows.

C.2 Placement constraints

- *Single driver constraint:* each driver should be placed at exactly one location. Thus, we have

$$exactly(1, \{i_{0(d)}(0,0), i_{0(d)}(0,1), ..., i_{0(d)}(3,2)\}),$$
$$exactly(1, \{i_{1(d)}(0,0), i_{1(d)}(0,1), ..., i_{1(d)}(3,2)\}),$$
$$exactly(1, \{g_{0(d)}(0,0), g_{0(d)}(0,1), ..., g_{0(d)}(3,2)\}),$$
$$...$$
$$exactly(1, \{g_{3(d)}(0,0), g_{3(d)}(0,1), ..., g_{3(d)}(3,2)\}).$$
$$exactly(1, \{o_{0(d)}(0,0), o_{0(d)}(0,1), ..., o_{0(d)}(3,2)\}),$$

- *Overlapping constraint:* each grid cell contain at most one majority or (exclusively) at most two inverters/wires. Thus, we have

$$
\begin{aligned}
& g_{0(d)}(0,0) \implies \\
& exactly(1, \{g_{0(d)}(0,0), ..., g_{3(d)}(0,0), o_{0(d)}(0,0)\}) \wedge \\
& exactly(0, \{i_{0(w)}(0,0), i_{1(w)}(0,0), g_{0(w)}(0,0), ..., g_{3(w)}(0,0), o_{0(w)}(0,0)\}), \\
& \qquad\qquad\qquad\qquad\qquad\qquad ... \\
& o_{0(d)}(2,2) \implies \\
& exactly(1, \{g_{0(d)}(2,2), ..., g_{3(d)}(2,2), o_{0(d)}(2,2)\}) \wedge \\
& exactly(0, \{i_{0(w)}(2,2), i_{1(w)}(2,2), g_{0(w)}(2,2), ..., g_{3(w)}(2,2), o_{0(w)}(2,2)\})
\end{aligned}
$$

  for the majority-based gates,

$$
\begin{aligned}
g_{2(d)}(0,0) \implies & atmost(2, \{i_{0(d)}(0,0), i_{1(d)}(0,0), g_{2(d)}(0,0), g_{3(d)}(0,0), i_{0(w)}(0,0), \\
& i_{1(w)}(0,0), g_{0(w)}(0,0), g_{1(w)}(0,0)\}), \\
& \qquad\qquad ... \\
g_{3(d)}(2,2) \implies & atmost(2, \{i_{0(d)}(2,2), i_{1(d)}(2,2), g_{2(d)}(2,2), g_{3(d)}(2,2), i_{0(w)}(2,2), \\
& i_{1(w)}(2,2), g_{0(w)}(2,2), g_{1(w)}(2,2)\}),
\end{aligned}
$$

  for the inverter-based gates, and

$$
\begin{aligned}
i_{0(d)}(0,0) & \implies atmost(2, \{i_{0(d)}(0,0), i_{1(d)}(0,0)\}), \\
& \qquad ... \\
i_{1(d)}(2,2) & \implies atmost(2, \{i_{0(d)}(2,2), i_{1(d)}(2,2)\}),
\end{aligned}
$$

  for the inputs.

- *Bottom majority constraint:* majority gates should not be placed in locations containing the same phase of the cell above. Thus, we have

$$
\begin{aligned}
& exactly(0, \{g_{0(d)}(0,1), g_{1(d)}(0,1), o_{0(d)}(0,1)\}, \\
& exactly(0, \{g_{0(d)}(2,2), g_{1(d)}(2,2), o_{0(d)}(2,2)\}.
\end{aligned}
$$

- *Majority near majority constraint:* majority gates should not be placed alongside other majorities. Thus, we have

$$
\begin{aligned}
g_{0(d)}(0,0) & \implies exactly(0, \{g_{1(d)}(1,0), g_{1(d)}(0,1), g_{1(d)}(1,1), o_{0(d)}(1,0), ..., o_{0(d)}(1,1)\}), \\
& \qquad\qquad ... \\
o_{0(d)}(2,2) & \implies exactly(0, \{g_{1(d)}(1,2), g_{1(d)}(2,1), g_{1(d)}(1,1), g_{2(d)}(1,2), ..., g_{2(d)}(1,1)\}).
\end{aligned}
$$

- *Wire near majority constraint:* wires should not be placed alongside a majority gate in the tracks presented in Figure 43.

$$\begin{aligned} g_{0(d)}(0,0) \implies & \neg(nw_{(0)}(1,0) \wedge nw_{(0)}(1,1)) \wedge \\ & \neg(wn_{(0)}(0,1) \wedge wn_{(0)}(1,1)) \wedge \\ & \neg(nw_{(0)}(1,1) \wedge wn_{(0)}(1,1)) \wedge \\ & \neg(nw_{(1)}(1,0) \wedge nw_{(1)}(1,1)) \wedge \\ & \neg(wn_{(1)}(0,1) \wedge wn_{(1)}(1,1)) \wedge \\ & \neg(nw_{(1)}(1,1) \wedge wn_{(1)}(1,1)), \\ & \ldots \\ o_{0(d)}(2,2) \implies & \neg(ne_{(0)}(1,2) \wedge ws_{(0)}(2,1)) \wedge \\ & \neg(ne_{(1)}(1,2) \wedge ws_{(1)}(2,1)). \end{aligned}$$

- *Wire near wire constraint:* a pair of wires should not be placed in the positions presented in Figure 44.

$$\begin{aligned} & \neg((ne_{(0)}(0,0) \wedge ne_{(0)}(0,1)) \wedge (nw_{(1)}(1,0) \wedge nw_{(1)}(1,1))), \\ & \neg((ne_{(1)}(0,0) \wedge ne_{(1)}(0,1)) \wedge (nw_{(0)}(1,0) \wedge nw_{(0)}(1,1))), \\ & \ldots \end{aligned}$$

  for dealing with the vertical wires and

$$\begin{aligned} & \neg((ws_{(0)}(0,0) \wedge ws_{(0)}(1,0)) \wedge (wn_{(1)}(0,1) \wedge wn_{(1)}(1,1))), \\ & \neg((ws_{(1)}(0,0) \wedge ws_{(1)}(1,0)) \wedge (wn_{(0)}(0,1) \wedge wn_{(0)}(1,1))), \\ & \ldots \end{aligned}$$

  for dealing with the horizontal wires.

C.3 Synchronization constraints

- *Driver synchronization constraint:* for each gate or output driver, its inputs should be submitted to the same clock cycle if the cell is on phases 1 or 2 and to the previous or the same clock cycle if the cell is on phase 0 (depending on the position of the driver). Thus, expanding the equations for the output node considering the

case where $c = 1$ (second cycle), we have

$$\begin{aligned} o^1_{0(d)}(0,0) &\implies g^1_0(0,1) \wedge g^1_1(0,1), \\ o^1_{0(d)}(1,0) &\implies g^1_0(0,0) \wedge g^1_1(0,0), \\ o^1_{0(d)}(2,0) &\implies (g^0_0(1,0) \vee g^0_0(2,1)) \wedge (g^0_1(1,0) \vee g^0_1(2,1)), \\ &\ldots \\ o^1_{0(d)}(2,2) &\implies \implies g^1_0(1,2) \wedge g^1_1(1,2), \end{aligned}$$

such that the same is valid for all the gate nodes, as well as for all cycles (which ranges from 0 to 1 since there are two cycles available for design as defined previously in this example).

- *Wire synchronization constraint:* for each wire cell, its input should be submitted to the same clock if the cell is submitted to phase 1 or 2 and to the previous or the same clock cycle if the cell is submitted to phase 0 (depending on the position of the driver). Thus, expanding the equations for $c = 1$ (second cycle), we have

$$\begin{aligned} o^1_{0(w)}(0,0) &\implies o^1_0(0,1), \\ o^1_{0(w)}(1,0) &\implies o^1_0(0,0), \\ o^1_{0(w)}(2,0) &\implies o^0_0(1,0) \vee o^0_0(2,1), \\ &\ldots \\ o^1_{0(w)}(2,2) &\implies \implies o^1_0(1,2), \end{aligned}$$

  such that the same is valid for all inputs, gates, and outputs, as well as for all cycles.

- *Single-cycle constraint:* each cell should operate under exactly one cycle. Thus, we have

$$\begin{aligned} g_0(0,0) &\implies exactly(1, \{g^0_0(0,0), g^1_0(0,0)\}), \\ g_0(1,0) &\implies exactly(1, \{g^0_0(1,0), g^1_0(1,0)\}), \\ &\ldots \\ g_0(2,2) &\implies exactly(1, \{g^0_0(2,2), g^1_0(2,2)\}), \end{aligned}$$

  such that the same is valid for all inputs, gates, and outputs.

- *Input/output phase synchronization constraint:* each input should be submitted to

the same phase. The same is valid for each output. Thus,

$$
\begin{aligned}
i_{0(d)}(0,0) &\implies i_{1(d)}(0,0) \vee i_{1(d)}(0,1) \vee i_{1(d)}(1,2),\\
i_{0(d)}(1,0) &\implies i_{1(d)}(1,0) \vee i_{1(d)}(2,1) \vee i_{1(d)}(2,2),\\
i_{0(d)}(2,0) &\implies i_{1(d)}(2,0) \vee i_{1(d)}(1,1) \vee i_{1(d)}(0,2),\\
&\ldots\\
i_{0(d)}(2,2) &\implies i_{1(d)}(2,2) \vee i_{1(d)}(1,0) \vee i_{1(d)}(2,1),
\end{aligned}
$$

such that the same is applied for the other input and for the output.

- *Input/output cycle synchronization constraint:* each input should start operating on the same clock cycle. The same is valid for each output. Thus, we have

$$
\begin{aligned}
i^0_{0(d)}(0,0) &\implies i^0_{1(d)}(0,0) \vee i^0_{1(d)}(1,0) \vee \ldots \vee i^0_{1(d)}(2,2),\\
&\ldots\\
i^0_{0(d)}(2,2) &\implies i^0_{1(d)}(0,0) \vee i^0_{1(d)}(1,0) \vee \ldots \vee i^0_{1(d)}(2,2),\\
i^1_{0(d)}(0,0) &\implies i^1_{1(d)}(0,0) \vee i^1_{1(d)}(1,0) \vee \ldots \vee i^1_{1(d)}(2,2),\\
&\ldots\\
i^1_{0(d)}(2,2) &\implies i^1_{1(d)}(0,0) \vee i^1_{1(d)}(1,0) \vee \ldots \vee i^1_{1(d)}(2,2).
\end{aligned}
$$

C.4 Routing constraints

- *No return constraint:* a signal should not enter and exit a cell by the same side. Thus, we have

$$
\begin{aligned}
&\neg(ws_{(0)}(0,0) \wedge wn_{(0)}(0,0)) \wedge \neg(nw_{(0)}(0,0) \wedge ne_{(0)}(0,0)) \wedge\\
&\neg(ws_{(1)}(0,0) \wedge wn_{(1)}(0,0)) \wedge \neg(nw_{(1)}(0,0) \wedge ne_{(1)}(0,0)) \wedge\\
&\neg(ws_{(0)}(1,0) \wedge wn_{(0)}(1,0)) \wedge \neg(nw_{(0)}(1,0) \wedge ne_{(0)}(1,0)) \wedge\\
&\neg(ws_{(1)}(1,0) \wedge wn_{(1)}(1,0)) \wedge \neg(nw_{(1)}(1,0) \wedge ne_{(1)}(1,0)) \wedge\\
&\ldots\\
&\neg(ws_{(0)}(2,2) \wedge wn_{(0)}(2,2)) \wedge \neg(nw_{(0)}(2,2) \wedge ne_{(0)}(2,2)) \wedge\\
&\neg(ws_{(1)}(2,2) \wedge wn_{(1)}(2,2)) \wedge \neg(nw_{(1)}(2,2) \wedge ne_{(1)}(2,2)).
\end{aligned}
$$

- *Border constraint:* ports located on the border of the routing area should not be

connected.

$$\begin{gathered}\neg nw(0,0) \wedge \neg ne(0,0) \wedge \neg ws(0,0) \wedge \neg wn(0,0) \wedge \\ \neg nw(1,0) \wedge \neg ne(1,0) \wedge \neg ws(0,1) \wedge \neg wn(0,1) \wedge \\ \neg nw(2,0) \wedge \neg ne(2,0) \wedge \neg ws(0,2) \wedge \neg wn(0,2) \wedge \\ \neg nw(2,0) \wedge \neg ne(2,0).\end{gathered}$$

- *Invalid corners constraint:* the routing paths presented in Figure 26 are not allowed. Thus, we have

$$\begin{gathered}\neg((nw_{(0)}(0,1) \vee ne_{(0)}(0,1)) \wedge (ws_{(0)}(0,0) \vee wn_{(0)}(0,0))), \\ \neg((nw_{(0)}(2,1) \vee ne_{(0)}(2,1)) \wedge (ws_{(0)}(2,0) \vee wn_{(0)}(2,0))), \\ \neg((nw_{(0)}(0,2) \vee ne_{(0)}(0,2)) \wedge (ws_{(0)}(1,1) \vee wn_{(0)}(1,1))), \\ \neg((nw_{(0)}(1,1) \vee ne_{(0)}(1,1)) \wedge (ws_{(0)}(2,1) \vee wn_{(0)}(2,1))), \\ \neg((nw_{(0)}(1,2) \vee ne_{(0)}(1,2)) \wedge (ws_{(0)}(1,2) \vee wn_{(0)}(1,2))), \\ \neg((nw_{(1)}(0,1) \vee ne_{(1)}(0,1)) \wedge (ws_{(1)}(0,0) \vee wn_{(1)}(0,0))), \\ \neg((nw_{(1)}(2,1) \vee ne_{(1)}(2,1)) \wedge (ws_{(1)}(2,0) \vee wn_{(1)}(2,0))), \\ \neg((nw_{(1)}(0,2) \vee ne_{(1)}(0,2)) \wedge (ws_{(1)}(1,1) \vee wn_{(1)}(1,1))), \\ \neg((nw_{(1)}(1,1) \vee ne_{(1)}(1,1)) \wedge (ws_{(1)}(2,1) \vee wn_{(1)}(2,1))), \\ \neg((nw_{(1)}(1,2) \vee ne_{(1)}(1,2)) \wedge (ws_{(1)}(1,2) \vee wn_{(1)}(1,2))).\end{gathered}$$

- *Majority access constraint:* in a cell containing a majority gate, two neighbor ports should not be connected if they are located in the corners of this BANCS cell. Thus, we have

$$\begin{aligned}g_{0(d)}(0,0) \implies & \neg(wn(0,0) \wedge nw(0,0)) \wedge \neg(ne(0,0) \wedge wn(1,0)) \wedge \\ & \neg(ws(0,0) \wedge nw(0,1)) \wedge \neg(ws(1,0) \wedge ne(0,1)), \\ g_{0(d)}(1,0) \implies & \neg(wn(1,0) \wedge nw(1,0)) \wedge \neg(ne(1,0) \wedge wn(2,0)) \wedge \\ & \neg(ws(1,0) \wedge nw(1,1)) \wedge \neg(ws(2,0) \wedge ne(1,1)), \\ & \ldots \\ g_{1(d)}(2,2) \implies & \neg(wn(2,2) \wedge nw(2,2)),\end{aligned}$$

  such that the same is applied for the output node.

- *Gate driver adjacency constraint:* each gate should be followed by the cells connected to its output or by its wires and should be preceded by the cells connected

to its inputs.

$$
\begin{aligned}
g_{0(d)}(0,0) &\implies exactly(1,\{i_0(0,1)\}) \wedge exactly(1,\{g_3(0,1)\}),\\
g_{0(d)}(1,0) &\implies exactly(1,\{i_0(0,0)\}) \wedge exactly(1,\{g_3(0,1)\}),\\
g_{0(d)}(2,0) &\implies exactly(1,\{i_0(1,0),i_0(2,1)\}) \wedge exactly(1,\{g_3(1,0),g_3(2,1)\}),\\
&\ldots\\
g_{0(d)}(2,2) &\implies exactly(1,\{i_0(2,1)\}) \wedge exactly(1,\{g_3(2,1)\}),
\end{aligned}
$$

such that the same is applied for the other gates and output node, and

$$
\begin{aligned}
g_{0(d)}(0,0) &\implies exactly(1,\{o_0(1,0)\}),\\
g_{0(d)}(1,0) &\implies exactly(1,\{o_0(2,0),o_0(1,1)\}),\\
g_{0(d)}(0,1) &\implies exactly(1,\{o_0(0,0)\}),\\
&\ldots\\
g_{0(d)}(2,2) &\implies exactly(1,\{o_0(2,1)\}),
\end{aligned}
$$

such that the same is also applied for the other gates and output node.

- *Input driver adjacency constraint:* each input driver connected to one gate should be followed by one of its own wires or by the gate. Furthermore, if the input is connected to more than one gate, then it should be followed by one of its own wires or by at least one of the gates connected to its output. Thus, we have

$$
\begin{aligned}
i_{0(d)}(0,0) &\implies i_{0(w)}(1,0) \vee g_{0(d)}(1,0) \vee g_{2(d)}(1,0),\\
i_{0(d)}(1,0) &\implies i_{0(w)}(1,1) \vee i_{0(w)}(2,0) \vee g_{0(d)}(1,1)\\
&\quad \vee g_{0(d)}(2,0) \vee g_{2(d)}(1,1) \vee g_{2(d)}(2,0),\\
&\ldots\\
i_{1(d)}(2,2) &\implies i_{1(w)}(2,1) \vee g_{1(d)}(2,1) \vee g_{3(d)}(2,1).
\end{aligned}
$$

- *Wire adjacency constraint:* each wire should be followed by the cells connected to its output or by other wire of the same signal and should be preceded by a wire

or by a driver of the same signal. Thus,

$$\begin{aligned} i_{0(w)}(0,0) &\implies exactly(1, \{i_{0(w)}(1,0), g_{0(d)}(1,0), g_{2(d)}(1,0)\}), \\ i_{0(w)}(1,0) &\implies exactly(1, \{i_{0(w)}(1,1), i_{0(w)}(2,0), g_{0(d)}(1,1), \\ & \quad g_{0(d)}(2,0), g_{2(d)}(1,1), g_{2(d)}(2,0)\}), \\ & \quad ... \\ i_{1(w)}(2,2) &\implies exactly(1, \{i_{1(w)}(2,1), g_{0(d)}(3,2), g_{2(d)}(2,1)\}) \end{aligned}$$

and

$$\begin{aligned} i_{0(w)}(0,0) &\implies atleast(1, \{i_{0(w)}(0,1), i_{0(d)}(0,1)\}), \\ i_{0(w)}(1,0) &\implies atleast(1, \{i_{0(w)}(0,0), i_{0(d)}(0,0)\}), \\ & \quad ... \\ i_{1(w)}(2,2) &\implies atleast(1, \{i_{1(w)}(1,2), i_{1(d)}(1,2)\}). \end{aligned}$$

- *Long wire constraint:* wires should not be composed of more than five magnets on the same phase. Thus, we have

$$\begin{aligned} i_{0(w)}(0,0) \wedge i_{0(w)}(0,1) &\implies \neg((nw_{(0)}(0,0) \vee ne_{(0)}(0,0)) \wedge (nw_{(0)}(0,2) \vee ne_{(0)}(0,2))) \wedge \\ & \quad \neg((nw_{(1)}(0,0) \vee ne_{(1)}(0,0)) \wedge (nw_{(1)}(0,2) \vee ne_{(1)}(0,2))), \\ i_{0(w)}(0,0) \wedge i_{0(w)}(0,1) &\implies \neg((wn_{(0)}(0,0) \vee wn_{(0)}(1,0)) \wedge (nw_{(0)}(0,2) \vee ne_{(0)}(0,2))) \wedge \\ & \quad \neg((wn_{(1)}(0,0) \vee wn_{(1)}(1,0)) \wedge (nw_{(1)}(0,2) \vee ne_{(1)}(0,2))), \\ i_{0(w)}(0,0) \wedge i_{0(w)}(0,1) &\implies \neg((nw_{(0)}(0,0) \vee ne_{(0)}(0,0)) \wedge (ws_{(0)}(0,1) \vee ws_{(0)}(1,1))) \wedge \\ & \quad \neg((nw_{(1)}(0,0) \vee ne_{(1)}(0,0)) \wedge (ws_{(1)}(0,1) \vee ws_{(1)}(1,1))), \\ i_{0(w)}(2,1) \wedge i_{0(w)}(2,2) &\implies \neg((nw_{(0)}(2,1) \vee ne_{(0)}(2,1)) \wedge (ws_{(0)}(2,2) \vee ws_{(0)}(2,2))) \wedge \\ & \quad \neg((nw_{(1)}(2,1) \vee ne_{(1)}(2,1)) \wedge (ws_{(1)}(2,2) \vee ws_{(1)}(2,2))). \end{aligned}$$

C.5 Satisfiable solution

Following the Algorithm 5.3.4, the conjunction of all formulas is computed through an SAT solver. In this case, the instance shown in this chapter is satisfiable, i.e., there is a valid solution for the NML design which fulfills all the design restrictions. Figure 54 presents the resulting circuit.

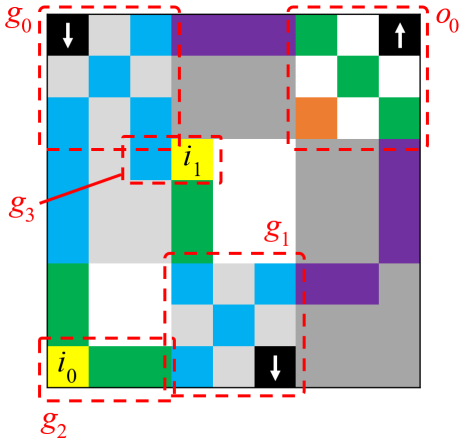


Figure 54 – Resulting NML circuit implementing the XOR2 circuit.

Annexes

ANNEX A – List of publications

A.1 Journal and Conference Publications

CARDOSO, MAICON; BUBOLZ, ANDREI; CORTADELLA, JORDI; ROSA, LEOMAR; MARQUES, FELIPE. **Transistor Placement for Automatic Cell Synthesis through Boolean Satisfiability**. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, Sevilla. 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020. p. 1.

CARDOSO, MAICON S.; SMANIOTTO, GUSTAVO H.; BUBOLZ, ANDREI A. O.; MOREIRA, MATHEUS T.; DA ROSA, LEOMAR S.; MARQUES, FELIPE DE S.. **Libra: an Automatic Design Methodology for CMOS Complex Gates**. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II-EXPRESS BRIEFS, v. 65, p. 1-1, 2018.

CARDOSO, MAICON S.; SMANIOTTO, GUSTAVO H.; BUBOLZ, ANDREI A. O.; DA ROSA JUNIOR, LEOMAR S.; DE MARQUES, FELIPE S.. **Area-Aware Design of Static CMOS Complex Gates**. In: 2018 16th IEEE International New Circuits and Systems Conference (NEWCAS), 2018, Montréal. 2018 16th IEEE International New Circuits and Systems Conference (NEWCAS), 2018. p. 282.

SMANIOTTO, GUSTAVO; ZANANDREA, REGIS; **CARDOSO, MAICON**; DE SOUZA, RENATO; MOREIRA, MATHEUS; MARQUES, FELIPE; DA ROSA, LEOMAR. **Post-processing of supergate networks aiming cell layout optimization**. In: 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017, Baltimore. 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017. p. 1.

CARDOSO, MAICON S.; SMANIOTTO, GUSTAVO H.; MACHADO, JOAO J. DA S.; MOREIRA, MATHEUS T.; DA ROSA, LEOMAR S.; MARQUES, FELIPE DE S.. **Transistor placement strategies for non-series-parallel cells**. In: 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017, Boston. 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017. p. 523-526.

AVILA, CHRISTIANO; CAVALHEIRO, SIMONE; BORDINI, ADRIANA; MARQUES, MONICA; **CARDOSO, MAICON**; FEIJO, GUSTAVO. Metodologias de Avaliação do Pensamento Computacional: uma revisão sistemática. In: XXVIII Simpósio Brasileiro de Informática na Educação SBIE (Brazilian Symposium on Computers in Education), 2017, Recife, 2017. p. 113.

SMANIOTTO, GUSTAVO; ZANANDREA, REGIS; **CARDOSO, MAICON**; DE SOUZA, RENATO; MOREIRA, MATHEUS; MARQUES, FELIPE; DA ROSA, LEOMAR. **A post-processing methodology to improve the automatic design of CMOS gates at layout-level**. In: 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2017, Batumi. 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2017. p. 42.

A.2 Symposium Publications

BUBOLZ, A. A. O.; SMANIOTTO, G. H.; ROSA JUNIOR, L. S.; **CARDOSO, M. S.**; MARQUES, F. S.. **Implementation of a Transistor Placement Method Based on Boolean Satisfiability into ASTRAN CAD Tool**. In: 18th Microelectronics Students Forum, 2018, Bento Gonçalves. Proceedings 18th SForum, 2018.

CARDOSO, M. S.; SMANIOTTO, G. H.; MACHADO, J. J. S.; MOREIRA, M. T.; ROSA JUNIOR, L. S.; MARQUES, F. S.. **Analysis of the Impacts of Diffusion and Polysilicon Gaps in Non-Series-Parallel Supergates**. In: 32th South Symposium on Microelectronics, 2017, Rio Grande. Proceedings 32th SIM, 2017.

SMANIOTTO, G. H.; ZANANDREA, R.; **CARDOSO, M. S.**; SOUZA, R. S.; MOREIRA, M. T.; MARQUES, F. S.; ROSA JUNIOR, L. S.. **Kernel Finder Post-processing Aiming Cell Layout Optimization**. In: 32th South Symposium on Microelectronics, 2017, Rio Grande. Proceedings 32th SIM, 2017.