UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JODY MAICK ARAUJO DE MATOS

Graph-Based Algorithms to Efficiently Map VLSI Circuits with Simple Cells

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Andre Inacio Reis

Porto Alegre
January 2018

CIP — CATALOGING-IN-PUBLICATION

Matos, Jody Maick Araujo de

Graph-Based Algorithms to
Efficiently Map VLSI Circuits
with Simple Cells / Jody Maick Araujo de Matos. – Porto Alegre: PPGC da UFRGS, 2018.

98 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Advisor: Andre Inacio Reis.

1. Graph-based algorithms. 2. Logic synthesis. 3. Technology mapping. 4. Standard cell library. 5. Simple cells. I. Reis, Andre Inacio. II. Título.

ABSTRACT

This thesis introduces a set of graph-based algorithms for efficiently mapping VLSI circuits using simple cells. The proposed algorithms are concerned to, first, effectively minimize the number of logic elements implementing the synthesized circuit. Then, we focus a significant effort on minimizing the number of inverters in between these logic elements. Finally, this logic representation is mapped into a circuit comprised of only two-input NANDs and NORS, along with the inverters. Two-input XORs and XNORs can also be optionally considered. As we also consider sequential circuits in this work, flip-flops are taken into account as well. Additionally, with high-effort optimization on the number of logic elements, the generated circuits may contain some cells with unfeasible fanout for current technology nodes. In order to fix these occurrences, we propose an area-oriented, level-aware algorithm for fanout limitation. The proposed algorithms were applied over a set of benchmark circuits and the obtained results have shown the usefulness of the method. We show that efficient implementations in terms of inverter count, transistor count, area, power and delay can be generated from circuits with a reduced number of both simple cells and inverters, combined with XOR/XNOR-based optimizations. The proposed buffering algorithm can handle all unfeasible fanout occurrences, while (i) optimizing the number of added inverters; and (ii) assigning cells to the inverter tree based on their level criticality. When comparing with academic and commercial approaches, we are able to simultaneously reduce the average number of inverters, transistors, area, power dissipation and delay up to 48%, 5%, 5%, 5%, and 53%, respectively. As the adoption of a limited set of simple standard cells have been showing benefits for a variety of modern VLSI circuits constraints, such as layout regularity, routability constraints, and/or ultra low power constraints, the proposed methods can be of special interest for these applications. Additionally, some More-than-Moore applications, such as printed electronics designs, can also take benefit from the proposed approach.

Keywords: Graph-based algorithms. logic synthesis. technology mapping. standard cell library. simple cells.

Algoritmos Baseados em Grafos para Mapear Eficientemente Circuitos VLSI com Portas Simples

RESUMO

Essa tese introduz um conjunto de algoritmos baseados em grafos para o mapeamento eficiente de circuitos VLSI com células simples. Os algoritmos propostos se baseiam em minimizar de maneira eficiente o número de elementos lógicos usados na implementação do circuito. Posteriormente, uma quantidade significativa de esforço é aplicada na minimização do número de inversores entre esses elementos lógicos. Por fim, essa representação lógica é mapeada para circuitos compostos somente por células NAND e NOR de duas entradas, juntamente com inversores. Células XOR e XNOR de duas entradas também podem ser consideradas. Como nós também consideramos circuitos sequenciais, flips-flops também são levados em consideração. Com o grande esforço de minimização de elementos lógicos, o circuito gerado pode conter algumas células com um fanout impraticável para os nodos tecnológicos atuais. Para corrigir essas ocorrências, nós propomos um algoritmo de limitação de fanout que considera tanto a área sendo utilizada pelas células quanto a sua profundidade lógica. Os algoritmos propostos foram aplicados sobre um conjunto de circuitos de benchmark e os resultados obtidos demonstram a utilidade dos métodos. Nós mostramos que implementações eficientes em termos de número de inversores, número de transistores, área, potência e atraso podem ser obtidas a partir de circuitos um número reduzido tanto de células simples quanto de inversores, combinados com optimizações baseadas em XORs e XNORs. O algoritmo de buffering proposto pode resolver todas as ocorrências de violação de fanout (i) enquanto otimiza o número de inversores inseridos no processo; e (ii) atribui as células aos seus inversores considerando suas profundidades lógicas. Quando comparado a abordagens tanto da academia quanto comerciais, o trabalho proposto é capaz de reduzir simultaneamente o número médio de inversores, transistores, área, potência e atraso em até 48%, 4%, 8%, 14%, and 16%, respectivamente. Como a adoção de um conjunto limitado de células simples tem mostrado benefícios para uma variedade de restrições de circuitos modernos, como regularidade de leiaute, restrições de roteabilidade, e/ou restrições de consumo de potência ultra baixos, os métodos propostos podem ser de especial interesse para essas aplicações. Adicionalmente, algumas aplicações More-

than-Moore, tais como circuitos baseados em eletrônica impressa, também podem ser beneficiadas pela abordagem proposta.

Palavras-chave: Algoritmos baseados em grafos, síntese lógica, mapeamento tecnológico, biblioteca de células, células simples.

LIST OF ABBREVIATIONS AND ACRONYMS

AIG	And-Inverter Graph
ASIC	Application Specific Integrated Circuit
BDD	Binary Decision Diagram
CMOS	Complementary Metal-Oxide-Semiconductor
DAG	Directed Acyclic Graph
EDA	Electrical Design Automation
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
OTR	Odd-level Transistor Replacement
PI	Primary Input
PO	Primary Output
PTL	Pass-Transistor Logic
RTL	Register-Transfer Level
SOP	Sum-Of-Products
TSBDD	Terminal-Suppressed Binary Decision Diagram
VLSI	Very-Large Scale Integration
XAIG	Xor-And-Inverter Graph

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

1 INTRODUCTION

The semiconductor industry has notably reached a distinctive position among other industries over the last decades (ITRS, 2015b). Much of this is due to both the speed in which improvements are achieved in its products and the industry's ability to exponentially decrease the minimum feature sizes used to fabricate integrated circuits. Among a number of trends related to these facts, the most frequently cited trend is the integration level, which is usually expressed as Moore's Law (MOORE, 1965).

In parallel, the electronic design automation (EDA) industry also needed to follow the evolution pace of semiconductors. This is mainly due to two reasons. First, as the semiconductors industry evolve, the designs complexity increase and this is also the driving force for innovations in EDA tools and design methodologies. Additionally, much of the advances in microelectronics would not be possible without innovation at the EDA side. This creates a virtuous cycle for both the industries.

However, the miniaturization and all the related trends have impact on the complexity while designing very-large scale integration (VLSI) circuits, so that designing integrated circuits is becoming an increasingly hard task (ITRS, 2015b; KAHNG et al., 2011). A large portion of this complexity is due to manufacturability issues and a continuously increasing size of the design rules manual (DRM), with more complex and conservative rules at each new technology node. This, together with a number of other factors, such as the quest for more energy efficient circuits, makes that some VLSI applications arise with constraints so their circuits either need or would benefit from a synthesis targeting only a limited set of simple cells. Thus, the study of efficient methods and algorithms to map VLSI circuits only with simple cells can be of special interest for these applications.

This chapter is organized as follows. Some important VLSI applications constrained to simple cells are introduced, in order to better understand where the proposed contributions lie on. After, we detail the main motivations and present the objectives of this thesis. This chapter ends by summarizing the organization of this text.

1.1 VLSI Applications Constrained to Simple Cells

In this section, we summarize some of the VLSI applications which may either be constrained to simple cells or could benefit from a synthesis with a small set of simple cells. The algorithms and methods presented in this thesis can be of special interest for designs under the constraints presented in the following subsections.

1.1.1 Design for Manufacturability and Layout Regularity

In advanced technology nodes, the transistor performance has been varying from its nominal behavior according to the device's proximity to certain adjacent features. Printed patterns can significantly and non-systematically differ from drawn patterns due to neighboring effects. This phenomenon was labeled as local layout effects (LLEs).

From 16nm technology node and below, LLEs have been hardly influencing the design of the basic building blocks of digital circuitry: the cells from standard cell libraries. In order to look for LLEs, the traditional, intracell rule checking is not sufficient anymore and intercell compatibility has been demanding for checks of layout interactions across cell boundaries. So, design for manufacturability (DFM) metrics have been adopted and either silicon or virtual characterization are used to identify regular layout patterns causing LLEs.

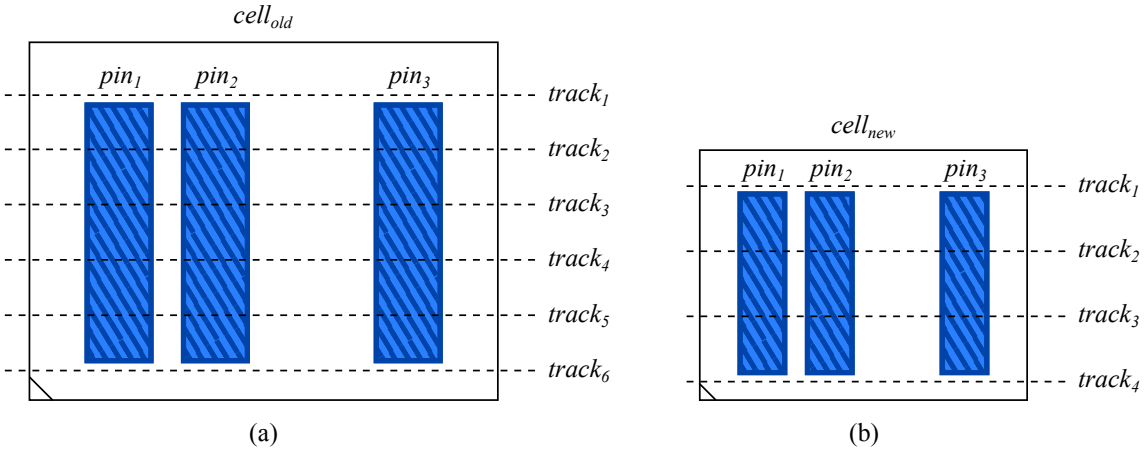
One way to mitigate these problems relies on the usage of a limited set of simple cells. Studies have been showing that these layout regularity and manufacturability constraints can be mitigated (with minimal impact on circuit performance) if a limited set of cells with regular, non-LLE critical patterns are adopted during the synthesis process (PAGLIARINI; MARTINS; PILEGGI, 2017; XU et al., 2016; CHAVA et al., 2015). Thus, a standard cell library comprised of a limited set of simple cells would both decrease the characterization effort and generate more regular, DFM-friendly layouts. Standard cell libraries with high intracell and intercell DFM quality will translate into chips with equally high DFM quality.

1.1.2 Design Routability and Cell Pin Access

As the technology advances, the downscaling pace has been different for some critical technology features. A good example is the slower scaling of metal spacing rules, which is also related to DFM. In this case, although the minimum feature size of metal layers has been scaled by the given technology node factor, some DFM-based constraints do not allow the metal spacing rules to scale by the same factor. This may further result in metal layers with less routing resources when compared to a previous technology.

Along with this, it is also worth to mention the regular goal of designing area-minimal standard cell layout. These aspects together are creating design routability issues and cell pin access challenges. To illustrate these cases, consider the context in Figure 1.1. $cell_{old}$ depicts the pin access in an older technology node, where each pin is accessible through four points. Then, $cell_{new}$ illustrates the pin access in a newer technology node, which was downscaled from the previous one. Notice that the metal spacing rules did not scale by the same factor as the technology and each pin of $cell_{new}$ has only two access points now (HSU et al., 2014).

Figure 1.1: Abstract layouts for cells in two different technology nodes.



Thus, standard cell pin access has become one of the most challenging issues for the back-end physical design in sub-14nm technology nodes (XU et al., 2016; PAN et al., 2015; YE et al., 2015; QIU; MAREK-SADOWSKA, 2013). These issues are notably highlighted (1) with unidirectional layout restrictions (XU; PAN, 2017); and (2) when a high number of complex cells (with a high number of pins) are concentrated in a given region of the circuit, creating high-pin-density regions.

One way to approach these issues relies on designing larger cells, what would

make the cell pins intersect more routing tracks and increase the pins accessibility. In contrast, this approach may not align with the goal of designing area-minimal standard cell layout and tends to deliver larger circuits even on regions that are not suffering from pin accessibility issues. Thus, in order to address this problem without the penalty of increasing the area of the entire design, the circuit regions suffering with routability issues due to cell pin access problems can be resynthesized with simple cells (with a reduced number of I/O pins). Notice that a cell with 5 pins (as an AOI22) requires that 5 nets compete for routing resources in a limited layout area, which tends to increase routing congestion issues. Although a simple-cell-based implementation would derive an 8-pin solution, these 8 nets would compete for routing resources in a larger layout area, decreasing routing congestion issues.

1.1.3 Sub/Near-Threshold Voltage Designs

Currently, one of the important barriers for the continuous CMOS downscaling is related to energy and power dissipation. Energy considerations are vital over a wide spectrum of applications, from high-performance platforms and the high energy consumption on data centers, to sensor-based platforms, which are critically dependent on both ultra-low power ($\leq \mu W$ in standby) and reduced form-factor (mm^3) (DRESLINSKI et al., 2010; MARKOVIC et al., 2010).

Voltage scaling has become one of the most effective methods to reduce power consumption due to two main reasons: (1) dynamic power is the most influencing component of power consumption in modern VLSI designs; and (2) the dynamic power can be reduced quadratically by reducing the supply voltage (DRESLINSKI et al., 2010; MARKOVIC et al., 2010). Emerging strategies rely on making the circuit devices to operate near (or even under) their threshold voltages. Although sub/near-threshold voltage designs are still suffering for major performance drawbacks, these strategies have been applied on some niche markets, such as sensor and biomedical applications that require ultralow energy (PAUL et al., 2017; PAUL et al., 2016; LIU et al., 2012b; HULZINK et al., 2011).

Still, a number of measures need to be taken into account so that sub/near-threshold voltage designs achieve the desired energy efficiency. As the number of either parallel or stacked transistors increases, the leakage current variability increases dramatically (STANGHERLIN, 2013; LIU et al., 2012a; KAUL et al., 2012;

ALIOTO, 2012; LUTKEMEIER et al., 2013; LOTZE; MANOLI, 2012; KWONG et al., 2009; FUKETA et al., 2011; PU et al., 2010; KWONG; CHANDRAKASAN, 2006; WANG; CALHOUN; CHANDRAKASAN, 2006). Due to this reason, designs based on sub/near threshold voltage are commonly synthesized with small libraries comprised of simple cells, with at most 2 stacked transistors, either in series or in parallel, in their pull-up PMOS and pull-down NMOS networks.

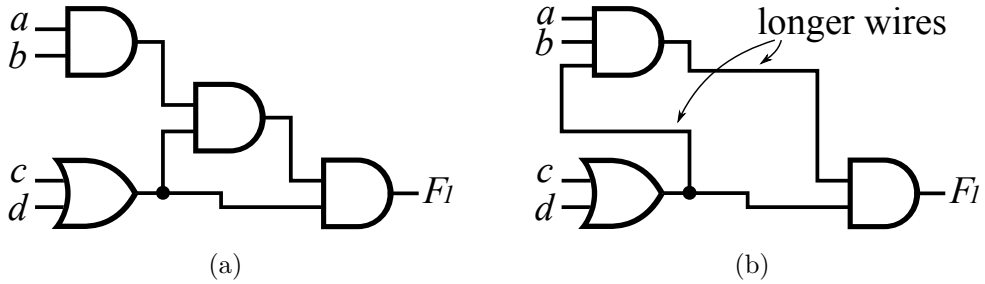
1.1.4 Simple Cells to Address Long Wires

In previous technology nodes, the main source of circuit delay was the logic cells and, at that time, the wire delay was almost negligible. As the technology has been advancing, the interconnect delay was becoming more and more perceptive at critical paths. In recent technology nodes, the wire delay is already dominating the overall circuit performance. This is specially true when observing the delay of long wires (WANG; CHANG; CHENG, 2009).

A very established approach to address this problem relies on splitting long wires into shorter interconnects by inserting buffers/repeaters. Extensive literature exists on optimal buffering for wires (ALPERT; DEVGAN; QUAY, 1999; ISMAIL; FRIEDMAN, 1998; NALAMALPU; BURLESON, 2000). However, this solution is becoming impractical. The number of repeaters required for this matter is exponentially increasing with each technology step and, nowadays, 10∼15% of the cells in large microprocessor chips are buffers that break down long interconnects (SYLVESTER; KEUTZER, 1999; SAXENA et al., 2004; SEO et al., 2008). Moreover, these repeaters are commonly inserted once the circuit is already placed. Thus, even in the cases that the cells were optimally placed, a considerably high room should be found to insert the repeaters and possibly the entire design should placed again (WANG; CHANG; CHENG, 2009).

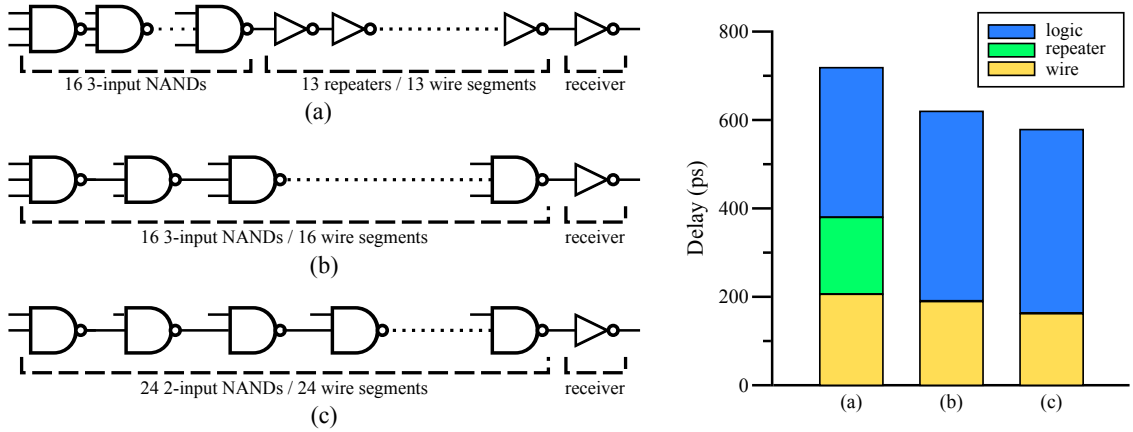
Another way to address this problem is to use simple cells to obtain shorter wires. As pointed out by Plaza, Markov and Bertacco (2008), conglomerating small cells into a large cell may produce non-monotonic interconnects, as depicted in Figure 1.2, and this may affect the circuit delay and routability. Conversely, a clever usage of simple cells is able to deliver circuits which are faster than the repeater-based solution, as illustrated in Figure 1.3. Thus, designs suffering with long wire issues can also take benefit from an algorithm to map VLSI circuits with a limited

Figure 1.2: The usage of simple cells (a) compared with the usage of more complex cells (b), which can derive longer wires.



Source: Adapted from (SEO et al., 2008).

Figure 1.3: Three schemes for comparison of a single path: (a) logic block driving an optimally repeated wire; (b) the same logic block placed along the wire; and (c) the logic block decomposed into even simpler cells and also placed along the wire.



Source: Adapted from (SEO et al., 2008).

set of simple standard cells.

1.1.5 More-than-More and Printed Electronics Applications

The trend on the semiconductor industry towards device miniaturization and its associated benefits in performance has been labeled as More Moore (ITRS, 2015a). Recently, a new trend labeled as More-than-Moore brings a novel perspective for functional diversification of semiconductor-based devices in order to complement the integrated systems capabilities. In this case, heterogeneous functionalities can provide additional value in different ways without necessarily scaling according to Moore's Law (FAN et al., 2016; CAO; XUE, 2014; KIM et al., 2013; MARINOV, 2015; SMITH et al., 2015).

In this context, printed electronics has been emerging as a comprehensive

More-than-Moore technology by exploring the possibility of hybrid fabrication processes, such as conventional clean-room approaches together with the benefits of digital printing techniques (ZHU; YAO; ZHANG, 2000; HAMBSCH et al., 2010; MOONEN; YAKIMETS; HUSKENS, 2012). Even so, there are several limitations and challenges related to either material or technological processes for developing high performance organic devices and circuits. Also, low charge carrier mobility of existing organic materials (polymers and small molecules), especially for n-type organic semiconductors, is also a limitation for implementing organic transistors and complex circuits (SIRRINGHAUS et al., 2000).

The aforementioned limitations constraint application-specific printed electronics circuits (ASPECs) to be designed only with small libraries comprised of simple cells. These applications are commonly synthesized using inverters and NANDs with a different number of inputs (LLAMAS et al., 2014a; MASHAYEKHI et al., 2014; LLAMAS et al., 2014b; LLAMAS et al., 2015). This is mainly due to the adopted pseudo PMOS logic style (WU; ZHANG; QIU, 2006) and parasitic capacitances.

It is also worth to mention that transistor count minimization is of special interest for ASPECs due to two main reasons. First, many of the printing technologies are limited to the feature sizes of some microns (or even tens of microns). Thus, with fairly large transistors, any optimization on the transistor count would further save significant space on the substrate. Also, printed electronics devices are still suffering from high process variability and low or mid-yield. So, increasing the number of transistors increases the probability of hard faults and can lead to even lower yield at transistor level.

1.2 Motivation

Current technology mappers for VLSI circuits are mainly concerned to address the vast majority of applications that can account for complex cells. For these applications, the minimization of inverters in between complex cells plays a completely different role when compared to the case of simple cells. For applications based on complex cells, concerns on the number of inverters are significantly less relevant, since they represent a smaller portion of the entire circuit.

In contrast, the optimization of polarity assignments in between simple cells

is a lot more important. As these inverters represent a larger portion of the circuits, more inverters would be worse for a number of different cost functions, such as transistor count, area and power consumption. Also, a clever usage of inverters in this case can also bring benefits in terms of performance.

1.3 Objective

The objective of this thesis is to introduce a set of algorithms for efficiently mapping VLSI circuits based on simple cells. These algorithms are concerned to, first, effectively minimize the number of logic elements implementing the synthesized circuit. Then, we focus a significant effort on minimizing the number of inverters in between these logic elements. Finally, this logic representation is mapped into a circuit comprised of only two-inputs NANDs and NORS, along with the inverters. Two-input XORs and XNORs can be also optionally considered. As we also consider sequential circuits in this work, flip-flops are taken into account as well. Additionally, with high-effort optimization on the number of logic elements, the generated circuits may contain some cells with unfeasible fanout in current technology nodes. In order to fix these occurrences, we propose an area-oriented, level-aware algorithm for fanout limitation. The proposed algorithms can bring benefits for most of the VLSI applications constrained to simple cells referred in this work.

1.4 Thesis Organization

The next chapters are organized as follows:

Chapter 2: *BACKGROUND AND PRELIMINARIES* — reviews all basic and established knowledge that is needed to understand the concepts presented in this work, such as the data structures mainly related with logic synthesis and the polarity assignment problem. Then, we present some preliminary concepts and discussion, such as the adopted concept of simple and complex cells, a discussion on the ideal composition of standard cell libraries, among others.

Chapter 3: *PREVIOUS WORKS* — reviews some previous and recent works that are connected to the work presented in this thesis.

Chapter 4: *EFFICIENTLY MAPPING VLSI CIRCUITS WITH SIMPLE CELLS*

— starts by providing an overview of the proposed approach, presenting examples and a general point of view in such a way to provide a better understanding of the proposed synthesis flow. Then, it presents a detailed explanation of each step and substep of the proposed approach, as well as presents pseudocodes of the proposed algorithms.

Chapter 5: *DETAILS THAT MATTER* — four secondary contributions are presented, two of them related with synthesis speedup and other two related with improvements to the quality-of-results (QoR).

Chapter 6: *EXPERIMENTAL RESULTS* — presents and discusses the obtained results when running the proposed technique in a set of benchmark circuits, analyzing the influence of each proposed contribution to the final result.

Chapter 7: *CONCLUSION AND FUTURE WORKS* — provides the main conclusions, summarizes the contributions of this work and presents the intended future works.

2 BACKGROUND AND PRELIMINARIES

This chapter summarizes the necessary technical background related to this work. Several topics, including data structures for logic synthesis and the polarity assignment problem are reviewed. In the sequence, we present the adopted definition of simple and complex cells. We also bring up a discussion about the ideal composition of standard cell libraries. The adopted definitions of positive and negative cells are also presented.

2.1 Logic Synthesis Data Structures

In the current VLSI design flow, the logic representing the circuit may be computationally described in a number of different data structures. Each one of them has its particular strengths and weaknesses, being more or less suitable to specific manipulations. In the following subsections, we describe the most relevant data structures related to logic synthesis algorithms.

2.1.1 Binary Decision Diagrams

A *binary decision diagram* (BDD) is a graph-based logic representation in which a set of binary-valued decisions culminate in either a *TRUE (1)* or a *FALSE (0)* state. Although BDDs have been a subject of study since the 50's (LEE, 1959; AKERS, 1978), this data structure just began attracting the research community's attention when Bryant (1986) brought out the advantages of BDDs as canonical representations (HACHTEL; SOMENZI, 1996; MICHELI, 2003).

Formally, a BDD is a rooted, directed acyclic graph (DAG, i.e., a directed graph with no directed cycles) with two terminal nodes, called *1-terminal* and *0-terminal*. These terminal nodes denote the *TRUE* and *FALSE* decisions, respectively. BDDs are comprised of nodes from three subsets: function nodes set Φ, internal nodes set V, and terminal nodes set $\{0, 1\}$:

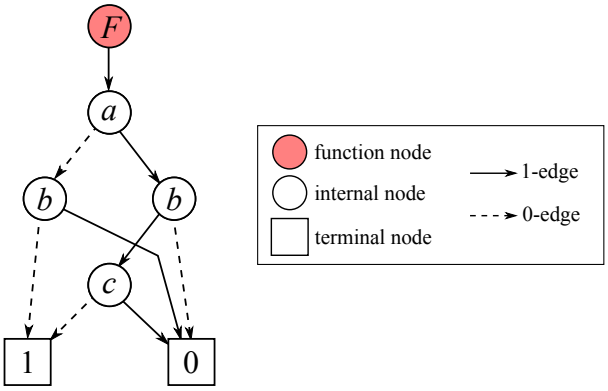
- a function node $\phi \in \Phi$ denotes the function being represented. It has one outgoing edge and have no incoming edges;
- each internal node $v \in V$ has a label $l(v) \in S_F$, where S_F denotes the *support* of

a function F, i.e., each label represents a variable on which F actually depends. The internal nodes have two outgoing edges: the *0-edge*, which denotes the *FALSE* decision with respect to the source node; and the *1-edge*, which denotes the *TRUE* decision with respect the to source node (HACHTEL; SOMENZI, 1996; MICHELI, 2003; GEREZ, 1999);

- a terminal node denotes the overall Boolean state for the function.

Figure 2.1 depicts a BDD representing the function $F = ab\overline{c} + \overline{a}\overline{b}$.

Figure 2.1: BDD representing the function $F = ab\overline{c} + \overline{a}\overline{b}$.



Source: Author.

A serious issue with BDDs is the ordering. To achieve canonicity, the BDD must be both reduced and the variables in its support must be ordered (BRYANT, 1986; HACHTEL; SOMENZI, 1996; MICHELI, 2003; GEREZ, 1999). Unfortunately, the size of the BDD critically depends on the specific ordering chosen. In some cases, this dependence is so drastic that it is impractical to build the BDD at all. It is also known that BDD's size may grow exponentially with relation to the number of variables, independent of the order. Considering $|S_F|$ as the number of variables in the support of the function F, the maximum number of nodes $|V|$ of a BDD is given by Equation (2.1). Thus, the use of BDDs becomes impractical in some cases and limited by the number of variables.

$$|V| = \frac{2^{|S_F|}}{|S_F|} \tag{2.1}$$

2.1.2 Boolean Networks

A *Boolean network* is a graph model to logically represent multi-level circuit structures (BRAYTON et al., 1987). Boolean networks are DAGs comprised of three types of nodes: primary input nodes, which have no incoming edges; primary output nodes, which have no outgoing edges; and intermediate nodes, which represent the internal structure of a circuit. Each of these intermediate nodes is associated with a Boolean function, called local function, that the node implements (HASSOUN; SASAO, 2012).

There is a number of different ways to represent local functions in intermediate nodes, each of which has its own merits and demerits. In MIS, Brayton et al. (1987) use a sum-of-product (SOP) expression for logic manipulation and a factored form for area estimation by literal counting. In the Bold system (BARTLETT et al., 1987), SOPs are used for both logic manipulation and area estimation. Still, these models have a heterogeneous form of representing intermediate nodes. Recently, local functions have been modeled as simple, homogeneous operators, what makes it easier to manipulate. Examples of homogeneous Boolean network representations are and-inverter graphs, xor-and-inverter graphs and majority-inverter graphs, which are presented in details in the following subsections.

2.1.3 And-Inverter Graphs

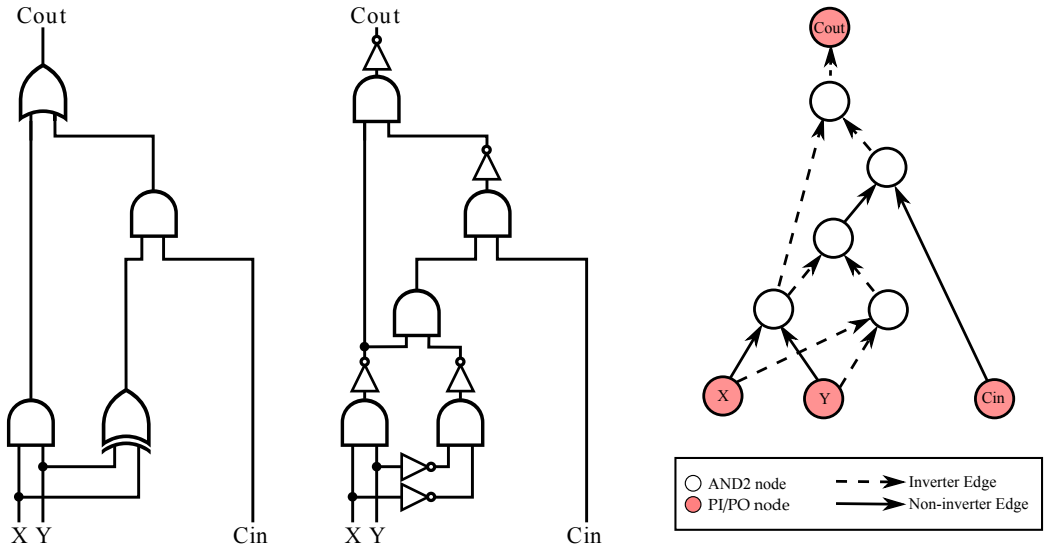
And-inverter graphs (AIGs) are data structures used in a number of state-of-the-art logic synthesis tools, like ABC (Berkeley Logic Synthesis and Verification Group, 2017). Although circuit transformations based on and-inverter representations date back to the 60's, including the works of Hellerman (HELLERMAN, 1963) and Darringer (DARRINGER et al., 1981), this sort of representation reclaimed the community's attention as a data structure for technology-independent circuit representation (i.e., before technology mapping).

Formally, AIGs are Boolean networks in which the intermediate nodes are modeled as two-input AND (AND2) nodes. Thus, AIGs are DAGs with specific types of nodes: 2-input AND (AND2) nodes, primary input (PI) nodes, and primary output (PO) nodes. Primary input nodes have no incoming edges. AND2 nodes have two incoming edges. Any node of an AIG can be labeled as an output node

(MISHCHENKO; CHATTERJEE; BRAYTON, 2006).

The edges of an AIG have a specific property: they are either in their positive or complemented form. A Boolean signal arriving at the target node through a positive edge has the same polarity as the source node. The complemented form of an AIG edge indicates the Boolean inversion operation of its signal (MISHCHENKO; CHATTERJEE; BRAYTON, 2006). Figure 2.2 shows a possible AIG of the logic function $cout = (x \bigotimes y \cdot cin) + (x \cdot y)$.

Figure 2.2: A combinational circuit (a), its representation with AND2 gates and inverters (b), and the derived AIG representation (c).



(a) (b) (c)

Source: Author.

The main advantages of AIGs as data structures for logic synthesis are as follows: (*i*) it is homogeneous, what makes it easier for logic manipulation; (*ii*) AIGs can be easily stored in memory, what makes it scalable even for circuits with tens of millions of nodes; and (*iii*) structurally, the AIG node count has a fair correlation with the circuit area, and the logic depth in AIGs can be correlated with the circuit delay. Based on that, state-of-the-art logic synthesis tools, like ABC (Berkeley Logic Synthesis and Verification Group, 2017), are able to minimize the circuit area, by minimizing the AIG node counting, and the circuit performance, by optimizing the logic depth in critical paths (BRAYTON; MCMULLEN, 1982; CORTADELLA, 2003; MISHCHENKO; CHATTERJEE; BRAYTON, 2006).

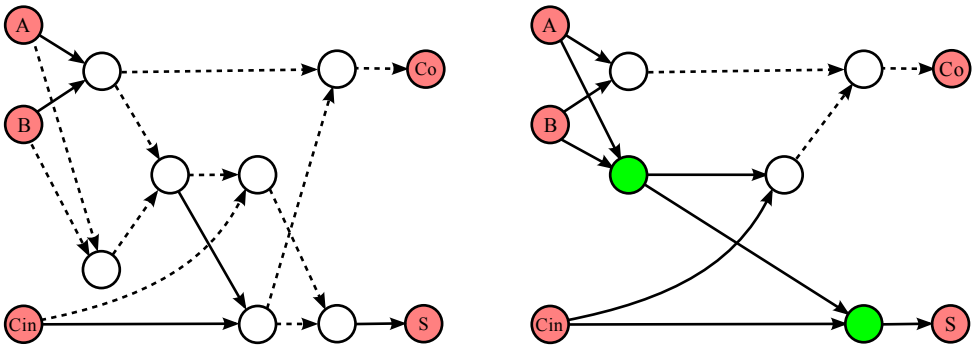
Although most of these methods are able to handle slightly more complex circuit representations, for the sake of simplicity and without limiting the generality, they are presented by taking AIGs as inputs. Still, some optimizations can be

potentially improved if specific types of logic nodes could be added into the AIG representation, or even by replacing the AND2 node for a different logic node. These are the cases for xor-and-inverter graphs and majority-inverter graphs.

2.1.4 Xor-And-Inverter Graphs

An *xor-and-inverter graph* (XAIG) can be roughly defined as an AIG extended to explicitly represent two-input XOR (XOR2) nodes. So, XAIGs are Boolean networks in which the intermediate nodes can be modeled either as AND2 nodes or as XOR2 nodes. Apart from that, XAIGs are just like AIGs, with PI and PO nodes and also with inverters in the edges. Figure 2.3 illustrates an AIG and an XAIG for a full adder.

Figure 2.3: A full adder represented both as an AIG (a) and as an XAIG (b). Green nodes represent XOR2 nodes.



(a) (b)

Source: Author.

The motivations for XAIGs rely on the idea that standard structures (and the algorithms based on them) are mainly AND/OR based and they do not efficiently cope with XOR logic (SCHMIDT; FISER, 2009). This is the case for the work presented herein. We show that it is possible to explore the parity property of XOR functions in order to treat XOR/XNOR cells as polarity don't cares in terms of transistor count. The proposed XOR-based optimizations can further optimize area and power, as we demonstrate in our experimental results.

2.1.5 Majority-Inverter Graphs

A majority-of-three (MAJ3) function evaluates to *TRUE* only when at least two of its inputs are *TRUE*. A(n) *majority-inverter graph* (MIG) is a Boolean network in which the intermediate nodes are modeled as three-input majority nodes, i.e., the nodes implement the MAJ3 function. As in AIGs and XAIGs, the inversions are represented in the edges.

MIGs were firstly introduced as data structures for logic synthesis by Amarú, Gaillardon and Micheli (2014). A number of works have also been proposed based on MIGs (AMARU et al., 2016; SOEKEN et al., 2017; AMARÚ et al., 2015). The authors claim that, by using the proposed Boolean algebra based only on majority and inverter operations, one can natively optimize MIGs. This would extend the capabilities of modern synthesis tools, especially with respect to datapath circuits, as majority functions are the ground for arithmetic operations. Still, the majority gate is a complex cell and, as MIGs are based on majority operations, a detailed study of MIGs goes beyond the scope of this thesis and we refer the reader to the work by Amaru (2017).

2.2 Polarity Assignment Problem

In this section, the polarity assignment problem is defined. Then, the phase-constraint graph and the polarity graph are presented as auxiliary data structures to solve the polarity assignment problem.

2.2.1 Problem Definition

Polarity assignment often refers to the problem of, given a combinational network, (1) assigning the polarity (or phase) of its inputs and outputs in order to yield the signals and their complements; (2) do not modify the network functionality; and (3) minimizing an objective function of interest (MICHELI, 2000).

Jain and Bryant (1993) model a variation of the polarity assignment problem to minimize the inverter counting in multi-level logic networks. Their approach is based on modeling the cells from the library as phase-constraint graphs and on

representing the complete circuit as a polarity graph. We propose to apply a variation of the inverter minimization procedure proposed by Jain and Bryant as a way to efficiently map VLSI circuits using simple cells. In the following, we review the phase-constraint and polarity graphs proposed by Jain and Bryant.

2.2.2 Phase-Constraint Graph

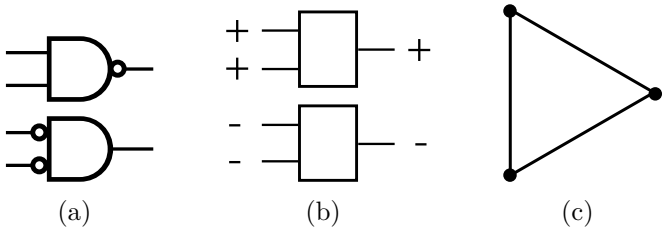
A phase-constraint graph defines phase constraints between the inputs and outputs of cells from the library based on a given base function (e.g. NAND). For simplicity, we will refer to these inputs and outputs as pins. Each node in a phase-constraint graph corresponds to a pin of the base function. The graph will, then, capture how the other cells from the library can be obtained by changing the phase assignments on its pins when compared to the base function. Thus, if a given cell can be obtained by keeping the same phase assignment of the base function in a given pin, then this pin is said to have a positive phase assignment. On the other way around, if a given cell needs to permute the phase assignment of a given pin to implement the base function, then this pin is said to have a negative phase assignment (JAIN; BRYANT, 1993).

Figure 2.4 shows a NAND/NOR phase-constraint graph. In this example, the NAND cell is the base function and so, by definition, its pattern has a positive phase assignment in all pins (JAIN; BRYANT, 1993). The NOR operation can be obtained by permuting all phase assignments from NAND cell ($\overline{A+B} \equiv \overline{A} \cdot \overline{B} \equiv \overline{\overline{A} \cdot \overline{B}}$). This way, the NOR pattern has a negative phase assignment in all pins. In this case, NAND and NOR cells define the phase-constraint set. The resulting phase-constraint graph has three nodes, corresponding to the number of pins in its base function. The edges specify that all three pins should always have the same phase assignment (either all positive or all negative).

2.2.3 Polarity Graph

The polarity graph states how each gate in a given logic network differs from the phase-constraint graph of the corresponding base function. For this, let n_i be the i_{th} net on the logic network, and let g_k be its k_{th} gate. For every input net n_i of a

Figure 2.4: NAND/NOR phase-constraint graph (c), derived from its allowed phase assignment permutations (a) and respective patterns (b).



Source: Author.

gate g_k, $\gamma(g_k, n_i)$ denotes whether the net is in its positive form ($\gamma_1(g_k, n_{i_1}) = +$) or in its complemented form ($\gamma_2(g_k, n_{i_2}) = -$). All output nets n_o are in their positive form with respect to their gate g_k, i.e., $\gamma_3(g_k, n_o) = +$ (JAIN; BRYANT, 1993).

Each vertex v_i in the polarity graph corresponds to a net in the logic network. Each edge e_{ij}, between vertex v_i and vertex v_j, implies that there is a constraint of phase assignment in the phase-constraint graph. All edges have a label $\lambda(e_{ij}) \in \{+, -\}$. A positive edge ($\lambda(e_{1_{ij}}) = +$) specifies that the nodes v_i and v_j are required to have the same phase assignment. A negative edge ($\lambda(e_{2_{il}}) = -$) specifies that the vertices v_i and v_l should have opposite phases (JAIN; BRYANT, 1993).

The edge labels in the polarity graph are obtained by applying the *times operator*, represented by the symbol $\bullet$. The times operator must be applied between the polarity of the nets in the logic network, i.e., over the elements of the set $\{+, -\}$. Let $\gamma(g_k, n_i)$ and $\gamma(g_k, n_j)$ be the polarity of nets n_i and n_j for the gate k. The label of edge e_{ij}, denoted by $\lambda(e_{ij})$, is defined by applying the times operator, such that:

$$\lambda(e_{ij}) = \gamma(g_k, n_i) \bullet \gamma(g_k, n_j) = \begin{cases} +, & \text{if } \gamma(g_k, n_i) \text{ and } \gamma(g_k, n_j) \text{ have the same label} \\ -, & \text{if } \gamma(g_k, n_i) \text{ and } \gamma(g_k, n_j) \text{ have different labels} \end{cases} \tag{2.2}$$

This way, according to Equation 2.2, $\lambda(e_{ij})$ could be:

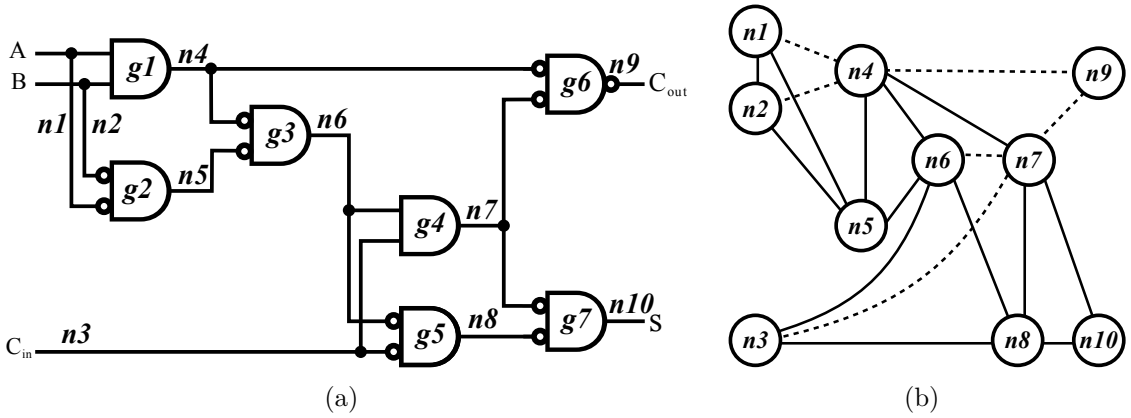
$$(+ \bullet +) = (- \bullet -) = + \tag{2.3}$$

$$(+ \bullet -) = (- \bullet +) = - \tag{2.4}$$

Figure 2.5 shows a logic network and its corresponding polarity graph for the NAND/NOR phase-constraint. In Figure 2.5(b), the positive edge (solid line)

between nodes n_1 and n_2, introduced due to gate g_1, indicates that the vertices should have the same phase in order to avoid adding explicit inverters. In the same way, the negative edge (dotted line) between nodes n_1 and n_4, also introduced due to gate g_1, indicates that the vertices should have opposite phases.

Figure 2.5: Logic network (a) and the corresponding polarity graph (b).



(a) (b)

Source: Author.

2.3 Simple and Complex Cells

Standard cell libraries are commercially available, optimized to achieve high quality results in terms of speed, area and/or power consumption. Such sets of cells usually contain hundreds of elements, ranging from simple to high complexity cells. The set of cells in a standard cell library commonly impacts the quality result of established technology mapping approaches (KEUTZER, 1987; DETJENS et al., 1987; MISHCHENKO; CHATTERJEE; BRAYTON, 2005; BRAYTON et al., 2005; CHATTERJEE et al., 2006). In this work, we are considering simple cells as straightforward implementations from elementary 2-input Boolean functions, as AND, OR, NAND, NOR, XOR and XNOR functions. Figure 2.6 shows an example of simple cell implementations on static complementary CMOS (SCCMOS) logic style. In the same way, we consider complex cells as non-elementary Boolean functions implemented with transistor-level networks using series-parallel (REIS et al., 1995; GAVRILOV et al., 1997) or pass-transistor logic (PTL) topologies (YANBIN; SAPATNEKAR; BAMJI, 2001; SHELAR; SAPATNEKAR, 2005). Figure 2.7 shows an example of complex cell implementations on SCCMOS logic style.

Figure 2.6: Simple cell implementations on SCCMOS logic style: *NOR2* (a) and *NAND2* (b) implementations.

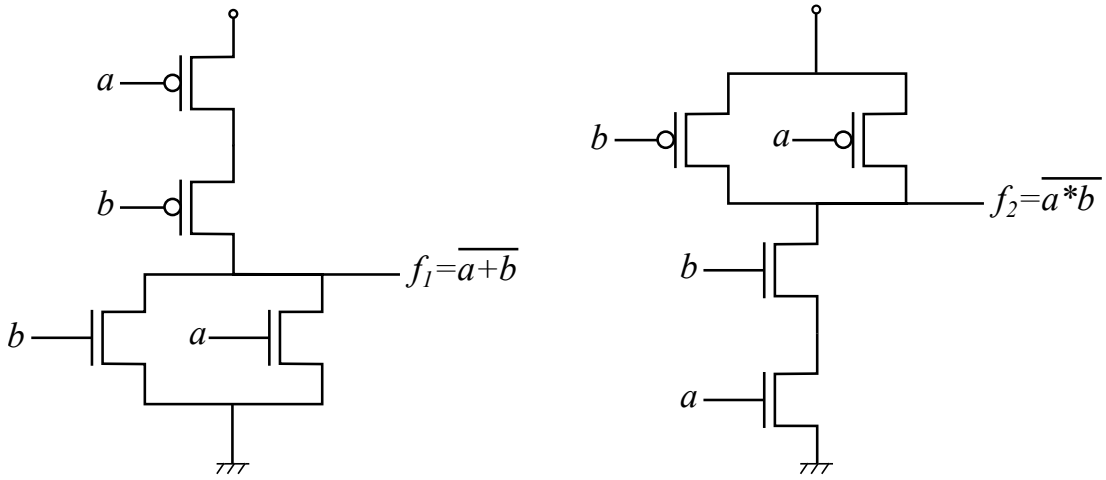
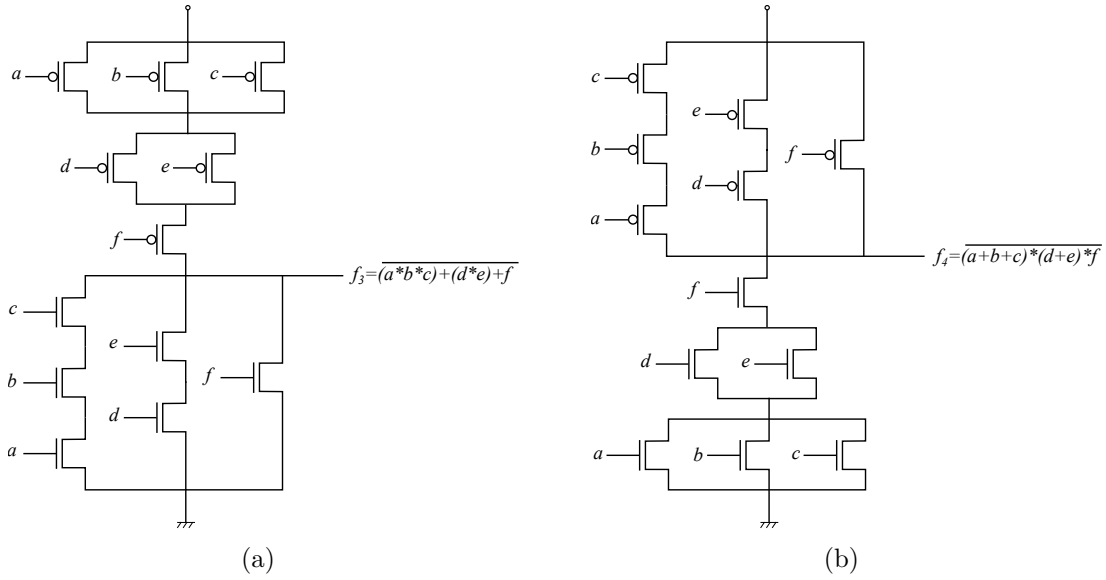


Figure 2.7: Complex cell implementations on SCCMOS logic style: *AOI321* (a) and *OAI321* (b) implementations.



2.4 The Ideal Composition of Standard Cell Libraries

The composition of standard cell libraries may vary according to the designer requirements and application constraints. Improvements to the library side are commonly thought by adding new drive strengths (DOOD; LEE; ALBERS, 2006), new available functions (GAVRILOV et al., 1997) and even particular transistor arrangements (MARQUES et al., 2007a; SCHNEIDER et al., 2005; KAGARIS; HANIOTAKIS, 2007). In this section, the ideal composition of standard cell libraries is discussed in terms of their composition w.r.t. simple and complex cells.

2.4.1 Large Libraries with Complex Cells

Considering the ideal composition of a cell library, some authors (REIS et al., 1995; GAVRILOV et al., 1997; CORREIA; REIS, 2004; MARQUES et al., 2007b; YANBIN; SAPATNEKAR; BAMJI, 2001; SHELAR; SAPATNEKAR, 2005) advocate the use of larger libraries using more complex cells (e.g. AOI and OAI cells). In the particular case of library-free approaches (REIS et al., 1995; MARQUES et al., 2007a), the use of complex cells is explored more extensively. Studies about library-free approaches have tried to demonstrate the advantages of using complex cells when compared to simple cells (REIS et al., 1995; GAVRILOV et al., 1997; YANBIN; SAPATNEKAR; BAMJI, 2001; SHELAR; SAPATNEKAR, 2005). The main advantage reported by those works is the reduction in the number of devices (transistors) needed to implement digital integrated circuits (ICs). Published results using complex cells claim reductions of the order of 40% in terms of transistor count when compared to simple cell implementations.

2.4.2 Small Libraries with Simple Cells

There are other authors (MASGONTY et al., 2001; SEO et al., 2008; RICCI; MUNARI; CIAMPOLINI, 2007) arguing in favor of using small libraries composed of simple cells (e.g. AND, OR, NAND and NOR). In (MASGONTY et al., 2001), the library was previously reduced by the designer, claiming that the logic synthesizers work better with a reduced number of gates. Seo *et al* (SEO et al., 2008) propose to use only *1*- or *2*-input cells, claiming that the use of larger standard cells increases the number of long wires and may undermine circuit delay optimization at $65nm$ and below. Also, Ricci *et al* (RICCI; MUNARI; CIAMPOLINI, 2007) propose to reduce the library iteratively and statistically. They claim that the circuit performance, with respect to full-size library synthesis, do not appreciably degrades and, in several cases, actually improves, whereas the synthesis time decreases and the library maintenance and characterization tasks can be significantly reduced (RICCI; MUNARI; CIAMPOLINI, 2007).

2.5 Positive and Negative Cells

In the context of this work, positive cells implement positive unate functions, such as *AND2* and *OR2*. Conversely, we say that negative cells implement negative unate functions, like *NAND2* and *NOR2*. Due to the nature of static complementary CMOS logic style, negative cells require less transistors to be implemented. For instance, *NAND2* and *NOR2* cells will be composed of a single stage and have four transistors each. Meanwhile, *AND2* and *OR2* cells will have six transistors as they need an implicit internal inverter.

Due to this reason, *NAND2* and *NOR2* cells will be used as base cells in this work. This allows us to minimize the total number of transistors by eliminating implicit inverters inside *AND2* and *OR2* cells while minimizing the number of explicit inverters in between *NAND2* and *NOR2* base cells.

2.6 Circuit Optimization with XAIGs

AIGs have been used as underlying data structures for logic synthesis over the last years. Nonetheless, there are some optimizations that would take benefit of extended versions of AIGs. In this section, we highlight some motivations to use XAIGs instead of AIGs for specific optimizations.

The number of nodes in AIGs can be directly correlated with the number of cells in implementations using 2-input ANDs, ORs, NANDs and NORs. These cells are variants of the primitive *AND2*, obtained by applying De Morgan's law. Notice that De Morgan's law modifies the polarity of the input and output signals. Due to this reason, AIGs with a minimized number of nodes can be used almost straightforwardly to generate minimum transistor count circuits based on using 2-input ANDs, ORs, NANDs and NORs. It is important to remark that *NAND2* and *NOR2* cells can be implemented with four transistors in Static Complementary CMOS (SCCMOS) logic style, while *AND2* and *OR2* cells require six transistors. Therefore, it is possible to choose the polarity of internal nodes, such that *NAND2* and *NOR2* cells are preferred. This way, when using this set of cells, the transistor count is reduced by choosing cells with a low number of transistors, whereas a few inverters are added.

The first motivation to use XAIGs instead of AIGs is that, although one

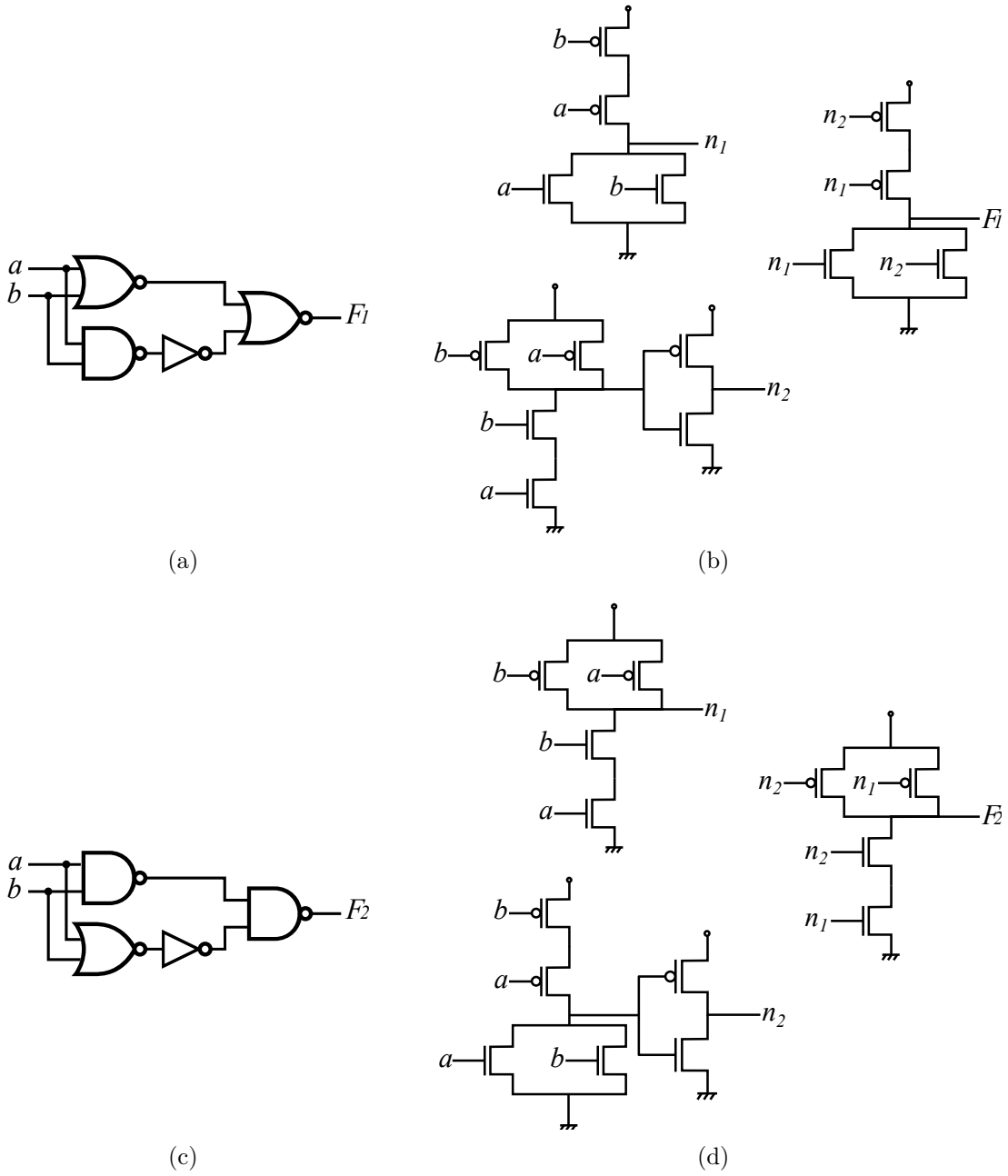
can implement 2-input ANDs, ORs, NANDs and NORs directly from an AIG node, this relation does not hold when considering *XOR2* and *XNOR2* functions. This is because 2-input XOR and XNOR functions need at least 3 AIG nodes to be represented. These three nodes will derive implementations with at least 14 transistors when considering directly corresponding implementation with 2-input ANDs, ORs, NANDs and NORs (see Figure 2.8). Once an XAIG considers *XOR2* nodes, it can be directly correlated with straightforward implementations in SCCMOS logic style of 2-input XOR and XNOR functions with 10 transistors only (see Figure 2.9).

The second motivation to use XAIGs instead of AIGs is concerned to XORs/ XNORs and inversions on their inputs and outputs (I/Os). For any n-input XOR function, any odd number of I/O inversions results in an XNOR function, whereas any even number of I/O inversions does not change the function output. Similarly, for any n-input XNOR function, any odd number of I/O inversions results in an XOR function, whereas any even number of I/O inversions does not change function output.

Once both XOR and XNOR cells can be implemented with 10 transistors, we consider *XOR2* nodes as polarity don't cares in terms of transistor count. Figure 2.10 illustrates all the 8 possible polarity assignments for an *XOR2* cell. Notice that each possibility corresponds to either the XOR2 function itself or the XNOR2 function and both can be implemented with 10 transistors. Similarly, Figure 2.11 illustrates all the 8 possible polarity assignments for an *XNOR2* cell. Notice also that each possibility corresponds to either the XNOR2 function itself or the XNOR2 function and both can be implemented with 10 transistors as well. This characteristic can be used to further optimize the inverter count in a circuit, as we discuss in Section 4.5.

XOR and XNOR functions can also be explored as polarity don't care nodes even if the library used for mapping does not have XOR/XNOR cells. Since each of the functions can be implemented with 14 transistors with NAND2, NOR2 and inverters, any applied optimization for the former case will also hold to the later one.

Figure 2.8: *XOR2* implementation with 14 transistors: cell-based representation (a) and its transistor network (b). *XNOR2* implementation with 14 transistors: cell-based representation (c) and its transistor network (d).



2.7 Inverter Count vs Transistor Count Minimization

One of the works that bases this thesis rely on an algorithm to minimize the inverter count in a circuit. Even so, in this thesis, we go beyond inverter count minimization and one of the adopted metrics is the transistor count. In this section, we explicitly differ transistor count from inverter count minimization.

Inverter count minimization does not suffice to minimize transistor count.

Figure 2.9: *XOR2* implementation with 10 transistors: cell-based representation (a) and its transistor network (b). *XNOR2* implementation with 10 transistors: cell-based representation (c) and its transistor network (d).

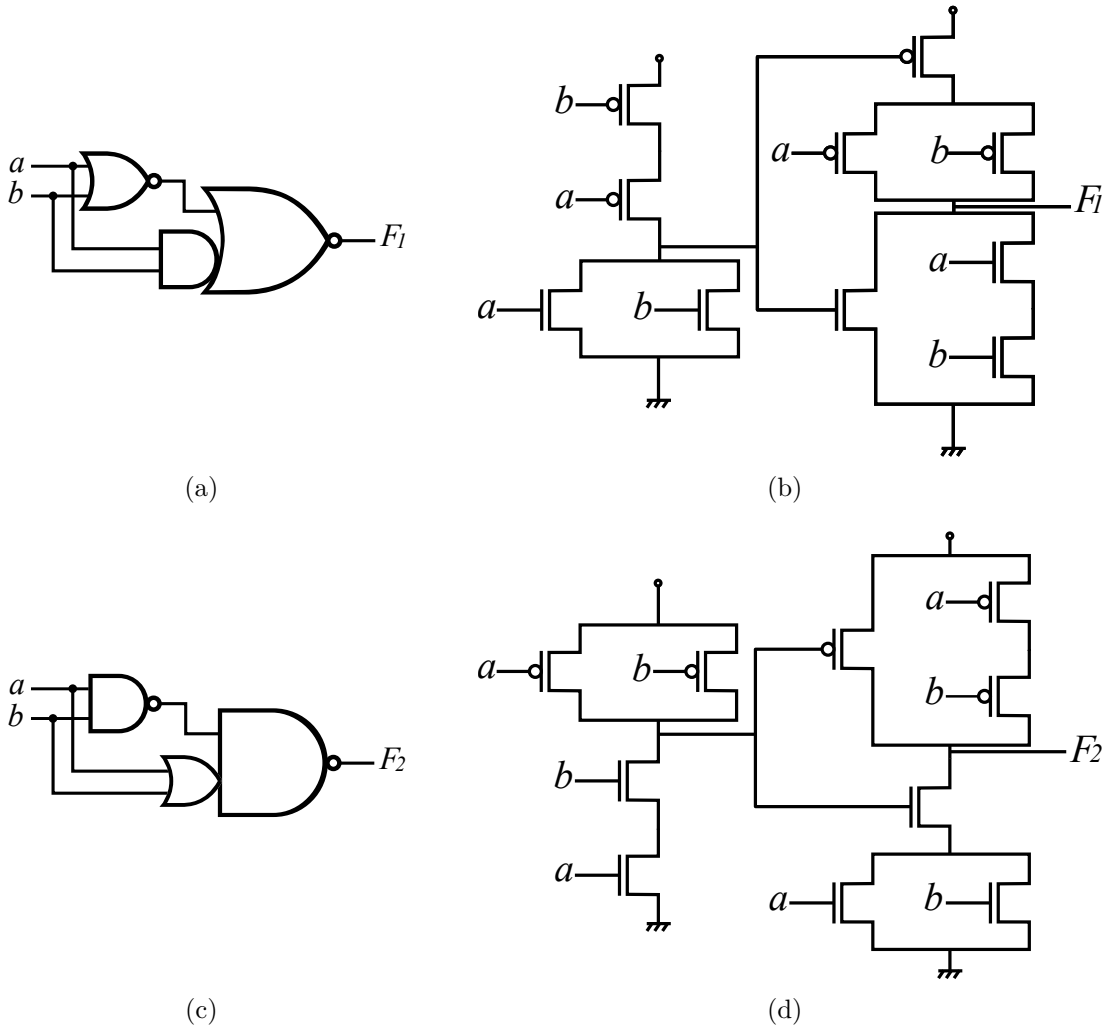
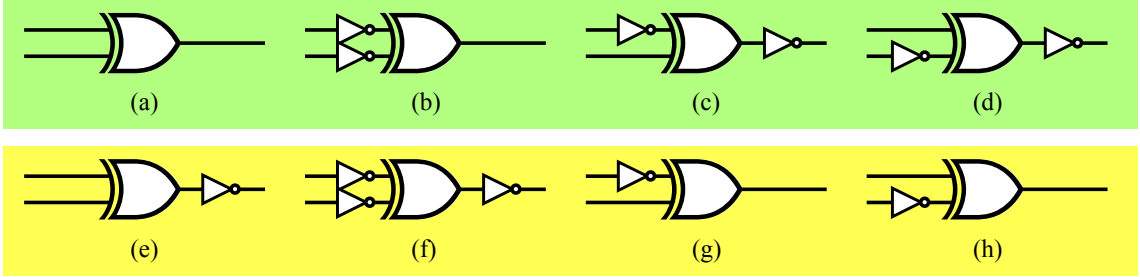
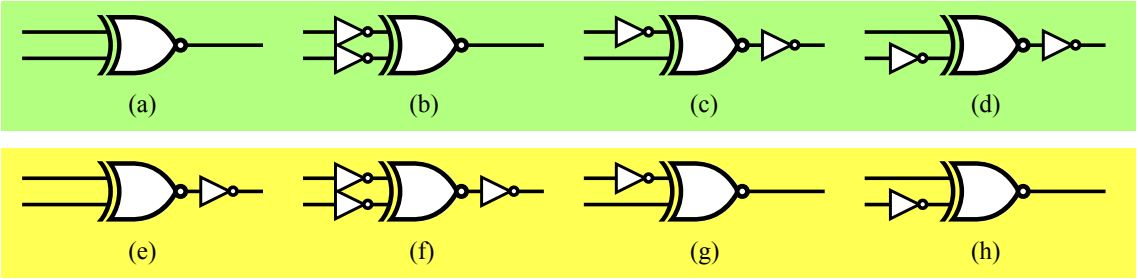


Figure 2.10: The 8 possible polarity assignments for an *XOR2* cell. The networks in green correspond to an *XOR2* function and each one of them can be implemented with 10 transistors only (see Figure 2.9(b)). The networks in yellow correspond to an *XNOR2* function and each one of them can also be implemented with 10 transistors only (see Figure 2.9(d)).



A first important difference is that, in order to reduce the number of transistors, negative cells should be used to avoid implicit inverters required by positive cells.

Figure 2.11: The 8 possible polarity assignments for an *XNOR2* cell. The networks in green correspond to an *XNOR2* function and each one of them can be implemented with 10 transistors only (see Figure 2.9(d)). The networks in yellow correspond to an *XOR2* function and each one of them can also be implemented with 10 transistors only (see Figure 2.9(b)).



One can perform high-effort inverter count minimization techniques, but if only positive cells are used, the implicit inverter in the positive cells can undermine transistor count even with very good results of (explicit) inverter count minimization. For this reason, our approach uses *NAND2* and *NOR2* as base cells. This is a simple but very meaningful change on the way from inverter count minimization to transistor count minimization.

Another important difference is the minimization on the number of logic cells. Minimizing the number of inverters does not necessarily lead to a minimized transistor count if the number of logic cells is not minimized along in the process. For this reason, our approach rely on minimizing the number of logic cells before minimizing the number of inverters, so that, at the end the process, both the cost functions are minimized.

3 PREVIOUS WORKS

This chapter reviews previous works that are connected to this thesis. First, approaches for AIG node count minimization are reviewed. Then, a historic perspective on technology mapping procedures is presented, briefly describing library-free- and standard-cell-based approaches. Finally, we review with more details two algorithms for both inverter and transistor count minimization based on polarity assignment.

3.1 AIG Node Count Minimization

In addition to homogeneity and simplicity, AIGs are good circuit representations in structural terms. Optimizations on logic depth in critical paths tend to optimize the circuit performance. Also, minimizing the AIG node counting tends to minimize the circuit area. In this section, the main approaches for AIG node count minimization are reviewed.

In 1982, Brayton and McMullen proposed an algorithm for decomposition and factorization in Boolean expressions (BRAYTON; MCMULLEN, 1982), which was implemented in SIS (SENTOVICH et al., 1992). The same algorithm has also been proposed as an AIG rewriting method for node count minimization. This method, called *refactor*, chooses large subgraphs for each AIG node, extracts the Boolean function of this subgraph and performs Brayton and McMullen's factorization. The result of factorization is converted back to an AIG and replaces the original subgraph if the number of nodes is reduced (Berkeley Logic Synthesis and Verification Group, 2017).

Cortadella (2003) proposed an algebraic balancing approach in DAG structures claiming reductions on logic depth and timing optimizations. This algorithm was adapted to AIGs and it is also implemented in ABC tool, called *balance* (Berkeley Logic Synthesis and Verification Group, 2017). The approach is based on finding the minimum-depth tree for a Boolean function with the usage of bi-decomposition techniques by building the tree from root to leaves. The depth reduction is achieved by means of rewriting rules that apply the associative, commutative and distributive laws of the Boolean algebra. This method is frequently used in between AIG node count minimization approaches to recover logic depth.

Mishchenko, Chatterjee and Brayton (2006) proposed a DAG-aware AIG rewriting approach called *rewrite*. This three-steps method uses a hash table of pre-computed AIGs for all NPN-equivalent classes of functions with up to four inputs. In the first step, given the AIG to be optimized, the method traverses the graph in topological order (from inputs to outputs) and, for each node, it exhaustively extracts all possible subgraphs with up to four inputs. Each of these subgraphs is compared to the pre-computed AIGs in the hash table. The one that leads to the greatest improvement replaces the original subgraph. The second and third steps consist on balancing and refactoring the AIG using the methods *balance* and *refactor*, respectively.

It is worth to mention that most of the methods based on AIGs is highly dependent on the circuit given as inputs. Thus, the quality of results can be even more improved if the applied methods are repeatedly iterated and interleaved with each other. As an example, Mishchenko, Chatterjee and Brayton (2006) suggest a script called *resyn2* that iterates 10 times over the AIG by interleaving the refactor, balance and rewrite methods, as follows: *b, rw, rf, b, rw, rwz, b, rfz, rwz, b*. In the abbreviated forms, *b* stands for balancing; *rw/rf* stand for AIG rewriting and refactoring; and *rwz/rfz* are also rewriting and refactoring, but allowing replacements with zero improvement. The authors claim that this approach leads to a reduction of area in the order of 10% and improvements in delay of 5%, whereas the runtime is reduced by a factor ranging between 7 and ~1000, when comparing with previous approaches.

3.2 Library-Free Technology Mapping

The first library-free technology mapping approach was proposed by Berkelaar and Jess (1988). Expressions of sums-of-products and product-of-sums with a prefixed notation are represented as graphs, which are traversed in reverse topological order (from outputs to inputs) and partitioned into logic cells. As the cuts are made top-down, the logic depth of what is below is unknown. So, this greedy algorithm cannot guarantee a solution with minimum logic depth or with minimum number of cells.

In 1992, two important works were presented. Liem and Lefebvre (1992) proposed a constructive matching, in which both the number of inputs and the

logic depth of a cell are considered. However, the method is hardly dependent on the initial structure and it is memory expensive, what makes it not scalable for mapping large circuits. The work of Abouzeid et al. (1992) claimed to increase the number of logic cells used by partitioning the initial DAG into n-ary trees. This representation decreases the dependence on the initial graph, allowing change of structure in a given set of nodes. Although the cuts are generated from inputs to outputs, they are made in a greedy way, disregarding logic depth minimization.

Reis, Robert and Reis (1998) presents some topological parameters for library free technology mapping (REIS; ROBERT; REIS, 1998). Reis et al. (1995) and Reis (1999) propose to use dynamic reordering on the initial circuit representation. Also, each logic cone is represented in a different way: a special type of BDD, called terminal-suppressed BDD (TSBDD). An interesting property of this structure is the direct association of BDD arcs and transistors, although it faces the same problems of representation by trees.

Later on, Yanbin, Sapatnekar and Bamji (2001) propose the odd-level transistor replacement method, which works directly on an electrical diagram at transistor level, represented as a graph. The goal of the algorithm is to select which gates can be collapsed in order to achieve better performance, but the method also suffers for depending strongly on the initial circuit decomposition.

In 2004, an algorithm proposed by Correia and Reis (2004) dynamically considers several decompositions of sub-trees (at a low computational cost), leading to a minimum covering using dynamic programming. However, as it is based on trees, the method cannot provide a broader view of the circuit. The VIRMA algorithm was presented by Marques et al. (2007a), which performs the library-free mapping over a DAG aiming the reduction of the circuit delay (SCHNEIDER et al., 2005).

3.3 Cell-Based Technology Mapping

The first approaches for automatic synthesis of digital circuits were based on applying a set of rules on the circuit representation (DARRINGER et al., 1981; GREGORY et al., 1986). These methods perform local optimizations, trying to reduce the costs of a region of the circuit, which not necessarily lead towards global minimals.

The first algorithmic solutions for technology mapping were proposed only

in 1987 (KEUTZER, 1987; DETJENS et al., 1987). Keutzer presented DAGON as a compiler-based approach. He proposes that the pattern matching between cells of a library and subgraphs of a circuit representation is similar to the problem of searching patterns between intermediate representations of a computer program and a given set of machine instructions. However, the search space to be explored by mapping becomes limited by the structural matching and the initial representation of the circuit, which directly affects the quality of the mapped circuit. Additionally, DAGON approach requires all isomorphic matches to be stored in each node of the tree until the end of covering step. Thus, this work precludes the use of large libraries, when the number of patterns found is usually higher.

Detjens et al. (1987) propose to use trees as the subject graph and to insert pairs of inverters in these trees. Although this algorithm increases the solution space, it relies on performing several decompositions for each element of the library, what also turns it unfeasible for large libraries. In 1989, Rudell extends Detjens approach, in which he proposes two main different improvements: (1) to use pattern graphs with leaf-DAG nodes, what makes it possible to match non-tree gates (such as multiplexers and XORs); and (2) to replace every wire in the subject graph by a pair of inverters in series in order to enlarge the set of matches. Rudell's algorithm became known as "the inverter-pair heuristic".

The idea of Boolean matching was introduced only in 1993 by using BDDs (MAILHOT; MICHELI, 1993). In this approach, finding matches does not depend on the structure of the sub-trees anymore. However, this algorithm was computationally expensive. In 1995, Lehman proposes to integrate decomposition and pattern matching by dynamically reorganizing the subject graph. Thus, the search space increases, since each node is associated with functionally equivalent subgraphs. Even so, this approach becomes unpractical for large circuits because the graph grows very fast. Kukimoto, Brayton and Sawkar (1998) and Stok, Iyer and Sullivan (1999) propose two approaches for DAG covering that guarantees optimality in terms of delay. The main drawback of these works is that their delay model ignores the load of the cell, taking into account only its propagation delay.

Recently, Chatterjee et al. (2006) propose an algorithm based on Kukimoto's approach, with two main differences. Instead of structural matching, a Boolean matching extended from Lehman's approach is applied. Also, Chatterjee proposes to encode multiple DAGs (without breaking them in trees) into choice nodes, deriving

the idea of "AIG with choices". This postpones detailed comparisons, what makes his algorithm faster than previous approaches. In this sense, the AIG with choices also prevents memory overload. He also applies a technique similar to Rudells's inverter-pair approach to consider matches in both polarities.

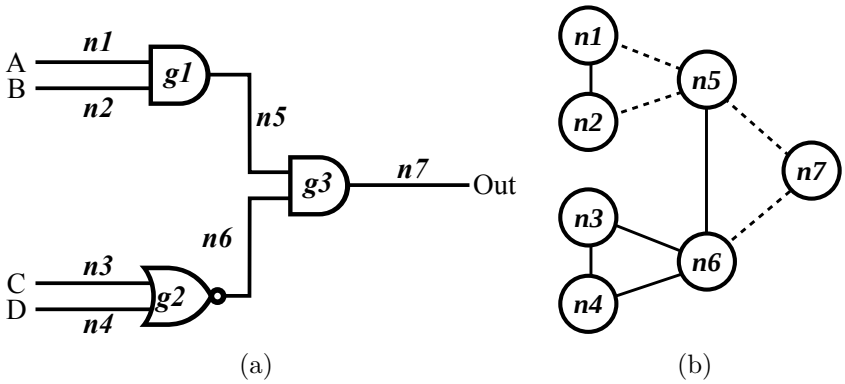
Martins et al. (2010) present a work for Boolean factoring oriented to multi-objective goals (MARTINS et al., 2010). Correia and Reis (2001) propose a method to classify n-input Boolean function (CORREIA; REIS, 2001). Togni et al. (2002) present a method to automatically build standard cell libraries (TOGNI et al., 2002). Poli et al. (2003) unified the theory to create cell-based transistor networks out of BDDs (POLI et al., 2003). Silva, Reis and Ribas (2009) present a study on the CMOS logic gate performance variability related to transistor network arrangements (SILVA; REIS; RIBAS, 2009). Junior et al. (2006) introduce fast disjoint transistor networks from BDDs (JUNIOR et al., 2006). Rosa et al. (2009) present a new method for switch-based optimizations (ROSA et al., 2009). Junior et al. (2007) bring a comparative study of logic gates with minimum transistor stacks (JUNIOR et al., 2007). Junior et al. (2008) introduces methods to automatically generate and evaluate transistor networks based on different logic styles (JUNIOR et al., 2008). Martins et al. present the functional composition as a new paradigm for performing logic synthesis (MARTINS; RIBAS; REIS, 2012; MARTINS et al., 2012). Reis and Anderson (2011) present a method to automatically generate a cell library emphasized on the cell sizes and variants (REIS; ANDERSON, 2011). Martins et al. (2011) present a method to efficiently compute the minimum decision chains of Boolean functions (MARTINS et al., 2011). Butzen et al. (2012) present a method to design CMOS logic gates with enhanced robustness against aging degradation (BUTZEN et al., 2012). Possani et al. present a methodology to find and build non-series-parallel transistor arrangements (POSSANI et al., 2012; POSSANI et al., 2013). Possani et al. (2016) present a graph-based algorithm for transistor network generation addressing supergate design (POSSANI et al., 2016).

3.4 Inverter Count Minimization by Polarity Assignment

Jain and Bryant (1993) propose a method to perform inverter minimization in multi-level combinational circuits. The method is formulated as a polarity assignment problem performed on a polarity graph. An input combinational circuit

(e.g. Figure 3.1(a)) is used to generate the corresponding polarity graph (e.g. Figure 3.1(b)) by defining a set of allowed base functions (such as a library). In the example presented in Figure 3.1, the base functions are *NAND2* and *NOR2*, due to the reasons described in Section 2.5. The polarity graph is created such that it represents how each gate in the initial combinational circuit differs from the corresponding base functions in terms of phase assignments.

Figure 3.1: A combinational circuit (a) and the corresponding polarity graph (b).



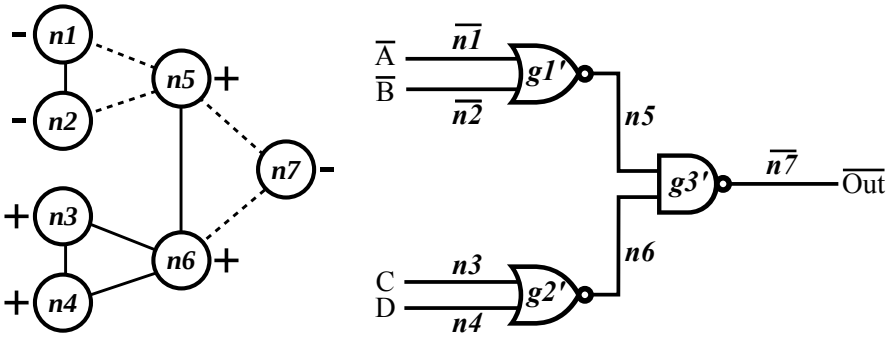
Consider the circuit shown in Figure 3.1(a). This circuit is composed of three logic cells: two *AND2* and one *NOR2* cells. This way, the circuit has two implicit inverters (due to the two AND2 cells). Figure 3.1(b) shows the derived polarity graph when considering *NAND2* and *NOR2* as base functions. The polarity graph has one node for each net in the initial combinational circuit. There are two types of edges in the polarity graph: (1) nodes connected through positive edges are required to have the same phase assignment (polarity); (2) nodes connected through negative edges should have opposite phases. In Figure 3.1(b), the positive edge (solid line) between nodes n_1 and n_2, introduced due to gate g_1, indicates that the inputs of gate g_1 are in the same phase assignment as the *NAND2* base function. However, the nodes n_1 and n_2 are both connected with negative edges (dotted lines) to node n5, indicating that the output of *AND2* gate g_1 is in a different input-output phase assignment compared to the *NAND2* base function. Thus, the polarity graph indicates that the polarity of g_1's output (net n_5) should be different from its inputs (nets n_1 and n_2). The gate g_3 induces a similar situation in the polarity graph for nodes n_5, n_6 and n_7. The gate g_2 is a *NOR2*, which is also a base function, so the relative polarities of inputs and outputs have to be kept the same. This way, nodes n_3, n_4 and n_6 are connected with solid edges, to indicate that the relative polarities need to be conserved.

Once the polarity graph has been derived from a given circuit, the problem of minimizing the number of inverters can be reduced to a problem which is a variation of the graph coloring problem (JAIN; BRYANT, 1993). In graph theory, the most common graph coloring problem would be that of coloring the vertices of a graph such that no two adjacent vertices have the same color, which is a decision problem. In order to model it to miminize the inverter count, a different graph coloring procedure should be applied Jain and Bryant (1993).

To the context of this work, the graphs should be colored to satisfy three restrictions: (1) only two colors can be used to color the graph; (2) all nodes connected through a positive edge must have the same color; and (3) all nodes connected through a negative edge must have a different color. In this sense, a polarity graph cannot be fully colored while there are cycles comprised by an odd number of negative edges (referred as odd cycles hereafter). In order to finish the coloring, it is necessary to break odd cycles by removing a node during the graph coloring procedure. Since each removed node will derive an inverter in the optimized circuit, minimizing the number of inverters is equivalent to find a minimum transversal of all odd cycles. This problem is NP-hard, since it is a special case of the minimum odd cycle transversal problem (LEWIS; YANNAKAKIS, 1980).

The polarity graph from Figure 3.1(b) can be colored with no conflicts in two distinct ways, as shown in Figure 3.2(a) and Figure 3.3(a). Allowed colors belong to the set $\{+, -\}$. The color $+$ indicates that the node will be implemented with the same polarity as the net in the original circuit, while the color $-$ indicates that the signal will be complemented with respect to the original circuit. This behavior can be observed in the colored polarity graphs and the derived circuits shown in Figure 3.2 and Figure 3.3.

Figure 3.2: A first possible coloring (a) for the polarity graph from Figure 3.1(b) and the corresponding circuit (b).

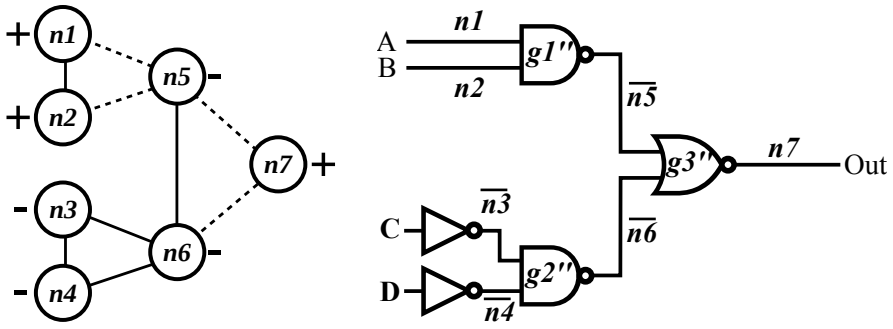


(a) (b)

Algorithm 3.1: QuickColor heuristic (JAIN; BRYANT, 1993).

```
1 quickColor(polGraph)
2   repeat
3     try to color polGraph;
4     if (there are still odd cycles)then
5       oddCycle = any arbitrary odd cycle in polGraph;
6       nodetoBeRemoved = maximum double-edge degree node in
          oddCycle;
7       remove nodeToBeRemoved from polGraph;
8   until (there are no odd cycles);
9   return colored polGraph;
```

Figure 3.3: A second possible coloring (a) for the polarity graph from Figure 3.1(b) and the corresponding circuit (b).



Jain and Bryant (JAIN; BRYANT, 1993) propose two heuristics to search for the minimum transversal of all odd cycles: the *QuickColor* and *GoodColor* heuristics. Both of them are applied in the work proposed herein. The *QuickColor* heuristic picks an arbitrary odd cycle from the graph and then selects the node in this cycle with the maximum double-edge degree, i.e., the node with the highest number of two edges (one positive and one negative) between the same neighbor. If two vertices have the same double-edge degree, then *QuickColor* shall select the one with the maximum edge degree. After removing the selected vertex and incident edges, try recoloring the remaining graph. Algorithm 3.1 presents a pseudocode of *QuickColor* approach.

Different from *QuickColor*, which takes an arbitrary odd cycle, the *GoodColor* heuristic tries to pick a "good" odd cycle. *GoodColor* picks the smallest cycle in the graph as the next candidate. Another remarkable feature is that the *GoodColor* tries to recover itself from an early "potentially bad" choice. According to Jain and Bryant, each eliminated odd cycle in the graph is said to be covered by the node

Algorithm 3.2: GoodColor heuristic (JAIN; BRYANT, 1993).

```
1  goodColor(polGraph)
2      repeat
3          try to color polGraph;
4          if (there are still odd cycles)then
5              oddCycle = smallest odd cycle in polGraph;
6              nodetoBeRemoved = maximum double-edge degree node in
                oddCycle;
               // search for covered vertices
7              foreach (cycleNode ∈ oddCycle)do
8                  if (ℵ(cycleNode) > 0)then
9                      coveringRoot = the root node covering the cycle;
10                     if (ℵ(coveringRoot)==1)then
11                         undo removing of coveringRoot from polGraph;
12                         nodetoBeRemoved = cycleNode;
13                         break;
               // covering the cycle
14             foreach (cycleNode ∈ oddCycle)do
15                 ℵ(cycleNode)++;
16             remove nodeToBeRemoved from polGraph;
17     until (there are no odd cycles);
18     return colored polGraph;
```

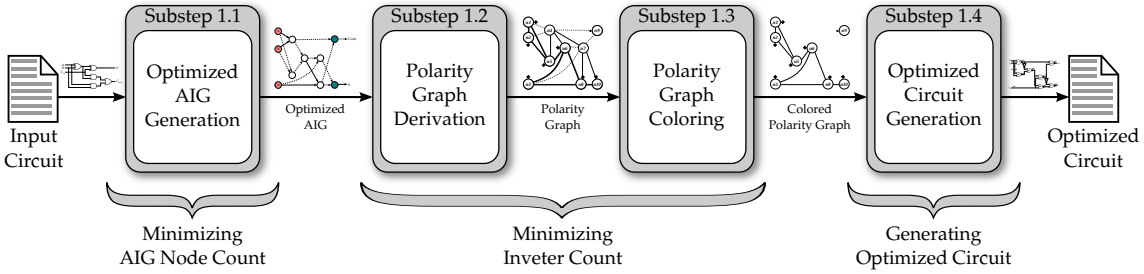
chosen to be removed. This way, it is possible to know when the next odd cycle shares vertices with a previously removed cycle. In order to explain this approach, let $cycle_n$ be the next odd cycle; $cycle_p$, a previously removed cycle, which is covered by vertex v_p and shares vertices with $cycle_n$; and $\aleph(v_p)$, be the number of removed cycles containing the vertex v_p. If $\aleph(v_p) = 1$, the *GoodColor* heuristic "undo" the removing of vertex v_p and removes one of the shared vertices covering both cycles, $cycle_n$ and $cycle_p$. Algorithm 3.2 presents a pseudocode of *GoodColor* approach.

3.5 Transistor Count Minimization by Polarity Assignment

The work presented in this thesis extends the work by Matos (2014). This previous work improves Jain and Bryant's inverter count minimization approach to address the transistor count minimization problem. Its synthesis flow is depicted in Figure 3.4.

The synthesis process starts by optimizing the AIG node count, then mini-

Figure 3.4: The adopted synthesis flow for obtaining reduced transistor count circuits mapped using simple cells proposed by Matos (2014).



Source: Author.

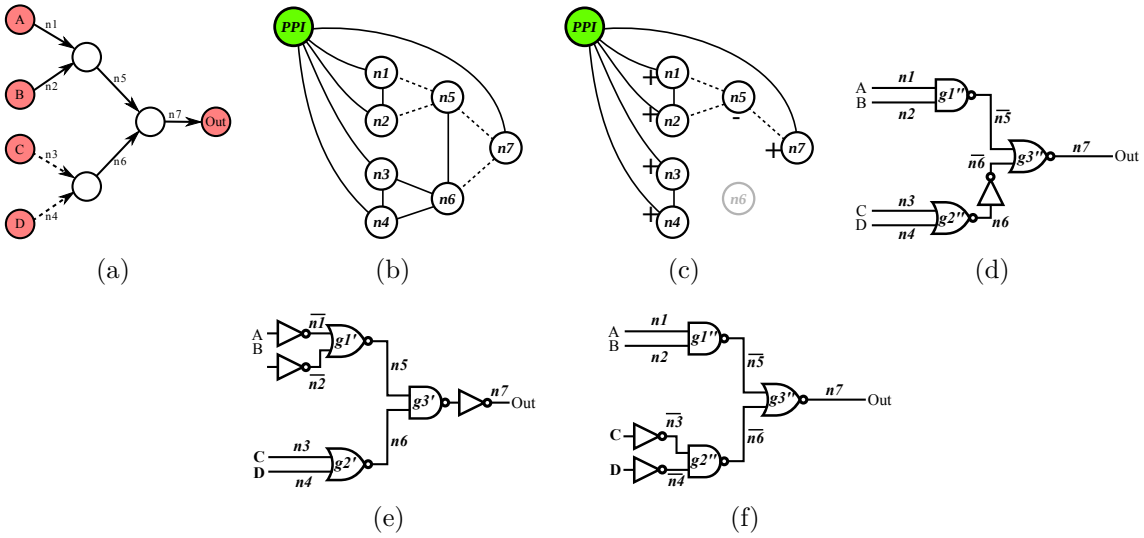
mizes the number of inverters with polarity graphs, and finally generates the final circuit. Some major differences from Matos' work to Jain and Bryant's approach are: (1) the usage of a positive polarity induction (PPI) node to force colors in the polarity graph; (2) a precomputation step to remove nodes with unfeasible fanout; and (3) a trade-off optimization flow to look for a better quality-of-results (QoR).

In order to avoid unnecessary inverters at the primary input and outputs, these nodes must have their polarity controlled in advance. For this reason, Positive Polarity Inducing (PPI) node can be used in the polarity graph. All nodes which are connected with the PPI node through a positive edge (solid line) must have a positive color. In the same way, all nodes that are connected to the PPI node through a negative edge (dashed line) must have a negative color. In this sense, the colors from the PPI node should be propagated before starting the graph coloring procedure. Figure 3.5 highlights the usage of PPI nodes. Notice that the obtained circuit (Figure 3.5(d)) has only one inverter, whereas the best possible solutions from Jain and Bryant's approach have at least two inverters (Figures 3.5(e) and 3.5(f)).

Some nodes may need to be available in both polarities. This can be due to different reasons. For instance, a given primary input signal can be available in both polarities because the input has an external inverter (MACHADO et al., 2012). Similarly, high fanout nodes will require inverter tree insertion to limit fanout, so the signal will be available in both polarities anyway, after inverters are inserted. In this case, high fanout nodes can be considered as being available in both polarities and it may be pointless to color these nodes with a single color to minimize inverter count. This way, high fanout nodes can be removed from the polarity graph.

Finally, Matos (2014) adopted a conservative technique for using the approaches of forcing colors and removing polarity don't care nodes. There are no

Figure 3.5: Graph coloring process for inverter minimization using the PPI node: the initial AIG (a); the derived polarity graph (b); the colored polarity graph (c) and the resulting circuit (d). The best circuit implementation by applying Jain and Bryant's approach are (e) and (f).



guarantees that neither forcing colors using PPI nodes nor removing polarity don't care nodes will lead to better results. Actually, forcing wrong colors using PPI nodes or removing wrong nodes could lead to worse results than if these approaches were not applied. Additionally, it is not possible to predict which is the best order for coloring the polarity graph.

The author proposes a technique based on brute force. All possible parameters are computed independently. After that, a trade-off analysis verifies the best solution. The proposed parameters are five, as following: (1) choose between *QuickColor* and *GoodColor*; (2) use a BFS or a DFS when coloring the graph; (3) force (or not) colors on inputs; (4) force (or not) colors on outputs; and (5) remove (or not) unfeasible fanout nodes. Thus, to verify these five binary parameters, 32 iterations (2^5) are necessary. The final solution is the best among all the 32 generated circuits.

Still, the work by Matos (2014) suffers from a few bottlenecks:

- it is not able to directly synthesize sequential circuits;
- even if combinational clouds are extracted from sequential circuits, the work does not perform any kind of optimization based on sequential elements;
- the methodology is limited w.r.t. AIG node count minimization;
- the author do not perform XOR/XNOR optimizations and the QoR is hardly affected when circuits are mainly implemented with XORs and XNORs;
- it only scratches the surface of polarity don't care optimizations, with unfea-

sible fanout nodes; and

- its fanout limitation algorithm is not level-aware.

In the work presented in this thesis, all the aforementioned shortcomings are addressed. Details on the proposed methodology are presented in the following chapters.

4 EFFICIENTLY MAPPING VLSI CIRCUITS WITH SIMPLE CELLS

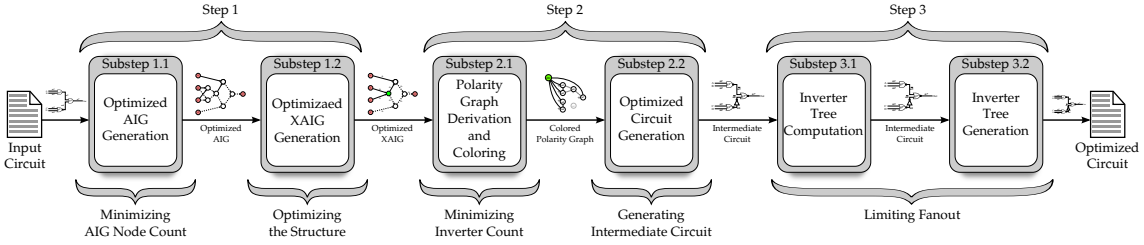
In this chapter, we present the proposed approach to efficiently map VLSI circuits with simple cells. For this, we first introduce the proposed synthesis flow by presenting a methodology overview. Then, each step in the flow is presented in details on the following sections.

4.1 Methodology Overview

This section overviews the proposed synthesis flow to efficiently map VLSI circuits with simple cells. We depict the general point of view, locating each of the steps and substeps in the proposed flow. Also, we describe the inputs and outputs of the substeps and we present some examples for a better understanding of the work presented herein.

The proposed methodology is depicted in Figure 4.1, and it is divided into three main steps. Step 1 starts by generating an optimized AIG in terms of number of nodes (Substep 1.1). By minimizing the AIG node count, we are also minimizing the number of logic cells in the final circuit. Then, the optimized AIG is used to generate an optimized XAIG (Substep 1.2). We propose to rewrite the AIG by replacing XOR/XNOR patterns with XOR nodes.

Figure 4.1: Proposed flow for obtaining efficient circuits mapped using simple cells.

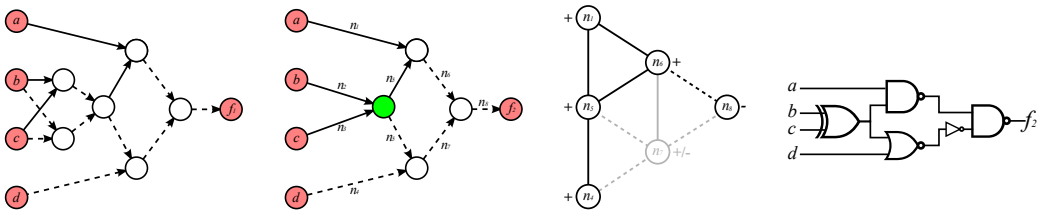


Step 2 aims to provide a circuit with a reduced number of simple cells and a minimized number of inverters in between them, which is directly correlated with minimizing transistor count. To minimize the inverter count, our approach applies a polarity assignment algorithm enhanced from Jain and Bryant's method. The optimized XAIG obtained from Step 1 is used to generate a polarity graph, which is then colored (Substep 2.1). The optimized circuit is obtained from the colored polarity graph (Substep 2.2).

Figure 4.2 presents an example of applying Step 1 and Step 2. The initial

circuit is represented as an AIG in Figure 4.2(a). The optimized XAIG is presented in Figure 4.2(b) (green nodes are XOR nodes). Figure 4.2(c) shows the colored polarity graph, which is used to generate the intermediate circuit presented in Figure 4.2(d).

Figure 4.2: Graph coloring process for transistor count minimization on a given AIG (a). The derived XAIG (b) and the colored version of its polarity graph after removing node $n7$ (c) generates the resulting cell representation (d).



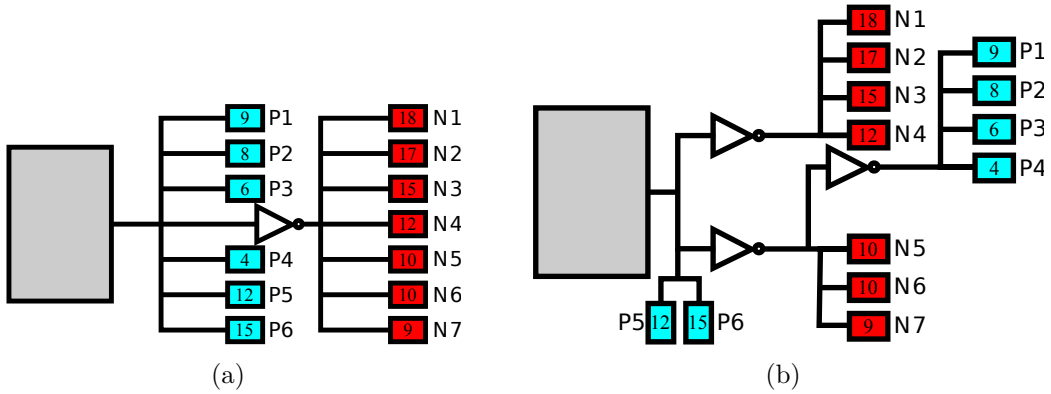
(a) (b) (c) (d)

After generating a minimal simple-cell-count implementation, the obtained circuit might have cells with fanout larger than desired. The third step relies on looking for fanout violations and fixing all of them. For that, we propose an area-oriented, level-aware buffering algorithm, extended from (MATOS et al., 2014). This algorithm takes into account the maximum fanout of the source cell, the maximum fanout of inverters, the number of positive consumer cells (i.e., the output cells driven directly from the source cell), and the number of negative consumer cells (i.e., the output cells driven from the source cell through an inverter). Using a mathematical formulation, to be described further, the method calculates the minimal number of inverters to drive both the positive and negative cells, respecting the given maximum fanout limits. Figure 4.3 shows an example of applying the algorithm in a cell whose maximum fanout limit was not respected. In the shown example, consider both the maximum fanout of the cell and the maximum fanout of inverters as 4.

4.2 Improvements in AIG Node Count Minimization

The QoR of AIGs out of state-of-the-art algorithms for node count minimization is highly dependent on the AIGs used as input. Usually, the results can be even more improved if the applied methods are repeatedly iterated and/or interleaved with each other. Thus, for Step 1.1 of the presented flow, we propose five different AIG node count minimization techniques to explore the optimization efforts up to saturation, i.e, when no more gain is obtained after 10 executions in sequence.

Figure 4.3: Example of fanout violation (a) and fanout limiting using an inverter tree (b).



First, we propose to incrementally iterate ABC's *resyn2* script up to saturation. *resyn2* is not entirely focused on AIG node count minimization, since it starts by balancing the structure to reduce delay upfront as much as possible and, then, minimizes node count under delay constraints. Still, exploring this script up to saturation can deliver very good results in terms of node count.

Another proposed technique is to incrementally iterate ABC's *dc2* script up to saturation. *dc2* is a version of *resyn2*, but now entirely focused on node count minimization. For this, it starts by balancing the structure, but without delay optimization, and, then, minimizes node count without delay constraints.

From these two initial approaches, we derived three others: (*i*) one is called *seq1*, in which we run *resyn2* up to saturation, then we run *dc2* up to saturation; (*ii*) another one called *seq2*, which is similar to *seq1*, but in the opposite order (first *dc2*, then *resyn2*); and (*iii*) a third one called *mix*, in which each iteration is an execution of *resyn2; dc2; dc2* and we run it up to saturation.

For the best outcome, we adopt a conservative, high-effort approach for the AIG node count minimization in Step 1.1. Thus, we propose to run all the five different minimization techniques and proceed with the minimum-node-count solution.

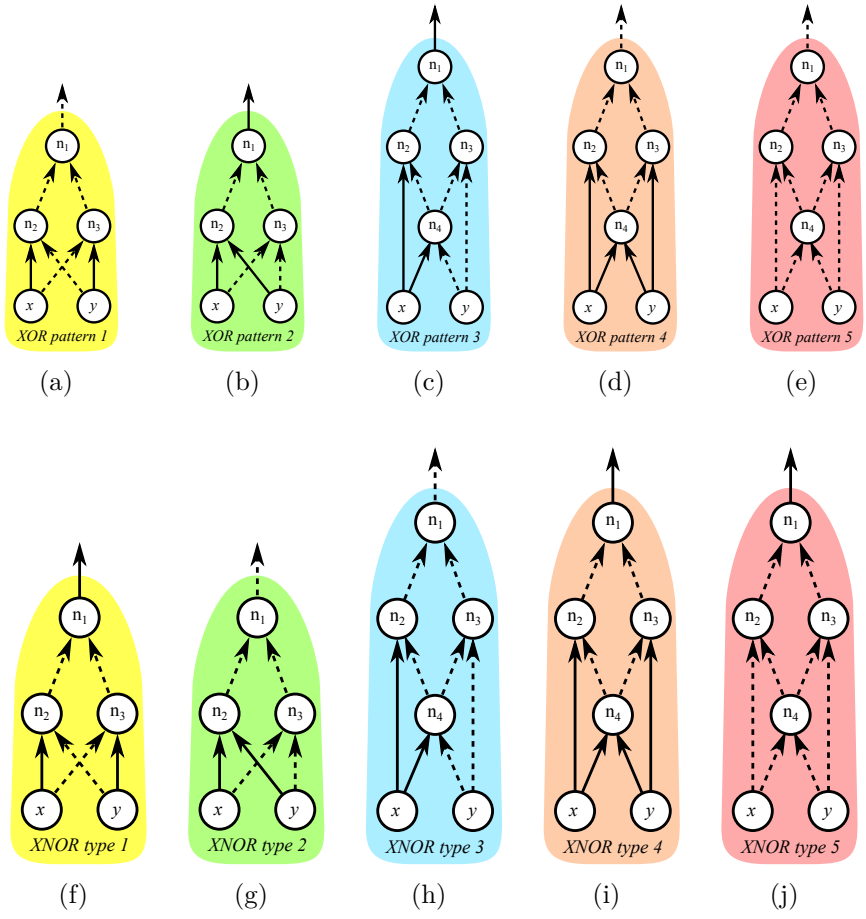
Although it is a fundamental step on the proposed synthesis flow, this node count minimization can be thought as a preprocessing on the AIGs before starting the execution of the proposed algorithms. Also, notice that we are not proposing a new algorithm for AIG node count minimization, but a new strategy to obtain a better outcome out of the already existing state-of-the-art algorithms for this purpose. Based on these two main reasons, we are not presenting in this thesis a time complexity analysis of this step on the proposed flow.

4.3 XAIG Generation

Once an optimized AIG is obtained, optimized XAIGs can be generated. In this section, we present in details the Step 1.2 of the proposed synthesis flow, i.e., the proposed procedure for XAIG generation.

A key role to create XAIGs out of AIGs is to identify the XOR/XNOR logic over the AIG nodes. Essentially, there are two main methodologies for this purpose: (1) Boolean-based identification, which would derive a Boolean representation for portions of the graph and seek for XOR/XNOR functions; or (2) pattern-based identification, which relies on traversing the graph looking for predetermined XOR/XNOR patterns. In this work, we follow a pattern-based approach. The predetermined XOR/XNOR patterns adopted herein are illustrated in Figure 4.4.

Figure 4.4: The XOR/XNOR patterns adopted in this work for the XAIG generation.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Algorithm 4.1 presents a pseudocode of the XAIG generation procedure. In order to look for such patterns, we propose to traverse the AIG in reverse topo-

logical order (from outputs to inputs). Then, we greedily replace every matched pattern that is a fanout-free cone, i.e., in which no internal node communicates with the remaining graph. Taking each of the patterns in Figure 4.4 as examples, only nodes n_1, x and y can have edges outgoing to other nodes not represented in these subgraphs. If this is not the case, the matched pattern is not replaced by the XOR node.

Algorithm 4.1: Generating an XAIG from an AIG.

```
   /* FFC stands for fanout-free cone                                  */
 1 xaig generateXAIG(aig)
 2   foreach (andNode ∈ aig in topo order)do
 3     res = table look-up XOR/XNOR pattern;
 4     if (res is an XOR pattern && res is an FFC)then
 5       replace res with xorNode;
 6     else if (res is an XNOR pattern && res is an FFC)then
 7       replace res with xnorNode;
 8   return aig as an xaig;
```

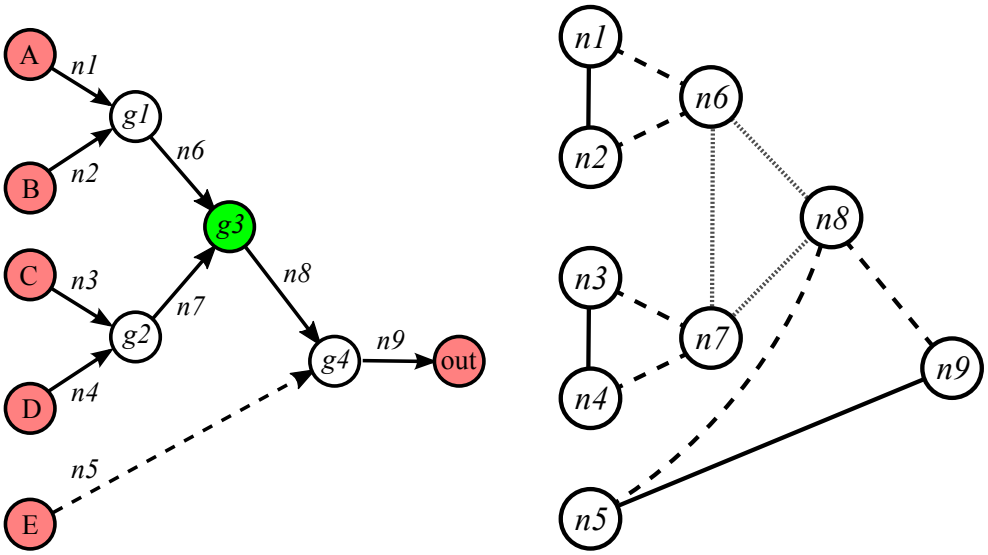
Given an AIG, let n be its number of AND nodes and m be its number of edges. In Algorithm 4.1, the loop between lines 2 and 7 will always execute n times and each execution can run in constant time. Thus, Algorithm 4.1 has a time complexity of $\Theta(n)$, which means that it is asymptotically bounded both above and below by a function $g(n)$.

4.4 Deriving Polarity Graphs from XAIGs

Step 2.1 starts by deriving a polarity graph from the optimized XAIG. For this, we propose a procedure enhanced from Jain and Bryant (1993). We derive polarity graphs using NAND2 as base function, applying NAND2/NOR2 phase constraints, and adopting an extra type of edge. Along with positive and negative edges, as proposed by Jain and Bryant (1993), we create neutral edges in between inputs and outputs of XOR2 nodes. Figure 4.5 illustrates the proposed procedure.

Figure 4.5(a) depicts an XAIG obtained after Step 1.2. The derived polarity graph for NAND2/NOR2 phase constraints is presented in Figure 4.5(b). Each set of outgoing edges from XAIG nodes (i.e., each net of the circuit) derives a node in the polarity graph. The XAIG node *g1* (Figure 4.5(a)) determines the polarity

Figure 4.5: An optimized XAIG (a) and the derived polarity graph (b).



Source: Author.

constraints between the nodes *n1*, *n2* and *n6* in the polarity graph (Figure 4.5(b)). As *g1* implements an AND2 operation (none of its inputs is inverted), applying NAND2/NOR2 phase constraints imply that (*i*) both of its inputs need to stay in the same polarity and (*ii*) its output needs to stay in a different polarity of its inputs. For this reason, (*i*) nodes *n1* and *n2* in the polarity graph (Figure 4.5(b)) are connected through a positive edge (solid line) and (*ii*) node *n6* is connected through negative edges (dashed lines) with nodes *n1* and *n2*. Notice that the polarity nodes *n6*, *n7* and *n8* are connected between each other through a neutral edge (dotted line). This is because *n6*, *n7* and *n8* are the inputs and output of an XOR2 node (*g3*). Applying the illustrated process to all nodes in the XAIG depicted in Figure 4.5(a) derives polarity graph shown in Figure 4.5(b).

Algorithm 4.2 presents a pseudocode for the proposed procedure to derive polarity graphs from XAIGs. The algorithm has only one input: the optimized XAIG. After creating the polarity nodes from the XAIG nets (lines from 2 to 4), the neutral edges are created for those polarity nodes related with XOR nodes 5. Then, the remaining edges between the polarity nodes are created (line 6. The derived polarity graph is the method's return. Algorithms 4.3 and 4.4 present pseudocodes detailing the connections related to XOR nodes and AND nodes, respectively.

Given an XAIG, let n be its number of both AND nodes and XOR nodes, and m be its number of edges. In Algorithm 4.2, the lines from 2 and 4 have a time

Algorithm 4.2: Deriving a polarity graph from an XAIG.

```
1 polGraph derivePolGraphFromXaig(xaig)
2     "create a polarity node for each XAIG net";
3     "map each polarity node with the related XAIG nodes";
4     "add each polarity node to the polarity graph";
5     connectXors(xaig, polarityGraph);
6     connectAnds(xaig, polarityGraph);
```

Algorithm 4.3: Connecting the polarity nodes related with XOR nodes.

```
1 void ConnectXorNodes(xaig, polarityGraph)
2     foreach (xorNode ∈ xaig)do
3         input1 = xaig.getPolNode(xorNode.getInput1);
4         input2 = xaig.getPolNode(xorNode.getInput2);
5         output = xaig.getPolNode(xorNode);
6         polGraph.createNeutralEdgeBetween(input1,input2);
7         polGraph.createNeutralEdgeBetween(input1,output);
8         polGraph.createNeutralEdgeBetween(input2,output);
```

complexity of $\Theta(n)$. Lines 5 and 6 also have a time complexity of $\Theta(n)$ each (see Algorithms 4.3 and 4.4). Thus, Algorithm 4.2 also has a time complexity of $\Theta(n)$.

4.5 Improvements in the Graph Coloring Procedure

In this work, we adopt a graph coloring procedure improved from the work by Matos (2014), which, in turn, is based on Jain and Bryant (1993). Still, the problem of coloring the polarity graph is heuristically solved in a very similar way. For this reason, no pseudocode is presented in this section concerning the coloring heuristics and we refer the reader to Section 3.4 for more details. In this section, we focus on describing the major improvements proposed herein regarding the coloring step.

The graph coloring procedure for inverter count minimization proposed by Jain and Bryant (1993) can be taken into a different perspective. A polarity node *n1* colored with a positive color means that, for that particular implementation, its corresponding net in the final circuit will keep its initial phase assignment. In contrast, another polarity node *n2* colored with a negative color means that, for that particular implementation, its corresponding net in the final circuit will change into the opposite polarity as the one in the initial circuit. Thus, in general terms,

Algorithm 4.4: Connecting the polarity nodes related with AND nodes.

```
1  void ConnectAndNodes(xaig, polarityGraph)
2      foreach (andNode ∈ xaig)do
3          input1 = xaig.getPolNode(andNode.getInput1);
4          input2 = xaig.getPolNode(andNode.getInput2);
5          output = xaig.getPolNode(andNode);
6          if (andNode.input1Polarity() ==
            andNode.input2Polarity())then
7              polGraph.createPositiveEdgeBetween(input1,input2);
8          else
9              polGraph.createNegativeEdgeBetween(input1,input2);
10         if (andNode.getNOutputs() == 1 &&
            andNode.outputNodeIsPrimaryOutput())then
11             outputNode = andNode.getOutputNode();
12             if (andNode.input1Polarity()! = outputNode.polarity())then
13                 polGraph.createPositiveEdgeBetween(input1,outputNode);
14             else
15                 polGraph.createNegativeEdgeBetween(input1,outputNode);
16             if (andNode.input2Polarity()! = outputNode.polarity())then
17                 polGraph.createPositiveEdgeBetween(input2,outputNode);
18             else
19                 polGraph.createNegativeEdgeBetween(input2,outputNode);
20         else
21             if (andNode.input1Polarity())then
22                 polGraph.createPositiveEdgeBetween(input1,outputNode);
23             else
24                 polGraph.createNegativeEdgeBetween(input1,outputNode);
25             if (andNode.input2Polarity())then
26                 polGraph.createPositiveEdgeBetween(input2,outputNode);
27             else
28                 polGraph.createNegativeEdgeBetween(input2,outputNode);
```

the graph coloring procedure proposed by Jain and Bryant (1993) plays with De Morgan's law seeking to minimize explicit inverters inside the circuit as the inversions are pulled/pushed into its PIs/POs.

We observed the described behavior and propose to use it in our benefit in order to minimize even further the inverter count by exploring polarity don't care nodes. A first case relies on the XOR nodes. As we described in Section 2.6, both XOR2 and XNOR2 functions can be implemented with the same number of transistors independently on the number of inversions in their inputs and outputs. For this reason, we say that XOR2/XNOR2 functions are polarity don't care in terms of transistor count.

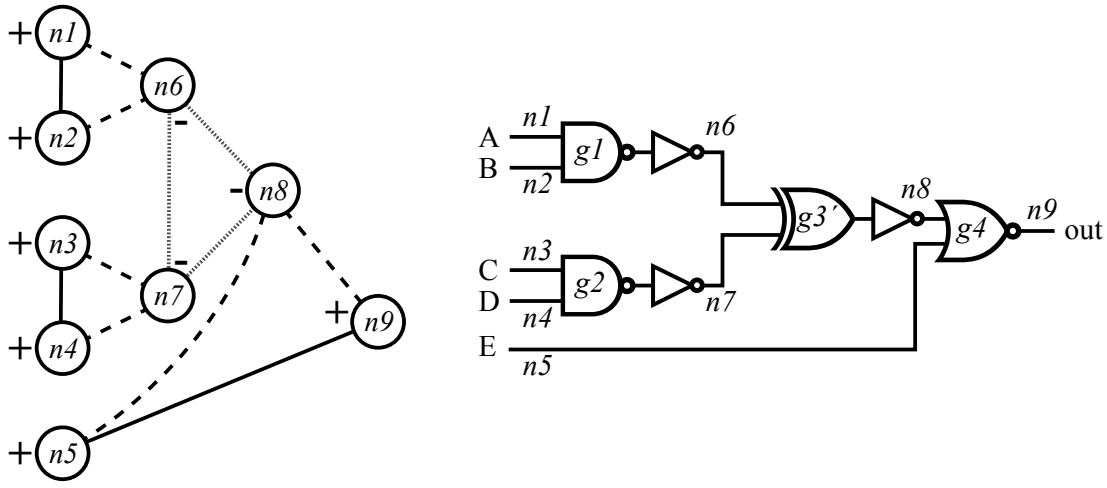
In order to explore XOR nodes as polarity don't care nodes, we propose to treat them as pseudo primary inputs and pseudo primary outputs of the circuit. The inputs of an XOR node are treated as pseudo primary outputs for their downstream network. Similarly, the output of an XOR node is treated as a pseudo primary input for its upstream network. By doing this, we are able to set the graph coloring procedure to force the explicit inversions into the XOR inputs and outputs so that we can get rid with them when deriving the final circuit.

Figure 4.6 illustrates the proposed technique. By applying the described approach, the polarity graph depicted in Figure 4.5(b) can be colored as shown in Figure 4.6(a). A direct translation from this colored version would generate a circuit as the one in Figure 4.6(b), with 3 explicit inversions. However, all the 3 inverters can be removed and the XOR2 cell can be replaced by an XNOR2 cell so that the circuitt functionality remains the same and the inverter count is improved even further. The proposed approach to derive optimized circuits out of colored polarity graphs is described in Section 4.6.

We also explore two-output flip-flops as polarity don't care nodes in terms of transistor count. With this, we bring two contributions at once. This is because both the previous works, Jain and Bryant (1993) and Matos (2014), could not directly handle sequential circuits. We not only handle flip-flops, but we also minimize the inverter count even further if two-output flip-flops are available in the library.

Two-output flip-flops are commonly available in standard cell libraries. This kind of flip-flop is able to provide both the registered value, at the Q output, and its complement, at the $\overline{Q}$ output. We say that two-output flip-flops are polarity don't care in terms of transistor count because, by cleverly swapping outputs from Q to

Figure 4.6: A colored polarity graph (a) and a valid circuit implementation (b). Notice that all the explicit inverters in (b) can be removed by exploring the XOR2 cell as a polarity don't care node.



(a) (b)

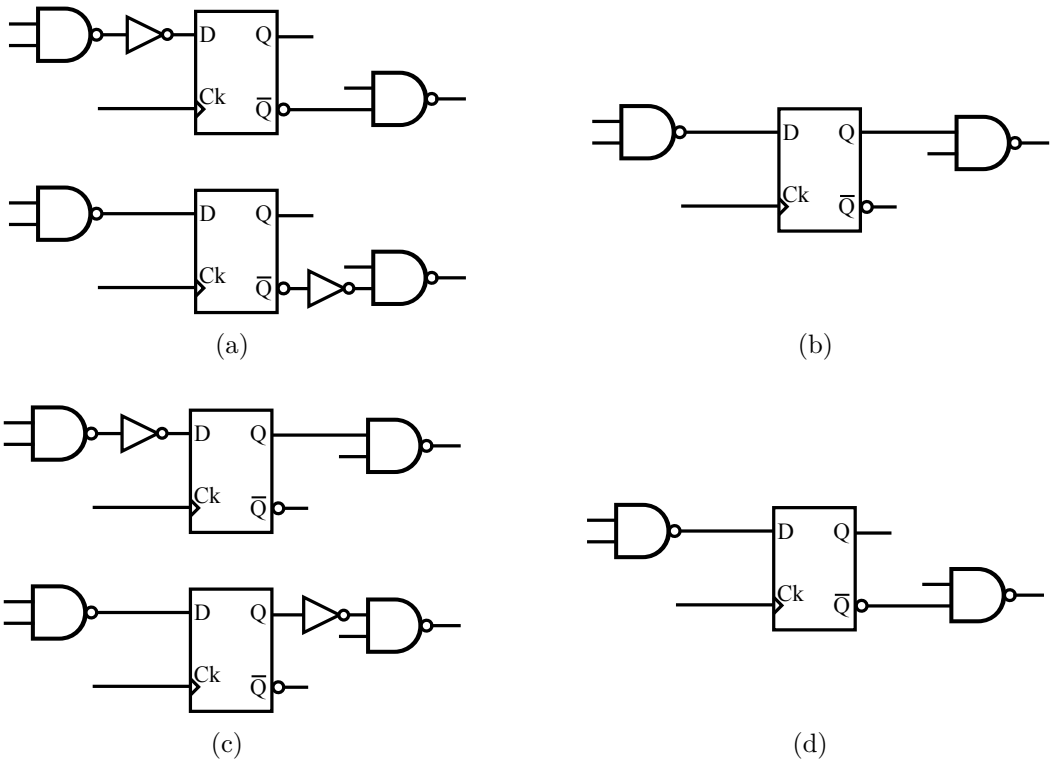
Source: Author.

$\overline{Q}$ (and vice-versa), no inverters are needed at a final implementation. Figure 4.7 illustrates four cases implemented with inverters (Figures 4.7(a) and 4.7(c)) that can be replaced by implementations with no inverters at all (Figures 4.7(b) and 4.7(d)).

This way, we propose to also treat two-output flip-flops as polarity don't care nodes and bring the inverters to their inputs and outputs. To do so, we adopt an approach similar to the one applied for XOR nodes: (1) flip-flop inputs should be treated as pseudo outputs for their downstream network; and (2) flip-flop outputs need to be treated as pseudo primary input for their upstream network. By doing this, similarly to the case of XOR nodes, we are able to set the graph coloring procedure to force the explicit inversions into the flip-flops inputs and outputs so that we can get rid with them when deriving the final circuit.

That all said, the time complexity of the algorithms run at Step 2.1 is as follows. Given an XAIG, let n be its number of both AND nodes and XOR nodes, n_{PO} be its number of PO nodes, and m be its number of edges. The core of the graph coloring procedure will keep looping until there are no odd cycles (please, see Algorithms 3.1 and 3.2). Each iteration tries to color the polarity graph, what can be done either through a BFS or through a DFS, each with time complexity of $\mathcal{O}(n+m)$. As, in the worse case, there will be n attempts of coloring the polarity graph, Step 2.1 has a time complexity of $\mathcal{O}(n^2+nm)$. Notice that the number of edges m of an AIG is $2n+n_{PO}$. Thus, Step 2.1 still has a quadratic time complexity.

Figure 4.7: Two-output flip-flops as polarity don't cares in terms of transistor count: two cases with inverters (a) that can be replaced by an implementation with no inverters (b); other two cases with inverters (c) that can be replaced by an implementation with no inverters (d).



Source: Author.

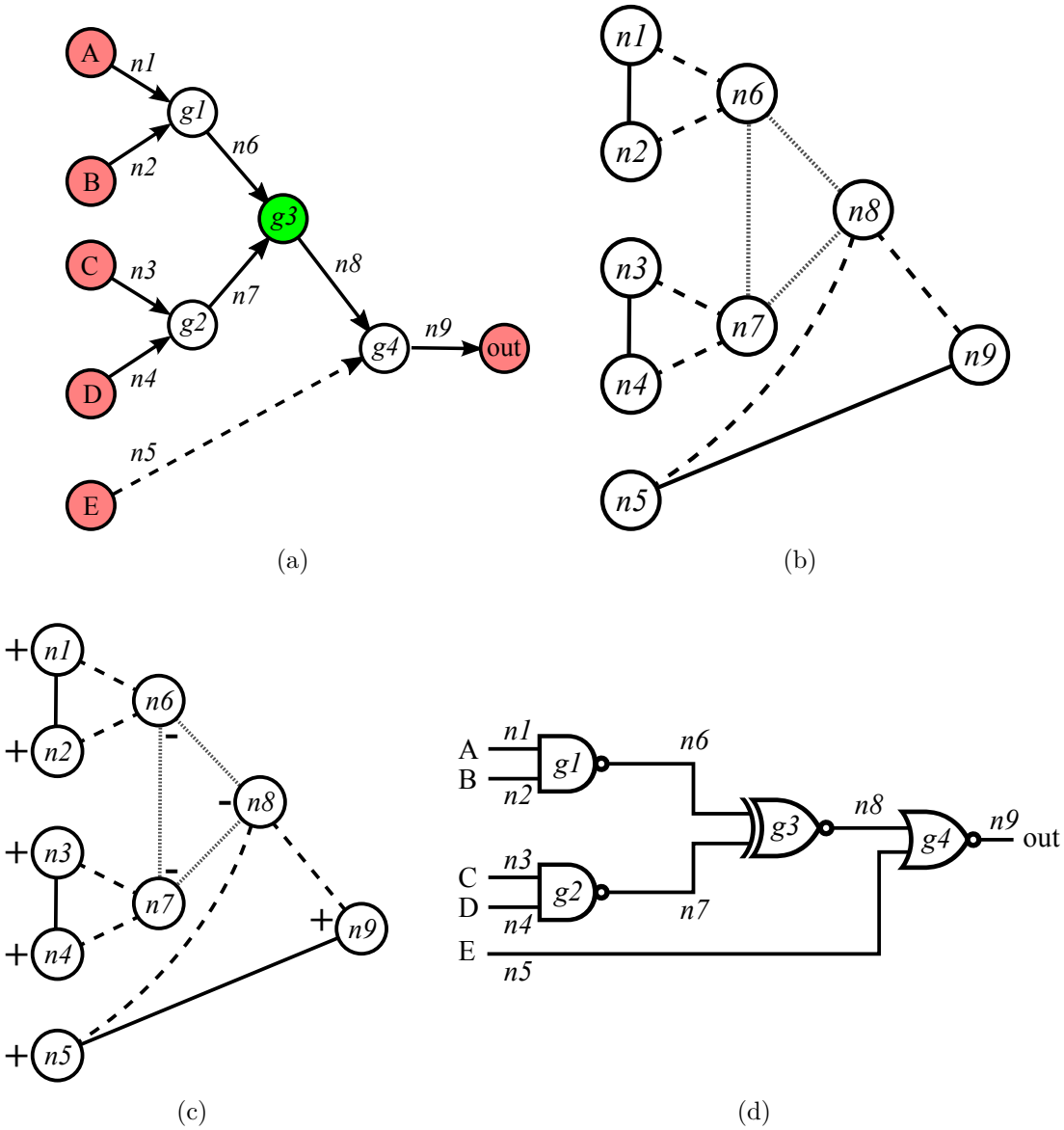
4.6 Deriving Optimized Circuits from Colored Polarity Graphs

Once the polarity graph is colored, the optimized circuit can be derived in Step 2.2. This section describes the proposed method to deriving optimized circuits from colored polarity graphs, which is enhanced from Matos (2014). Figure 4.8 presents a complete example of the mapping process. The optimized circuit in Figure 4.8(d) was generated from the colored polarity graph in Figure 4.8(c).

The proposed process is based on: (i) the times operator ($\bullet$); *(ii)* the initial polarity of nets (γ); and *(iii)* the final color of each net n_j (*color*(n_j)). Given an XAIG node g_i and its corresponding polarity nodes n_j, n_k and n_l, the pattern $\{\gamma(g_i, n_j) \bullet color(n_j), \gamma(g_i, n_k) \bullet color(n_k), \gamma(g_i, n_l) \bullet color(n_l)\}$ must lead to a known pattern in the phase constraint set. In this work, the known patters are $\{+, +, +\}$ for a NAND2 cell and $\{-, -, -\}$ for a NOR2 cell. The XOR2/XNOR2 cells are handled in a different way.

As an example, consider the XAIG node *g1* and its corresponding polarity

Figure 4.8: A complete example of the proposed mapping process: an optimized XAIG (a), the derived polarity graph (b), the colored polarity graph (c) and the final circuit (d).



Source: Author.

nodes *n1*, *n2* and *n6*. Applying the times operator in this case derives the NAND2 pattern $\{+,+,+\}$, as follows:

$$\begin{aligned} color(n1) \bullet \gamma(n1, g1) &= + \bullet + = + \\ color(n2) \bullet \gamma(n2, g1) &= + \bullet + = + \\ color(n6) \bullet \gamma(n6, g1) &= - \bullet - = + \end{aligned} \tag{4.1}$$

For this reason, cell *g1* in Figure 4.8(d) is mapped into a NAND2 cell.

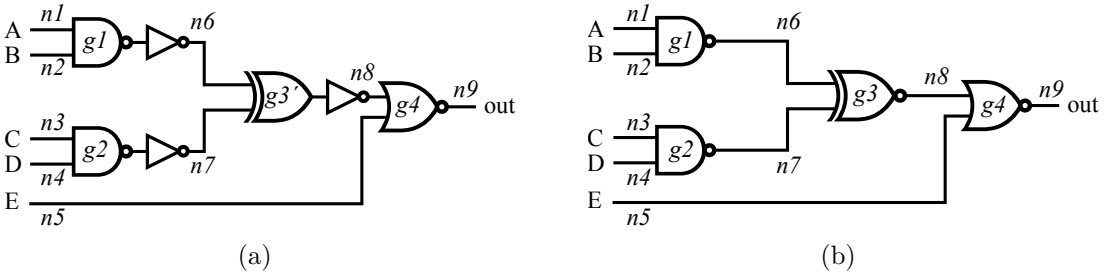
Similarly, consider the XAIG node *g4* and its corresponding polarity nodes *n5*, *n8* and *n9*. Applying the times operator in this case derives the NOR2 pattern $\{-,-,-\}$, as follows:

$$\begin{aligned} color(n5) \bullet \gamma(n5, g4) &= + \bullet - = - \\ color(n8) \bullet \gamma(n8, g4) &= - \bullet + = - \\ color(n9) \bullet \gamma(n9, g4) &= + \bullet - = - \end{aligned} \tag{4.2}$$

For this reason, cell *g4* in Figure 4.8(d) is mapped into a NOR2 cell.

Notice that, as described in Section 4.5, the polarity nodes *n6* and *n7* are inputs of an XOR node in the XAIG. Thus, they are treated as pseudo primary outputs and their negative color (Figure 4.8(c)) derive two explicit inverters. The same occurs with the polarity node *n8*, which is the output of an XOR node in the XAIG and, thus, is treated as a pseudo primary input. In this case, according with the discussion in Section 2.6, all three inverters (an odd number) can be removed and the XOR node will derive an XNOR2 cell instead. Figure 4.9 illustrates this optimization process.

Figure 4.9: An example of XOR/XNOR optimization when deriving the final circuit.



Source: Author.

Algorithm 4.5 presents a pseudocode for Step 2.2. The method has only one input: the colored polarity graph. The proposed algorithm starts by creating the input and pseudo input pins (lines 2 to 5) and the output and pseudo output pins (lines 6 to 8). After creating one inverter for each removed node, the NANDs and NORs are mapped (line 10. Then, the XOR/XNOR cells are derived (lines 11 23. Notice that: (*i*) if the library does not have XORs/XNORs, these nodes can be mapped with NANDs/NORs and inverters and also take benefit of the applied optimizations; and (*ii*) the inverters are created on input and output nodes, depending on the final color and the derived cell. These inverters are created in line 4 and lines 24 to 25, respectively. The method's return is the circuit after mapped.

Algorithm 4.5: Derive an optimized circuit from a colored polarity graph.

```
1  circuit deriveCircuitFromColoredGraph(polGraph)
2      "create an input pin for each input node";
3      "create a pseudo input pin for each output of both XOR and FF
        nodes";
4      "create an inverter for each non-removed, negative-colored input
        node";
5      "mark input and pseudo input nodes as mapped";
6      "create an output pin for each output node";
7      "create a pseudo output pin for each input of both XOR and FF
        nodes";
8      "insert these output pins into the circuit";
       // polarity nodes w.r.t.  output and pseudo output pins
          will be mapped further.
9      "create an inverter for each removed node";
10     mapNandsNors(polGraph);
11     foreach (xorNode ∈ polGraph)do
12         count = 0;
13         if (xorNode.input1IsInverted())then
14             count++;
15         if (xorNode.input2IsInverted())then
16             count++;
17         if (xorNode.outputIsInverted())then
18             count++;
19         if (count "is pair")then
20             "map xorNode with XOR cell";
21         else
22             "map xorNode with XNOR cell";
23         "remove the inverters at xorNode inputs and outpus";
24     foreach (nonInvertedOutput ∈ polGraph.getOutputNodes())do
25         "consider inverting this output performing the same tests in
            Algorithm 4.6, lines from lines 11 to 17";
26     return circuit;
```

The core of the procedure to derive NANDs and NORs is illustrated in the pseudocode in Algorithm 4.6. The algorithm derives NAND and NOR cells according to the obtained patterns.

Algorithm 4.6: Proposed procedure to map NANDs and NORs.

```
1  void mapNandsNors(polGraph)
2      mapped = false;
3      while (!mapped)do
4          mapped = true;
5          foreach (polNode ∈ polGraph.getUnmappedNodes())do
6              "skip polNode if it is related with XOR nodes" if
                (polNode.noneInputsWereRemoved())then
7                  "map this node according to its inputs' coloring pattern";
8                  "NAND patterns derive NAND cells and NOR patterns
                    derive NOR cells";
9              else
10                 if ("removed input is already mapped")then
11                     if (polNode.getNonRemovedInputPattern == "NAND
                        pattern")then
12                         if ((removedInput.isNandCell() &&
                            correspondingAigNodeIsNotInverted) ||
                            (removedInput.isNorCell() &&
                            correspondingAigNodeIsInverted))then
13                             nandCell.connectToInvertedInput();
14                         else
15                             nandCell.connectToDirectedInput();
16                         circuit.addNandCell(nandCell);
17                         polNode.markNodeAsMapped();
18                     else if (polNode.getNonRemovedInputPattern ==
                        "NOR pattern")then
19                         "do as in lines from 11 to 17, considering the
                            opposite polarities.
20                 else
21                     mapped = false;
22                     break;
                       /* the removed inputs shall be mapped first
                          */
```

Thus, given an XAIG, let n be its number of AND nodes, x be its number of XOR nodes, and m be its number of edges. In Algorithm 4.5, the lines from 2 to 8 visit each PI and PO once. Then, line 10 can map NANDs and NORs cells by visiting each AND node n_i once. Also, lines from 11 to 23 can map XORs and

XNORs cells by visiting each XOR node x_i once. Finally, lines from 24 to 25 visit each PO node once more. So, Algorithm 4.5 has a linear time complexity.

4.7 Area-Oriented Level-Aware Fanout Limitation

Algorithms to reduce node count in AIGs, like the ones in Section 4.2, tend to increase the logic sharing. The greater is the logic sharing, the smaller tends to be the AIG, especially when it results from an algorithm for minimizing the node count. However, a potential drawback of this is that they also tend to increase the fanout of AIG nodes. A logic cell with excessive fanout has negative impact in the circuit performance for most of the current fabrication technologies. In this section, we describe the proposed algorithm to limit fanout, which is both area-oriented and level-aware.

The proposed synthesis flow may generate circuits in which some cells have fanout greater than desired. Thus, given maximum allowed fanouts, the algorithm proposed in this section fixes all fanout violations by inserting inverter trees. This approach takes into account the maximum fanout of the source logic cells and the maximum fanout of inverters. Both the maximum fanout values are determined by the user through algorithm parameters. The root of each inverter tree is the source logic cell with a fanout violation. We say this algorithm is area-oriented because it fixes all fanout violations by inserting inverter trees with a minimized number of inverters.

The proposed algorithm has two substeps. In Step 3.1, it finds the minimum tree height which respects both the number of positive consumer cells (PCC) and the number of negative consumer cells (NCC). Step 3.2 creates the inverter tree itself. In order to find the minimum tree height, the proposed algorithm uses an exhaustive search approach. It starts with the smallest heights (0 for PCC and 1 for NCC) and verifies if the fanout limit of the cell can be respected by these levels. Otherwise, the method increases the heights by one until it finds the levels which do not lead to fanout violations.

Let MFC be the maximum fanout of source cells, MFI be the maximum fanout of inverters, ip be the index for positive consumers (i.e., the maximum tree level for PCC), and in be the index for negative consumers (i.e., the maximum tree level for NCC). The resulting values for ip and in are obtained from the procedure

illustrated in Algorithm 4.7.

Algorithm 4.7: Proposed procedure to compute the minimum height for an inverter tree.

```
1  {ip,in} computeMinHeight(MFC, MFI, PCC, NCC)
2      done = false;
3      for (i = 0; !done; i++)do
4          if (i "is pair")then
5              ip = i;
6              in = i + 1;
7              if (MFC^(ip+1) ≥ PCC &&
                 MFI^(ip+1) − PCC) ∗ MFI ≥ NCC)then
8                  done = true
9          else
10             in = i;
11             ip = i + 1;
12             if (MFI^(in+1) ≥ NCC &&
                 MFI^(in+1) − NCC) ∗ MFI ≥ PCC)then
13                 done = true
14     return {ip,in};
```

In Step 3.2, we apply a local search procedure to heuristically generate the inverter tree (respecting the solution of Step 3.1) while trying to minimize the number of inverters of this tree. In each step of the search, the proposed procedure: removes from the current tree a given subtree of height 1 (and the corresponding inverter); minimizes the number of inverters in the remaining tree; and, finally, tries to reallocate the removed subtree without introducing new inverters. The first subtree for which this procedure succeeds is accepted, and the local search is repeated for all subtrees until no further improving subtree can be found. This is typically achieve after two iterations.

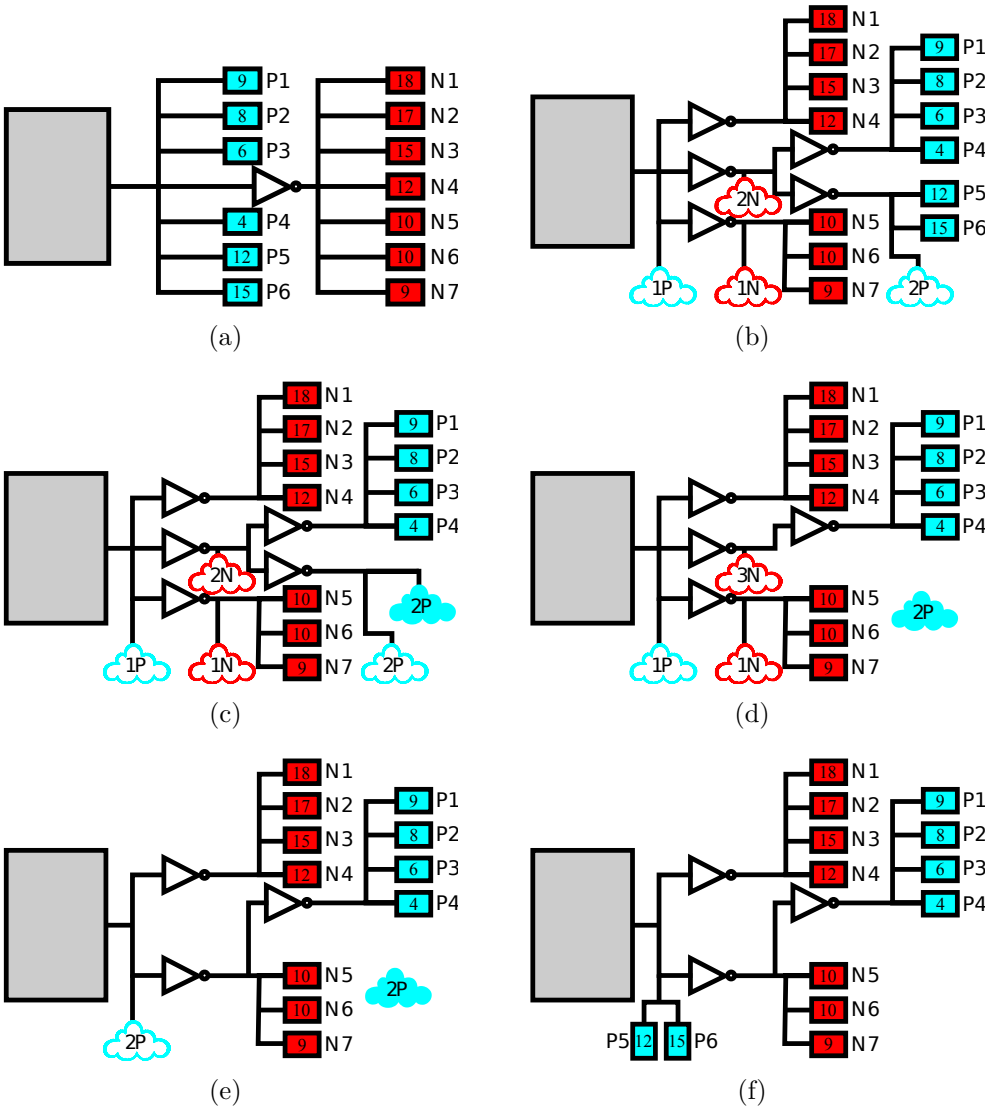
The algorithm we introduce herein extends the buffering algorithm proposed by Matos et al. (2014) to make it level-aware. To do that, we simply sort both the positive and negative consumer cells according to their logic depth. Consumer cells with higher logic depth are assigned as close as possible to root of the tree. This simple (yet effective) change yields final circuits with a better performance.

Figure 4.10 depicts the algorithm of Step 3.2 being applied. In this figure, the gray boxes denote the source cell, the blue boxes are the positive consumer cells (PCC), and the red boxes are the negative consumer cells (NCC). The numbers

inside these boxes denote the cells' logic depth. For this example, consider both the maximum fanout of inverters (MFI) and the maximum fanout of source cells (MFC) as 4. This way, Figure 4.10(a) illustrates two fanout violations, since both the source cell and the inverter have fanout 7. Applying Step 3.1, the minimum tree height is 2, the index of positive cells (ip) is 2 and the index of negative cells (in) is 1. Figure 4.10(b) shows the subcircuit after allocating all PCC and NCC in their respective indexes and propagating the required number of inverters. Figure 4.10(c) depicts the subcircuit after removing two PCC (*P5* and *P6*) for further reallocation. In Figure 4.10(d), The needless inverter on level 2 is removed. Figure 4.10(e) shows the subcircuit after propagating the needed number of inverters. Figure 4.10(f) presents the subcircuit after *P5* and *P6* are reallocated to level 0. Notice that, after propagating the required inverters, the number of inverters reduces from 5 to 3. Notice also that, as cells *P5* and *P6* are the ones with higher logic depth among the positive consumer cells, they were chosen to be assigned closer to root of the three.

In this sense, given an XAIG, let n be its number of both AND nodes and XOR nodes, and m be its number of edges. The time complexity of the algorithms executed in Steps 3.1 and 3.2 are as follows. The worst case would be that in which all XAIG nodes have a fanout violation. In this case, Step 3.1 would need to find an inverter tree height for each n_i node and run at most $\log n$ checks in each of them. This indicates that this step has a time complexity of $\mathcal{O}(n \log n)$. Step 3.2 would need to visit each node once more and typically finishes with less than 2 iterations. Thus, this step would have a linear time in the average-case complexity.

Figure 4.10: Example of fanout violation (a), substeps to limit fanout (b, c, d and e) and fanout limited circuit using an inverter tree (f).



5 DETAILS THAT MATTER

As the devil is in the details, the work proposed in this thesis can be improved slightly more by paying due attention to some specifics. We refer to them as secondary contributions. Thus, in this chapter, four secondary contributions are presented, each of which representing a relevant detail of this work. Two of the contributions are related with synthesis speedup: (1) an statistical analysis was made on the possible synthesis parameters and three effort levels are proposed to trade-off QoR and runtime; and (2) we propose to use coarse-grain parallel approaches to speedup both the AIG minimization step and mapping procedure. The other two contributions are related with improvements on QoR: (1) we propose to merge locally equivalent flip-flops to reduce transistor count; and (2) we explore two-output flip-flops even further to reduce inverter count. These four contributions are described in details in the following sections.

5.1 Improving the QoR with Trade-off Optimizations

Matos (2014) propose an exhaustive trade-off optimization technique based on 5 binary parameters (abbreviation):

1. (I) forcing colors on Input nodes (yes or no);
2. (O) forcing colors on Output nodes (yes or no);
3. (U) removing Unfeasible fanout nodes in advance (yes or no);
4. (H) which color Heuristics to apply (*QuickColor* or *GoodColor*); and
5. (T) in which order to Traverse the graph when coloring it (BFS or DFS).

In order to obtain the best circuit implementation out of these 5 binary parameters, 32 circuits (2^5) need to be derived.

In this thesis, we perform a statistical analysis on the parameters and propose to allow the designer to choose among three different effort levels: (1) a high-effort synthesis, that would go for all the 32 iterations regarding the 5 possible parameters; (2) a medium-effort synthesis, which would explore only 2 of the 5 parameters and takes the best result out of 4 circuits; and (3) a low-effort synthesis, that goes for only one synthesis based on the best statistical results. With these three possible effort levels, the designer can decide which trade-off better fits with his expectations

on QoR versus runtime.

In order to perform the statistical analysis, we propose to derive all the 32 possibilities for each circuit in the ISCAS'85 benchmark suite (BRYAN, 1985). Then, for each circuit, we take into account all the sets of parameters that are equal to its best solution. The results are summarized in Table 5.1, in which the columns are labeled with the previously introduced abbreviation. Notice that the set of parameters 101GD was able to achieve the best solution in 40% of the circuits.

Table 5.1: Data for a statistical analysis on the possible synthesis parameters.

I	*O*	*U*	*H*	*T*	#
0	0	0	Q	D	10%
0	0	1	G	B	20%
0	0	1	G	D	20%
0	0	1	Q	D	10%
0	1	0	Q	D	10%
0	1	1	G	B	10%
0	1	1	G	D	20%
1	0	0	G	B	10%
1	0	0	Q	D	10%
1	0	1	G	B	30%
1	0	1	G	D	40%
1	0	1	Q	D	10%
1	1	0	G	D	20%
1	1	0	Q	D	10%
1	1	1	G	B	10%
1	1	1	G	D	20%
1	1	1	Q	D	10%

Based on these results, we decided to adopt the effort levels as follows:

High-effort: run all the 32 possibilities;

Medium-effort: run only 4 possibilities by exploring parameters *I* and *O*;

Low-effort: run only 1 possibility with parameters 101GD.

5.2 Improving Runtime with Parallel Synthesis

As introduced in the previous section, one way to speed up the synthesis process is to trade off runtime and QoR with effort levels. In this section, we introduce a second way to speed up the synthesis, which is based on parallel synthesis.

According to Hwang (2007), the granularity of a task in parallel computing

is a measure of the amount of computation performed by that task. In this sense, in the context of this thesis, the synthesis process could be split into either: (1) a large number of small tasks (fine-grained), as if we parallelize the graph coloring procedure; or (2) a small number of large tasks (coarse-grained), as we propose to split the 1 serial execution of 32 tasks for the high-effort synthesis into 32 parallel executions with the same purpose. In terms of processing levels, again in the context of this thesis, the synthesis could be (among others): at instruction level, where the parallelism is coded with instructions in a given programming language; or at procedure level, in which procedures of the synthesis flow can be ran in parallel seeking for speedup. Thus, as the high-effort synthesis level is based on 32 different executions of the same task, each of which running with a different set of parameters, this methodology perfectly fits with coarse-grained, procedure-level parallelism.

In order to do that, we propose to adopt a thread pool approach. With this, the user can define the desired number of threads to be created and, then, the thread pool keeps this number of threads running while managing the tasks to be executed. Algorithm 5.1 presents a pseudocode for the proposed parallel flow. The algorithm starts by creating the thread pool and dispatching the threads (lines 1 to 5). Then, each thread keeps running until all tasks are finished. For this, a task manager needs to control the queued/finished tasks, which is represented by the *getWork* call. Also, the threads must register the derived circuits when each task is finished, so that the best circuit can be returned at the end of the process. This is represented by the *updateBestCircuit* call.

Algorithm 5.1: Thread pool approach to map circuits in parallel.

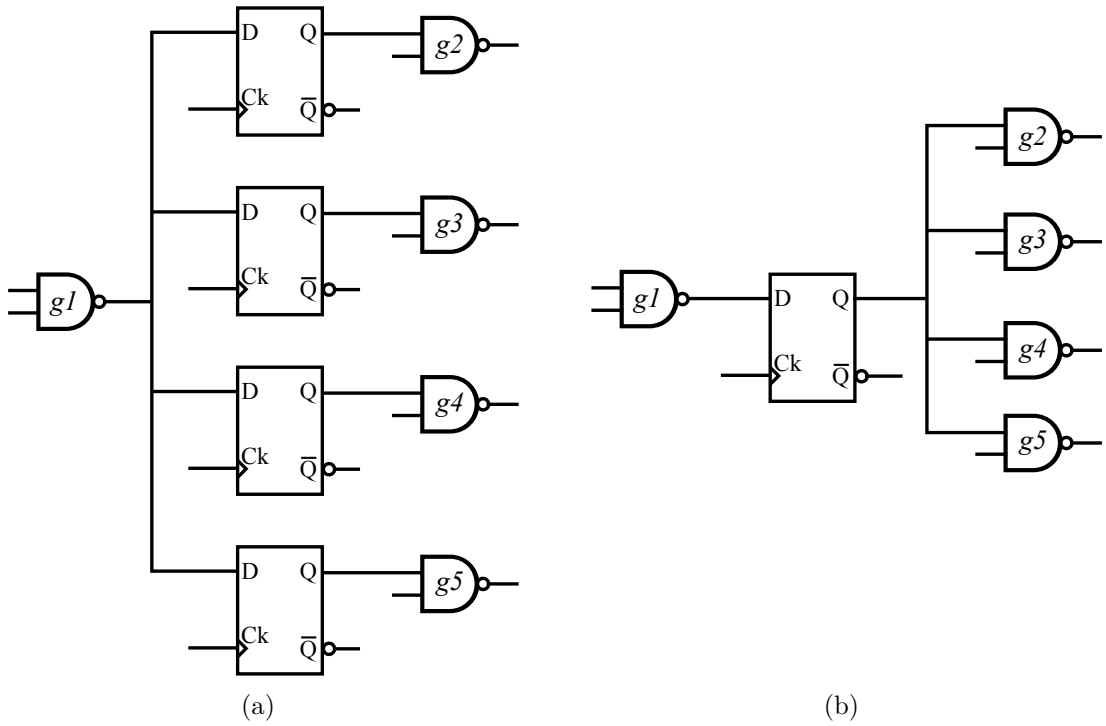
```
1 circuit parallelMapping(optXaig, MFI, MFC, numCores)
2     PolarityWorker* workers[numCores];
3     foreach (worker ∈ workers)do
4         worker.run(optXaig, MFI, MFC);
5     return PolarityWorker::getBestCircuit();
6 void run(optXaig, MFI, MFC)
7     for (flag = getTask(); flag < 32; flag = getTask())do
8         circuit = runSynthesisFlow(optXaig, MFI, MFC, task);
9         updateBestCircuit(circuit);
```

5.3 Merging Logically Equivalent Flip-Flops

In a sequential circuit, some flip-flops (FFs) can be logically equivalent. In this section, we show how to improve the QoR a bit further by merging logically equivalent FFs.

We say that two FFs are logically equivalent if they are fed by the same driver. Once equivalent FFs are identified, we propose to remove them from the circuit and connect their loads on a new, merged FF. Figure 5.1 illustrates four logically equivalent FFs before and after merging. This simple (yet effective) optimization is able to improve QoR a bit further both in terms of area and transistor count. Performance can be recovered with buffering if necessary.

Figure 5.1: Example of logically equivalent flip-flops: before merging (a) and after merging (b).

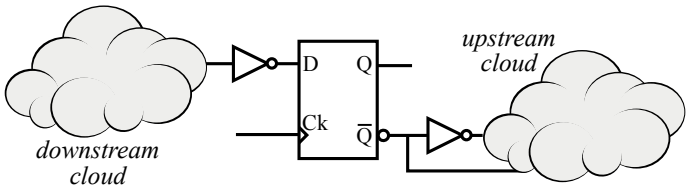


5.4 Exploring Two-Output Flip-Flops

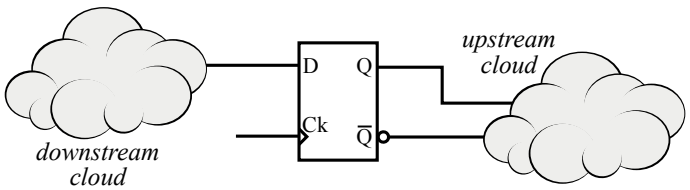
In Section 4.5, we show how to explore two-output FFs as polarity don't care nodes. Still, the QoR can be additionally improved by cleverly exploring the usage of both FF outputs when the signal is needed in both polarities.

Most of the logic synthesis tools, including commercial ones, treat combinational and sequential elements in different moments of the synthesis flow. In general terms, combinational clouds are extracted from sequential circuits and, then, the combinational synthesis is performed. Later on, when the logic is already defined, the sequential synthesis is ran. This methodology can lead to strange outcomes, as the one illustrated in Figure 5.2(a). In this case, while optimizing the downstream cloud, pin D of the FF was treated as pseudo primary output and, to preserve the circuit functionality, an inverter was needed to deliver the signal in the correct polarity. We already showed in Section 4.5 that this inverter can be removed. Still, while optimizing the upstream cloud, the synthesis process required both polarities from the illustrated FF and, as it is treated as pseudo primary input in this case, an inverter was used for the purpose. We already showed that this inverter can be removed as well.

Figure 5.2: Two-output flip-flops: ignoring Q while inverting $\overline{Q}$ to have both polarities (a) and correctly exploring Q and $\overline{Q}$ to have both polarities (b).



(a)



(b)

In this thesis, we propose to handle these cases differently. Although we also treat inputs and outputs of FFs as pseudo POs and pseudo PIs during the graph coloring process, we propose to treat them as real FFs when deriving the final circuit from colored polarity graphs. Thus, if two-output flip-flops are available in the library (as they commonly are), we are able to identify the cases in which both polarities are needed and explore both Q and $\overline{Q}$, as illustrated in Figure 5.2(b).

6 EXPERIMENTAL RESULTS

In order to validate the proposed simple-cell-based mapping, experiments were carried out using different sets of benchmark circuits. The proposed algorithms are implemented in Simple Flow tool (LogiCS Research Lab, 2017) using *C++11* programming language and compiled with *gcc 4.9.2*. In the following sections, we present some analysis of the proposed contributions, then a comparison with the state-of-the-art approaches.

6.1 Analysis of the AIG Node Count Minimization Approach

In this section, we present an analysis of the proposed strategies to minimize AIG node count. For this, we propose to use a single execution of ABC's *resyn2* as reference, then compare it with both ABC's *dc2* (also a single execution) and the strategies proposed in Section 4.2, which run up to saturation. The developed approaches are implemented in Simple Flow tool as the following commands: *opt_resyn2*, *opt_dc2*, *opt_seq1*, *opt_seq2*, and *opt_mix1*. This experiment was carried out on a personal computer with Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz, 8GB RAM, over the OpenCores benchmark circuits (PISTORIUS et al., 2007).

The obtained results are presented in Table 6.1. ABC's *resyn2* is taken as reference and we present its number of nodes. For the other commands, we present the ratio over the reference. Notice that, although ABC's *dc2* is an area-oriented version of *resyn2*, their results are very similar (*dc2* is 1% better, on average). However, the improvements of running them up to saturation are meaningful: *opt_resyn2* and *opt_dc2* are up to 9% better (4% on average) than their single execution. Finally, even though *opt_seq1* and *opt_seq2* could not overcome their saturation points and provide better results, *opt_mix1* was able to improve the QoR a bit further and provide results up to 10% better, 5% on average.

6.2 Analysis of the Runtime Speedup with Parallelization

In this section, we analyze one of the speedup approaches proposed in this thesis, which is based on the parallelization of the high-effort synthesis. For this

Table 6.1: Obtained results when analyzing the proposed AIG node count minimization approaches.

	ABC		*Simple Flow*				
circuit	*resyn2*	*dc2*	*opt_resyn2*	*opt_dc2*	*opt_seq1*	*opt_seq2*	*opt_mix1*
oc_aquarius	19653	(0.95)	(1.00)	(0.94)	(0.94)	(0.94)	(0.94)
oc_cfft_1024x12	8718	(0.99)	(0.98)	(0.98)	(0.96)	(0.97)	(0.96)
oc_cordic_p2r	8012	(1.00)	(0.99)	(0.98)	(0.98)	(0.98)	(0.98)
oc_cordic_r2p	10502	(1.00)	(0.96)	(0.93)	(0.92)	(0.92)	(0.90)
oc_des_perf	20970	(1.00)	(0.99)	(0.97)	(0.97)	(0.97)	(0.97)
oc_ethernet	8392	(0.99)	(0.99)	(0.98)	(0.98)	(0.96)	(0.97)
oc_fpu	15835	(1.00)	(0.96)	(0.95)	(0.94)	(0.94)	(0.93)
oc_mem_ctrl	13456	(0.93)	(1.00)	(0.91)	(0.92)	(0.91)	(0.92)
oc_video_dct	32000	(1.01)	(0.99)	(0.99)	(0.97)	(0.98)	(0.97)
oc_video_jpeg	41892	(1.00)	(1.00)	(0.99)	(1.00)	(0.99)	(1.00)
geomean		(0.99)	(0.99)	(0.96)	(0.96)	(0.96)	(0.95)

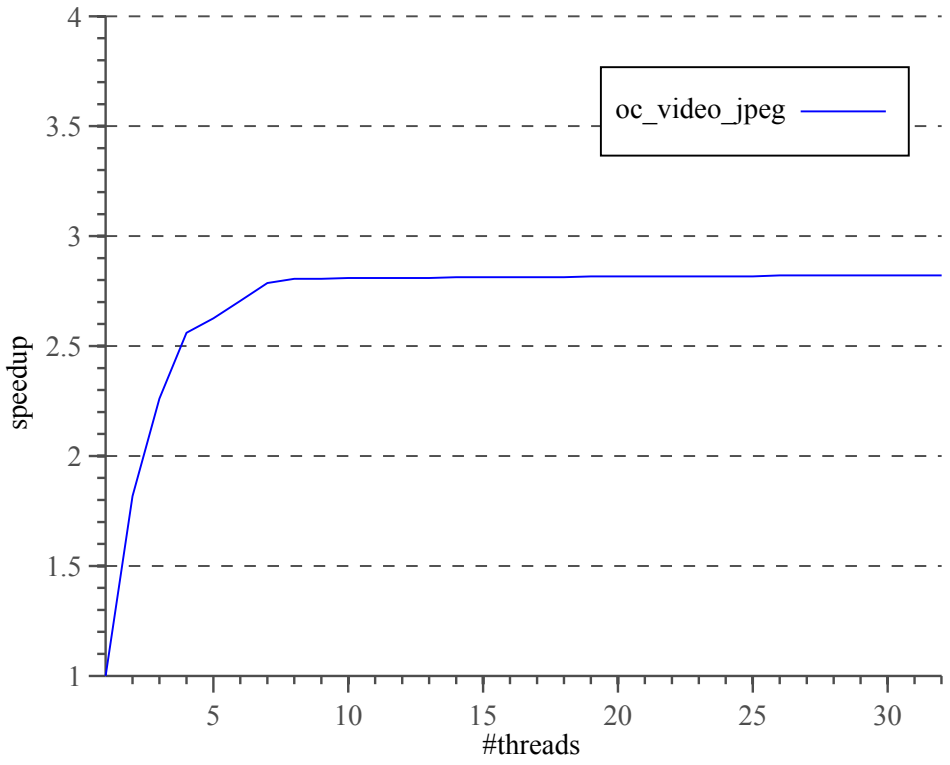
analysis, we propose to run the synthesis of the 32 tasks on this effort level varying the number of threads from 1 (single-threaded version) to 32 (one thread for each task). This experiment was carried out on a server with 2 Intel(R) Xeon(TM) L5520 CPUs @ 2.27GHz (8 physical cores, 16 logic cores with hyperthreading(R)), 32GB RAM.

In order to perform this analysis, we propose to take circuit oc_video_jpeg as a representative of the Opencores benchmark suite. The obtained results are illustrated in Figure 6.1. Notice that, although there are 32 tasks to be run, the speedup saturates from 7 threads on. This happens because each of these 32 tasks has a different runtime and ones are significantly faster than others. Thus, according to Amdahl's law, the speedup curve based on parallel computing will always be constrained to the workload of the sequential fractions (AMDAHL, 1967). In the presented case, there is a task that takes around 7 minutes to finish. Thus, when getting to this point, the speedup cannot be improved anymore though parallel execution. This trend was also observed in all circuits from the Opencores benchmark suite, where the speedup saturates around 7 threads on.

6.3 Analysis of the Effort Levels

In this section, we provide an analysis of the trade-off QoR versus speedup concerning the proposed effort levels. For this, we run one execution of each effort level for each circuit on the OpenCores benchmark circuits (PISTORIUS et al., 2007). This experiment was performed on the same personal computer of the

Figure 6.1: Obtained results when analyzing speedup for the parallel synthesis of circuit oc_video_jpeg at the high-effort level.



previous sections. For simplicity, the QoR is analyzed in terms of transistor count.

Table 6.2 summarizes the obtained results. We take the *high-effort* synthesis as reference and present its obtained number of transistors and runtime in seconds. For the medium- and low-effort levels, we present the ratio over the reference for QoR and the opposite ratio for speedup.

Table 6.2: Obtained results when analyzing QoR versus speedup for the proposed effort levels.

	high-effort		*medium-effort*		*low-effort*	
circuit	*#xtors*	*runtime*	*QoR*	*speedup*	*QoR*	*speedup*
oc_aquarius	123332	287	(1.00)	15x	(1.00)	57x
oc_cfft_1024x12	64490	22	(1.01)	6x	(1.01)	28x
oc_cordic_p2r	53816	21	(1.00)	7x	(1.00)	22x
oc_cordic_r2p	71626	22	(1.00)	6x	(1.00)	19x
oc_des_perf	140194	209	(1.00)	9x	(1.00)	34x
oc_ethernet	70556	39	(1.00)	12x	(1.00)	46x
oc_fpu	83972	138	(1.00)	10x	(1.00)	39x
oc_mem_ctrl	107864	85	(1.00)	11x	(1.00)	39x
oc_video_dct	230426	646	(1.01)	8x	(1.01)	29x
oc_video_jpeg	285566	1884	(1.00)	10x	(1.01)	41x
geomean			(1.00)	9x	(1.00)	34x

Notice that both *medium-* and *low-effort* levels are much faster than the *high-effort* synthesis. The *medium-effort* case is up to 15x faster (9x on average), whereas

the *low-effort* synthesis is up to 57x faster (34x on average), both at negligible QoR cost.

This shows that the proposed approach is very robust in terms of the analyzed parameters. As the work by Matos (2014) could vary in more than 6% by simply changing 1 of the 5 parameters, the work presented herein could keep the QoR almost unchanged even when changing all possible parameters.

6.4 Comparison with the State-of-the-Art

In order to compare the proposed approach with the state-of-the-art techniques, two sets of experiments were performed, which are presented in details in the next subsections. We compare the obtained results against both a commercial tool and with ABC tool (Berkeley Logic Synthesis and Verification Group, 2017), the state-of-the-art mapper from the academia. In the first experiments, we aim to evaluate the quality of the obtained results in terms of transistor count. In the second set of experiments, we show the results in terms of area, power and delay. For both of the experiments, the set of OpenCores circuits is initially submitted to the high-effort node count minimization script described in Section 5.1. Then, these optimized circuits are used as common input to both the proposed approach and the reference tools.

6.4.1 Synthesis for Transistor Count Minimization

For this set of experiments, we ran the proposed methods considering all possible scenarios of library composition and buffering addressed in this thesis. Regarding library composition, the simplest syntheses have only NAND2 ($\overline{*}$), NOR2 ($\overline{+}$), inverters and flip-flops; whereas we also consider syntheses including XOR2 ($\oplus$) and XNOR2 ($\otimes$). The mapping process was run under the low-effort level.

In order to compare with the reference tools, we propose to trick both ABC and the adopted commercial tool for them to optimize the overall transistor count while considering the same set of cells we are using in each synthesis. To do such a trick, we set the tools to run high-effort area-oriented synthesis and provided them with a developed library in which the cells' area and delay estimations are the

cells' transistor count. In ABC tool, we ran "*map -a*", "*amap*" and "*&nf -R 1000*" commands, and, then, took the best results out of these three methods.

Concerning the methodology to limit fanout, ABC results were achieved by re-buffering its unlimited fanout versions through its "*unbuffer/buffer*" commands. For the commercial tool, we used the constraint "*set_max_fanout*".

Table 6.3 presents the obtained results for the more complete case, with XORs and XNORs, with the fanout limited to 4. In this table, the first set of columns presents the obtained results from the proposed approach (*Simple Flow*). The second and third sets present a ratioed comparison of *Simple Flow* over *ABC* and the adopted commercial tool, respectively. When comparing against ABC, Simple Flow has up to 50% less inverters (17% on average) and up to 4% less transistors (2% on average). When comparing against the commercial tool, Simple Flow has up to 55% less inverters (38% on average) and up to 11% less transistors (5% on average).

Two specific results from Table 6.3 may claim the reader's attention. When comparing against ABC tool, the Simple Flow circuits *oc_aquarius* and *oc_mem_ctrl* have 22% and 39% more inverters than ABC's outputs, respectively. Still, both Simple Flow and ABC achieved similar results in terms of transistor count. This is mainly because these numbers are for the limited fanout case. As Simple Flow is managing to handle the fanout violations only with inverter trees, ABC is handling these cases both with buffering and with logic replication. Thus, notice that Simple Flow circuits also has less NANDs and NORs, which ABC replicated to limit fanout in some cases.

Table 6.3: Comparisons in terms of transistor count for circuits with limited fanout mapped with NAND2, NOR2, XOR2, XNOR2, inverters and flip-flops.

	Simple Flow					*Simple Flow / ABC*					*Simple Flow / Commercial Tool*				
circuit	$\overline{*}$, $\overline{+}$	$\oplus$, $\otimes$	*#ff*	*#invs*	*#xtors*	$\overline{*}$, $\overline{+}$	$\oplus$, $\otimes$	*#ff*	*#invs*	*#xtors*	$\overline{*}$, $\overline{+}$	$\oplus$, $\otimes$	*#ff*	*#invs*	*#xtors*
oc_aquarius	18322	238	1477	5983	128990	(0.99)	(0.90)	(1.00)	(1.22)	(1.00)	(0.95)	(0.70)	(1.00)	(0.73)	(0.94)
oc_cfft_1024x12	5941	855	1048	1175	64008	(1.02)	(0.93)	(1.00)	(0.50)	(0.96)	(1.08)	(0.73)	(1.01)	(0.56)	(0.96)
oc_cordic_p2r	6276	562	719	1818	54492	(1.00)	(0.96)	(1.00)	(0.97)	(0.99)	(0.99)	(0.77)	(1.00)	(0.72)	(0.95)
oc_cordic_r2p	7584	957	1015	1802	71930	(0.94)	(1.00)	(1.00)	(0.91)	(0.97)	(0.85)	(1.08)	(1.00)	(0.46)	(0.90)
oc_des_perf	16560	1338	1976	6778	148504	(1.02)	(0.91)	(1.00)	(0.56)	(0.97)	(0.92)	(0.70)	(1.00)	(0.45)	(0.89)
oc_ethernet	7572	85	1264	2172	70874	(0.94)	(0.64)	(0.99)	(0.95)	(0.96)	(0.95)	(0.43)	(1.00)	(0.68)	(0.95)
oc_fpu	12945	932	659	3433	86418	(1.00)	(0.97)	(1.00)	(0.77)	(0.98)	(1.07)	(0.91)	(1.00)	(0.67)	(1.00)
oc_mem_ctrl	12578	143	1825	3521	109884	(1.00)	(0.92)	(1.00)	(1.39)	(1.01)	(0.98)	(1.09)	(1.02)	(0.62)	(0.97)
oc_video_dct	24538	2467	3548	5147	232460	(1.00)	(0.97)	(1.00)	(0.84)	(0.99)	(1.10)	(0.64)	(1.00)	(0.60)	(0.96)
oc_video_jpeg	33711	2694	3972	8423	289846	(1.01)	(0.92)	(1.00)	(0.58)	(0.96)	(0.93)	(1.12)	(1.00)	(0.79)	(0.96)
geomean						(0.99)	(0.91)	(1.00)	(0.83)	(0.98)	(0.98)	(0.79)	(1.00)	(0.62)	(0.95)

In Table 6.4, we show the ratioed comparisons considering all the addressed cases of library composition and buffering for fanout limitation. Notice that the trend presented in Table 6.3 holds to all the synthesis versions. Simple Flow circuits have around 49% less inverters and 4% less transistors than the reference tools for the unlimited fanout cases. Additionally, we have around 16% (39%) less inverters and 2% (5%) less transistors for the limited fanout cases when compared with ABC (commercial tool).

Table 6.4: Comparisons in terms of transistor count for all the addressed cases.

	NANDs/NORs and INVs				NANDs/NORs, XORs/XNORs and INVs			
	Simple Flow / ABC		Simple Flow / Commercial Tool		Simple Flow / ABC		Simple Flow / Commercial Tool	
	unlimited	limited	unlimited	limited	unlimited	limited	unlimited	limited
nand/nor	0.99	0.98	0.95	0.95	1.00	0.99	1.00	0.98
xor/xnor	N/A	N/A	N/A	N/A	0.93	0.91	0.78	0.79
ff	1.00	1.00	1.03	1.03	0.97	1.00	1.03	1.00
invs	0.52	0.85	0.55	0.60	0.49	0.83	0.50	0.62
xtors	0.96	0.98	0.96	0.95	0.97	0.98	0.97	0.95

Now, we propose to analyze the circuits in Table 6.3 in terms of area, power and delay metrics. To do such an analysis, we used the obtained netlists as input into a commercial tool to report their area, power and delay estimations. All those estimations are considering the Nangate 15nm Open Cell Library (MARTINS et al., 2015).

Table 6.5 presents the obtained estimations out of the circuits previously introduced in Table 6.3. When comparing against ABC tool, Simple Flow circuits need up to 9% (2%) less area, consume up to 35% (13%) less power, and perform up to 94% (51%) faster (on average). When comparing against the commercial tool, Simple Flow circuits need up to 12% (5%) less area, consume up to 35% (5%) less power, and perform up to 97% (53%) faster (on average).

Table 6.5: Comparisons in terms of area, power and delay for circuits with limited fanout mapped with NAND2, NOR2, XOR2, XNOR2, inverters and flip-flops.

	Simple Flow			*Simple Flow / ABC*						*Simple Flow / Commercial Tool*					
circuit	*area* (um^2)	*power* (mW)	*delay* (ps)	*area* (um^2)		*power* (mW)		*delay* (ps)		*area* (um^2)		*power* (mW)		*delay* (ps)	
oc_aquarius	6477	16.72	853	6913	(0.94)	25.66	(0.65)	1041	(0.82)	6606	(0.98)	18.02	(0.93)	991	(0.86)
oc_cfft_1024x12	3059	18.12	303	3296	(0.93)	19.88	(0.91)	346	(0.88)	3238	(0.94)	18.63	(0.97)	391	(0.77)
oc_cordic_p2r	2669	11.85	278	2625	(1.02)	11.23	(1.06)	364	(0.76)	2866	(0.93)	11.85	(1.00)	466	(0.60)
oc_cordic_r2p	3477	11.85	332	3420	(1.02)	13.98	(0.85)	856	(0.39)	3969	(0.88)	10.22	(1.16)	683	(0.49)
oc_des_perf	7372	79.44	83	7189	(1.03)	82.12	(0.97)	96	(0.86)	7649	(0.96)	81.22	(0.98)	135	(0.61)
oc_ethernet	3436	26.41	217	3673	(0.94)	31.84	(0.83)	426	(0.51)	3715	(0.92)	29.89	(0.88)	286	(0.76)
oc_fpu	4306	1.97	11523	4426	(0.97)	2.10	(0.94)	13140	(0.88)	4115	(1.05)	2.09	(0.94)	12917	(0.89)
oc_mem_ctrl	5388	22.44	175	5192	(1.04)	24.05	(0.93)	407	(0.43)	5498	(0.98)	25.44	(0.88)	446	(0.39)
oc_video_dct	11209	19.17	1164	12268	(0.91)	24.57	(0.78)	3355	(0.35)	11931	(0.94)	29.57	(0.65)	2142	(0.54)
oc_video_jpeg	14138	14.3	234	14407	(0.98)	15.90	(0.90)	3774	(0.06)	15210	(0.93)	11.29	(1.27)	7849	(0.03)
geomean					(0.98)		(0.87)		(0.49)		(0.95)		(0.95)		(0.47)

Some specific results from Table 6.3 may also claim the reader's attention. When comparing against the commercial tool, the Simple Flow circuits *oc_cordic_p2r* and *oc_video_jpeg consume* 16% and 27% more power than commercial tool's outputs, respectively. This is mainly because the Simple Flow circuits operate in much higher frequency than the commercial tool circuits, what makes them consume more dynamic power. Simple Flow circuit *oc_video_jpeg* is also more than 94% faster than both the reference tools, mainly because neither of them effectively handled the fanout violations on the critical path.

In Table 6.5, we show the ratioed comparisons considering all the addressed cases of library composition and buffering for fanout limitation presented in Table 6.5. Simple Flow circuits need around $5 \sim 10\%$ less area than the reference tools for all considered cases. Also, we consume about 15% less power than ABC and about 5% less power than the commercial tool. Regarding delay, our circuits are more than 50% (15%) faster than ABC's circuits for the limited fanout case (unlimited fanout case) and around 50% (20%) faster than the circuits from the commercial tool.

Table 6.6: Comparisons in terms of area, power and delay for all the addressed cases.

	NANDs/NORs and INVs				NANDs/NORs, XORs/XNORs and INVs			
	Simple Flow / ABC		Simple Flow / Commercial Tool		Simple Flow / ABC		Simple Flow / Commercial Tool	
	unlimited	limited	unlimited	limited	unlimited	limited	unlimited	limited
area	0.91	0.97	0.88	0.95	0.92	0.98	0.90	0.95
power	0.86	0.85	0.96	0.95	0.87	0.87	0.95	0.95
delay	0.84	0.49	0.84	0.46	0.84	0.49	0.81	0.47

6.4.2 Timing-Constrained, Area/Power-Optimized Synthesis

One can say that it might not be fair to trick standard synthesis tools for them to optimize a given cost function different from the ones these tools were designed to optimize. Thus, it would be interesting to compare the proposed approach with a state-of-the-art synthesis tool when it is optimizing the cost functions it was designed to optimize. For that, we propose to synthesize the circuits in a commercial tool under the most common synthesis scenario for VLSI designs: a timing-constrained, area/power-optimized synthesis.

For this experiment, the critical-path delays from Simple Flow circuits in Ta-

ble 6.5 were taken as target delays for the timing-constrained, area/power-optimized syntheses in the commercial tool. In this case, Nangate 15nm Open Cell Library (MARTINS et al., 2015) was used for the synthesis.

Table 6.7 presents the obtained results. Notice that Simple Flow numbers are exactly the same as the ones presented in Tables 6.3 and 6.5. In contrast, now under the timing-constrained, area/power-optimized synthesis scenario, the commercial tool needed more area and consumed more power in order to meet the target delay. Thus, Simple Flow circuits have up to 16% less transistors (10% on average), need up to 18% less area (12% on average), and consume up to 12% less power (7% on average) when compared to a commercial tool in this scenario.

Table 6.7: Comparison with a commercial tool under a timing-constrained, area/power-optimized synthesis.

circuit	target *delay (ps)*	*Simple Flow*			*Simple Flow / Commercial Tool*		
		#xtors	*area (um^2)*	*power (mW)*	*#xtors (ratio)*	*area (ratio)*	*power (ratio)*
oc_aquarius	853	128990	6477	16.72	(0.90)	(0.89)	(0.97)
oc_cfft	303	64008	3059	18.12	(0.92)	(0.90)	(0.96)
oc_cordic_r2p	278	54492	2669	11.85	(0.88)	(0.87)	(0.89)
oc_cordic_p2r	332	71930	3477	11.85	(0.85)	(0.82)	(0.87)
oc_des_perf	83	148504	7372	79.44	(0.84)	(0.82)	(0.97)
oc_ethernet	217	70874	3436	26.41	(0.91)	(0.90)	(0.96)
oc_fpu	11523	86418	4306	1.97	(0.90)	(0.87)	(0.88)
oc_mem_ctrl	175	109884	5388	22.44	(0.95)	(0.95)	(0.98)
oc_video_dct	259	232460	11209	19.17	(0.92)	(0.91)	(0.95)
oc_video_jpeg	234	289846	14138	14.3	(0.93)	(0.92)	(0.93)
geomean					(0.90)	(0.88)	(0.93)

7 CONCLUSION AND FUTURE WORKS

This thesis introduced a set of algorithms for efficiently mapping VLSI circuits based on simple cells. We showed that efficient VLSI circuit implementations based on simple cells can be realized from a minimal implementation on the number of logic elements combined with a minimized number of inverters in between the logic cells.

A first contribution presented in this thesis is a high-effort AIG node count minimization approach. We demonstrated that, as the methods based on AIGs are highly dependent on the circuit given as inputs, the quality of results can be even more improved if the applied methods are repeatedly iterated and interleaved with each other up to saturation. Experimental results showed that, by applying the proposed techniques, one can obtain AIGs with up 10% less nodes (5% on average) when compared to state-of-the-art AIG node count minimization approaches.

From fairly optimized AIGs, we are able to generate optimized XAIGs, and this comprises a second contribution of this thesis. By doing this, we can consider XOR nodes while generating polarity graphs and minimizing the inverter count under a phase assignment optimization model. The proposed approach explores the parity property of XORs and XNORs and, as both of the functions can be implemented with the number of transistors, we claim that they are polarity don't care nodes in terms of transistor count. Then, we propose to use a graph coloring procedure to bring the inverters to the XORs/XNORs inputs and outputs such that we can get rid with them during the mapping process. Similar to the XOR case, we showed that two-output flip-flops can also be thought as polarity don't care nodes and be used to further optimize the inverter count.

In the proposed flow, high-effort algorithms for AIG node count minimization tend to generate cells with unfeasible fanout in the final circuit. In order to address this issue, we propose an area-oriented, level-aware buffering algorithm based on inverter trees. The proposed algorithm is area-oriented in the sense that we limit the cells' fanout inserting an inverter tree while minimizing the number of inverters in the tree. It is also level-aware because we consider the logic depth of the consumer cells when assigning them to the tree. This simple (yet effective) change can further optimize the circuit performance.

Four secondary contributions were also presented in this thesis, two of them

related to algorithm speedup and other two related with QoR improvements. The work presented herein extends the work by Matos (2014), which can be time consuming while generating 32 circuits per synthesis based on 5 binary parameters.

In order to speed up the synthesis process, we first propose a coarse-grained, procedure-level parallel approach. So, instead of running 1 serial execution of 32 tasks, we propose to parallelize the execution of these 32 tasks. The obtained results showed that the synthesis process can be sped up around 2.5x with this approach. A second speedup process we proposed is based on different effort levels to trade off QoR and runtime. For this, we performed a statistical analysis on the 5 binary parameters and proposed 3 effort levels. The obtained results showed that the effort-level approach can be up to 57x faster (34x on average) at negligible QoR cost (less than 1%). This also shows the robustness of the proposed approach, since the work by Matos (2014), which bases this work, can vary in more than 6% by simply changing 1 of the 5 parameters.

The two secondary contributions for QoR improvement are based on exploring sequential cells. We first propose to merge logically equivalent flip-flops in order to improve area, as the performance be further recovered with buffering. We also propose to use both the outputs of flip-flops when the output signal is required in both polarities.

The proposed approach was validated by mapping benchmark circuits using two-input NANDs and NORs, inverters and flip-flops. Two-input XORs and XNORs were also considered. The obtained results were evaluated in terms of different cost functions, such as inverter count, transistor count, power, area and performance. When comparing the obtained results against state-of-the-art algorithms for technology mapping, the proposed approach shows its usefulness. When comparing with academic and commercial approaches, we are able to simultaneously reduce the average number of inverters, transistors, area, power dissipation and delay up to 48%, 5%, 5%, 5%, and 53%, respectively.

There is still much work to be carried on. A coarse-grained, program-level parallel approach can be used to speed up the proposed AIG node count minimization approach. Instead of running 1 serial execution of 5 minimization approaches, all of them can be ran in parallel and speed up this optimization step. Also, a standard cell with different drive strengths could also be used in the case of timing-constrained synthesis, so that a gate sizing step would handle the necessary trade-off.

Finally, for advanced technology, the wire delay is becoming more important than cell delay (especially for long wires) and the circuit performance can significantly change after place and route. Thus, we aim to apply such a simple-cell-based approach under a physically-aware logic synthesis environment. Experiments on this way were presented already (MATOS; REIS, 2015; MATOS et al., 2015b) and we also proposed an innovative flow to bring technology information (specially related with place and route) to the early steps of logic synthesis (REIS; MATOS, 2017).

Nonetheless, even if the presented research opened the way for several future works, the current results have shown the computational and practical viability of the methods presented herein. The proposed algorithms have been proved useful for efficiently mapping VLSI circuits based on simple cells. The proposed algorithms can bring benefits for most of the VLSI applications constrained to simple cells referred in this work.

REFERENCES

ABOUZEID, P.; LEVEUGLE, R.; SAUCIER, G.; JAMIER, R. Logic synthesis for automatic layout. In: PROC. OF EURO ASIC. **Proceedings...** [S.l.: s.n.], 1992. p. 146–151.

AKERS, S. B. Binary Decision Diagrams. **IEEE Trans. on Computer**, IEEE, v. 27, n. 6, p. 509–516, 1978.

ALIOTO, M. Ultra-low power vlsi circuit design demystified and explained: A tutorial. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 59, n. 1, p. 3–29, 2012.

ALPERT, C. J.; DEVGAN, A.; QUAY, S. T. Buffer insertion with accurate gate and interconnect delay computation. In: PROC. OF DESIGN AUTOMATION CONFERENCE (DAC). **Proceedings...** [S.l.: s.n.], 1999.

AMARU, L.; GAILLARDON, P.-E.; CHATTOPADHYAY, A.; MICHELI, G. D. A sound and complete axiomatization of majority-n logic. **IEEE Transactions on Computers**, IEEE, v. 65, n. 9, p. 2889–2895, 2016.

AMARÚ, L.; GAILLARDON, P.-E.; MICHELI, G. D. Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization. In: ACM. PROCEEDINGS OF THE 51ST ANNUAL DESIGN AUTOMATION CONFERENCE. **Proceedings...** [S.l.], 2014. p. 1–6.

AMARÚ, L.; GAILLARDON, P.-E.; MITRA, S.; MICHELI, G. D. New logic synthesis as nanotechnology enabler. **Proceedings of the IEEE**, IEEE, v. 103, n. 11, p. 2168–2195, 2015.

AMARU, L. G. **New Data Structures and Algorithms for Logic Synthesis and Verification**. [S.l.]: Springer, 2017.

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In: ACM. PROCEEDINGS OF THE APRIL 18-20, 1967, SPRING JOINT COMPUTER CONFERENCE. **Proceedings...** [S.l.], 1967. p. 483–485.

BACKES, M.; MATOS, J.; RIBAS, R.; REIS, A. Polarity-oriented aig rewriting for xor/xnor. In: SOUTH SYMPOSIUM ON MICROELECTRONICS. **Proceedings...** [S.l.: s.n.], 2014.

BACKES, M.; MATOS, J. M.; RIBAS, R.; REIS, A. Reviewing aig equivalence checking approaches. In: SOUTH SYMPOSIUM ON MICROELECTRONICS. **Proceedings...** [S.l.: s.n.], 2015.

BARTLETT, K.; BOSTICK, D.; HACHTEL, G.; JACOBY, R.; LIGHTNER, M.; MOCEYUNAS, P.; MORRISON, C.; RAVENSCROFT, D. Bold: A multiple-level logic optimization system. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN. **Proceedings...** [S.l.: s.n.], 1987.

BERKELAAR, M.; JESS, J. Technology mapping for standard-cell generators. In: PROC. OF INT'L CONF. ON COMPUTER-AIDED DESIGN (ICCAD). **Proceedings...** [S.l.: s.n.], 1988. p. 470–473.

Berkeley Logic Synthesis and Verification Group. Abc: A system for sequential synthesis and verification. In: . [s.n.], 2017. Release 20170710. Available from Internet: <http://www.eecs.berkeley.edu/~alanmi/abc/>.

BRAYTON, R.; MISHCHENKO, A.; CHATTERJEE, S.; CIESIELSKI, M. An integrated technology mapping environment. In: PROC. OF INT'L WORKSHOP ON LOGIC AND SYNTHESIS (IWLS). **Proceedings...** [S.l.: s.n.], 2005.

BRAYTON, R. K.; MCMULLEN, C. The decomposition and factorization of boolean expressions. In: PROC. OF INT'L SYMP. ON CIRCUITS AND SYSTEMS (ISCAS). **Proceedings...** [S.l.: s.n.], 1982.

BRAYTON, R. K.; RUDELL, R.; SANGIOVANNI-VINCENTELLI, A.; WANG, A. R. MIS: A multiple-level logic optimization system. **IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 6, n. 6, p. 1062–1081, 1987.

BRYAN, D. The ISCAS'85 benchmark circuits and netlist format. **North Carolina State University**, 1985.

BRYANT, R. E. Graph-based algorithms for Boolean functions manipulation. **IEEE Trans. on Computer**, IEEE, v. 35, n. 8, p. 677–691, 1986.

BUTZEN, P. F.; BEM, V. D.; REIS, A. I.; RIBAS, R. P. Design of cmos logic gates with enhanced robustness against aging degradation. **Microelectronics Reliability**, v. 52, n. 9, 2012.

BUTZEN, P. F.; ROSA, L. S. da; FILHO, E. J. C.; REIS, A. I.; RIBAS, R. P. Standby power consumption estimation by interacting leakage current mechanisms in nanoscaled cmos digital circuits. **Microelectronics Journal**, v. 41, n. 4, 2010.

CAO, W.; XUE, J. Recent progress in organic photovoltaics: device architecture and optical design. **Energy & Environmental Science**, Royal Society of Chemistry, v. 7, n. 7, p. 2123–2144, 2014.

CHATTERJEE, S.; MISHCHENKO, A.; BRAYTON, R. K.; WANG, X.; KAM, T. Reducing structural bias in technology mapping. **IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,**, IEEE, v. 25, n. 12, p. 2894–2903, 2006.

CHAVA, B.; RIO, D.; SHERAZI, Y.; TRIVKOVIC, D.; GILLIJNS, W.; DEBACKER, P.; RAGHAVAN, P.; ELSAID, A.; DUSA, M.; MERCHA, A. et al. Standard cell design in n7: Euv vs. immersion. In: SPIE ADVANCED LITHOGRAPHY. **Proceedings...** [S.l.: s.n.], 2015.

CORREIA, V.; REIS, A. I. Advanced technology mapping for standard-cell generators. In: PROC. OF SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN (SBCCI). **Proceedings...** [S.l.: s.n.], 2004.

CORREIA, V. P.; REIS, A. I. Classifying n-input boolean functions. In: PROC. OF INT'L WORKSHOP ON LOGIC AND SYNTHESIS (IWLS). **Proceedings...** [S.l.: s.n.], 2001.

CORTADELLA, J. Timing-driven logic bi-decomposition. **IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 22, n. 6, p. 675–685, 2003.

DARRINGER, J.; JOYNER, W.; BERMAN, C.; TREVILLYAN, L. Logic Synthesis Through Local Transformations. **IBM Journal of Research and Development**, v. 25, n. 4, 1981.

DETJENS, E.; GANNOT, G.; RUDELL, R.; SANGIOVANNI-VINCENTELLI, A.; WANG, A. Technology mapping in mis. In: PROC. OF IEEE INT'L CONF. ON COMPUTER AIDED DESIGN (ICCAD). **Proceedings...** [S.l.: s.n.], 1987. p. 116–119.

DOOD, P. de; LEE, B.; ALBERS, D. **Optimization of circuit designs using a continuous spectrum of library cells**. 2006. US Patent 7107551.

DRESLINSKI, R. G.; WIECKOWSKI, M.; BLAAUW, D.; SYLVESTER, D.; MUDGE, T. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. **Proceedings of the IEEE**, IEEE, v. 98, n. 2, p. 253–266, 2010.

FAN, C.-L.; CHEN, Y.-C.; YANG, C.-C.; TSAI, Y.-K.; HUANG, B.-R. Novel ltps-tft pixel circuit with oled luminance compensation for 3d amoled displays. **Journal of Display Technology**, IEEE, v. 12, n. 5, p. 425–428, 2016.

FIGUEIRÓ, T.; RIBAS, R. P.; REIS, A. I. Constructive aig optimization considering input weights. In: QUALITY ELECTRONIC DESIGN (ISQED), 2011 12TH INTERNATIONAL SYMPOSIUM ON. **Proceedings...** [S.l.: s.n.], 2011.

FUKETA, H.; YASUFUKU, T.; IIDA, S.; TAKAMIYA, M.; NOMURA, M.; SHINOHARA, H.; SAKURAI, T. Device circuit interactions in extremely low voltage CMOS designs. In: PROC. OF INT'L ELECTRONIC DEVICES MEETING (IEDM). **Proceedings...** [S.l.: s.n.], 2011.

GAVRILOV, S.; GLEBOV, A.; PULLELA, S.; MOORE, S. C.; DHARCHOUDHURY, A.; PANDA, R.; VIJAYAN, G.; BLAAUW, D. T. Library-less synthesis for static CMOS combinational logic circuits. In: PROC. OF ICCAD. **Proceedings...** [S.l.: s.n.], 1997.

GEREZ, S. **Algorithms for VLSI Design Automation**. 1. ed. [S.l.]: John Wiley & Sons Ltd, 1999.

GREGORY, D.; GEUS, A. D.; BARTLETT, K.; HACHTEL, G. Socrates: A system for automatically synthesizing and optimizing combinational logic. In: DESIGN AUTOMATION CONF. (DAC). **Proceedings...** [S.l.: s.n.], 1986. p. 79–85.

HACHTEL, G.; SOMENZI, F. **Logic Synthesis and Verification Algorithms**. 1. ed. [S.l.]: Kluwer Academic Publishers, 1996.

HAMBSCH, M.; REUTER, K.; STANEL, M.; SCHMIDT, G.; KEMPA, H.; FUGMANN, U.; HAHN, U.; HUBLER, A. Uniformity of fully gravure printed organic field-effect transistors. **Materials Science and Engineering: B**, Elsevier, v. 170, n. 1, p. 93–98, 2010.

HASSOUN, S.; SASAO, T. **Logic synthesis and verification**. [S.l.]: Springer Science & Business Media, 2012.

HELLERMAN, L. A Catalog of Three-Variable Or-Invert and And-Invert Logical Circuits. **IEEE Trans. on Electron. Computers**, EC-12, n. 3, 1963.

HSU, M.-K.; KATTA, N.; LIN, H. Y.-H.; LIN, K. T.-H.; TAM, K. H.; WANG, K. C.-H. Design and manufacturing process co-optimization in nano-technology. In: IEEE PRESS. PROCEEDINGS OF THE 2014 IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN. **Proceedings...** [S.l.], 2014. p. 574–581.

HULZINK, J.; KONIJNENBURG, M.; ASHOUEI, M.; BREESCHOTEN, A.; BERSET, T.; HUISKEN, J.; STUYT, J.; GROOT, H. de; BARAT, F.; DAVID, J. et al. An ultra low energy biomedical signal processing system operating at near-threshold. **IEEE transactions on biomedical circuits and systems**, IEEE, v. 5, n. 6, p. 546–554, 2011.

HWANG, K. **Advanced Computer Architecture: Parallelism, Scalability, Programmability, Fourteenth Reprint**. [S.l.]: Tata McGraw-Hill Edition, ISBN-0-07-053070-X-2007, 2007.

ISMAIL, Y. I.; FRIEDMAN, E. G. Optimum repeater insertion based on a cmos delay model for on-chip rlc interconnect. In: PROC. OF INTERNATIONAL ASIC CONFERENCE. **Proceedings...** [S.l.: s.n.], 1998.

ITRS. THE INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS: Beyond CMOS. In: ITRS 2.0. **Proceedings...** [s.n.], 2015. Available from Internet: <http://www.semiconductors.org>.

ITRS. THE INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS: EXECUTIVE REPORT. In: ITRS 2.0. **Proceedings...** [s.n.], 2015. Available from Internet: <http://www.semiconductors.org>.

JAIN, A.; BRYANT, R. Inverter minimization in multi-level logic networks. In: PROC. OF ICCAD. **Proceedings...** [S.l.: s.n.], 1993.

JUNIOR, L. S. da R.; MARQUES, F. S.; CARDOSO, T. M.; RIBAS, R. P.; SAPATNEKAR, S. S.; REIS, A. I. Fast disjoint transistor networks from bdds. In: PROC. OF ANNUAL CONFERENCE ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN. **Proceedings...** [S.l.: s.n.], 2006.

JUNIOR, L. S. da R.; REIS, A. I.; RIBAS, R. P.; MARQUES, F. d. S.; SCHNEIDER, F. R. A comparative study of cmos gates with minimum transistor stacks. In: PROC. OF ANNUAL CONFERENCE ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN. **Proceedings...** [S.l.: s.n.], 2007.

JUNIOR, R. et al. Automatic generation and evaluation of transistor networks in different logic styles. 2008.

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

KAHNG, A.; LIENIG, J.; MARKOV, I.; HU, J. **VLSI Physical Design: From Graph Partitioning to Timing Closure**. [S.l.]: Springer Netherlands, 2011.

KAUL, H.; ANDERS, M.; HSU, S.; AGARWAL, A.; KRISHNAMURTHY, R.; BORKAR, S. Near-threshold voltage (NTV) design: opportunities and challenges. In: PROC. OF DESIGN AUTOMATION CONFERENCE (DAC). **Proceedings...** [S.l.: s.n.], 2012.

KEUTZER, K. Dagon: technology binding and local optimization by dag matching. In: DESIGN AUTOMATION CONF. (DAC). **Proceedings...** [S.l.: s.n.], 1987. p. 341–347.

KIM, S.; COOK, B.; LE, T.; COOPER, J.; LEE, H.; LAKAFOSIS, V.; VYAS, R.; MORO, R.; BOZZI, M.; GEORGIADIS, A. et al. Inkjet-printed antennas, sensors and circuits on paper substrate. **Microwaves, Antennas & Propagation, IET**, IET, v. 7, n. 10, p. 858–868, 2013.

KUKIMOTO, Y.; BRAYTON, R. K.; SAWKAR, P. Delay-optimal technology mapping by dag covering. In: DESIGN AUTOMATION CONF. (DAC). **Proceedings...** [S.l.: s.n.], 1998. p. 348–351.

KWONG, J.; CHANDRAKASAN, A. P. Variation-driven device sizing for minimum energy sub-threshold circuits. In: PROC. OF INT'L SYMP. ON LOW POWER ELECTRONICS AND DESIGN (ISLPED). **Proceedings...** [S.l.: s.n.], 2006.

KWONG, J.; RAMADASS, Y. K.; VERMA, N.; CHANDRAKASAN, A. P. A 65 nm sub-microcontroller with integrated SRAM and switched capacitor DC-DC converter. **IEEE Journal of Solid-State Circuits**, IEEE, v. 44, n. 1, 2009.

LEE, C. Y. Binary Decision Programs. **Bell System Technical Journal**, American Telephone and Telegraph Company, v. 38, n. 4, p. 985–999, 1959.

LEWIS, J.; YANNAKAKIS, M. The node-deletion problem for hereditary properties is NP-complete. **Journal of Computer and System Sciences**, v. 20, n. 2, 1980.

LIEM, C.; LEFEBVRE, M. A constructive matching algorithm for cell generator based technology mapping. In: PROC. OF INT'L SYMP. ON CIRCUITS AND SYSTEMS (ISCAS). **Proceedings...** [S.l.: s.n.], 1992. v. 6, p. 2965–2968.

LIU, B.; ASHOUEI, M.; HUISKEN, J.; GYVEZ, J. P. D. Standard cell sizing for subthreshold operation. In: PROC. OF DESIGN AUTOMATION CONFERENCE (DAC). **Proceedings...** [S.l.: s.n.], 2012.

LIU, X.; ZHOU, J.; LIAO, X.; WANG, C.; LUO, J.; MADIHIAN, M.; JE, M. Ultra-low-energy near-threshold biomedical signal processor for versatile wireless health monitoring. In: IEEE. SOLID STATE CIRCUITS CONFERENCE (A-SSCC), 2012 IEEE ASIAN. **Proceedings...** [S.l.], 2012. p. 381–384.

LLAMAS, M.; MASHAYEKHI, M.; CARRABINA, J.; MATOS, J.; REIS, A. Optimization on cell-library design for digital Application Specific Printed Electronics Circuits. In: PROC. OF INT'L WORKSHOP ON POWER AND TIMING MODELING, OPTIMIZATION AND SIMULATION (PATMOS). **Proceedings...** [S.l.: s.n.], 2014.

LLAMAS, M.; MASHAYEKHI, M.; CARRABINA, J. et al. Top-down design flow for application specific printed electronics circuits (ASPECs). In: PROC. OF CONF. ON DESIGN OF CIRCUITS AND INTEGRATED CIRCUITS (DCIS). **Proceedings...** [S.l.: s.n.], 2014.

LLAMAS, M.; MATOS, J.; MASHAYEKHI, M.; REIS, A.; CARRABINA, J. Technology mapping tools for building optimal circuits. In: PROC. OF INT'L EXHIBITION AND CONF. FOR THE PRINTED ELECTRONICS INDUSTRY (LOPEC). **Proceedings...** [S.l.: s.n.], 2015.

LogiCS Research Lab. Simple Flow. In: . [s.n.], 2017. Release 20170710. Available from Internet: <http://www.inf.ufrgs.br/logics/downloads/>.

LOTZE, N.; MANOLI, Y. A 62 mV 0.13 m CMOS Standard-Cell-Based Design Technique Using Schmitt-Trigger Logic. **IEEE Journal of Solid-State Circuits**, IEEE, v. 47, n. 1, 2012.

LUTKEMEIER, S.; JUNGEBLUT, T.; BERGE, H. K. O.; AUNET, S.; PORRMANN, M.; RUCKERT, U. A 65 nm 32 b subthreshold processor with 9T multi-Vt SRAM and adaptive supply voltage control. **IEEE Journal of Solid-State Circuits**, IEEE, v. 48, n. 1, 2013.

MACHADO, L.; MARTINS, M.; CALLEGARO, V.; RIBAS, R. P.; REIS, A. I. KL-cut based digital circuit remapping. In: NORCHIP. **Proceedings...** [S.l.: s.n.], 2012.

MAILHOT, F.; MICHELI, G. D. Algorithms for technology mapping based on binary decision diagrams and on boolean operations. **IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,**, IEEE, v. 12, n. 5, p. 599–620, 1993.

MARINOV, V. R. Embedded flexible hybrid electronics for the internet of things. In: INTERNATIONAL MICROELECTRONICS ASSEMBLY AND PACKAGING SOCIETY. INTERNATIONAL SYMPOSIUM ON MICROELECTRONICS. **Proceedings...** [S.l.], 2015. v. 2015, n. 1, p. 000006–000013.

MARKOVIC, D.; WANG, C. C.; ALARCON, L. P.; LIU, T.-T.; RABAEY, J. M. Ultralow-power design in near-threshold region. **Proceedings of the IEEE**, IEEE, v. 98, n. 2, p. 237–252, 2010.

MARQUES, F.; ROSA, L.; RIBAS, R.; SAPATNEKAR, S.; REIS, A. DAG based library-free technology mapping. In: PROC. OF GLSVLSI. **Proceedings...** [S.l.: s.n.], 2007.

MARQUES, F. S.; JR, L. S. R.; RIBAS, R. P.; SAPATNEKAR, S. S.; REIS, A. I. Dag based library-free technology mapping. In: PROC. OF GLSVLSI. **Proceedings...** [S.l.: s.n.], 2007.

MARTINELLO, O.; MARQUES, F. S.; RIBAS, R. P.; REIS, A. I. Kl-cuts: a new approach for logic synthesis targeting multiple output blocks. In: DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION (DATE), 2010. **Proceedings...** [S.l.: s.n.], 2010.

MARTINS, M.; MATOS, J. M.; RIBAS, R. P.; REIS, A.; SCHLINKER, G.; RECH, L.; MICHELSEN, J. Open cell library in 15nm freepdk technology. In: PROC. OF INT'L SYMP. ON PHYSICAL DESIGN (ISPD). **Proceedings...** [S.l.: s.n.], 2015.

MARTINS, M. G.; CALLEGARO, V.; MACHADO, L.; RIBAS, R. P.; REIS, A. I. Functional composition paradigm and applications. In: INTERNATIONAL WORKSHOP ON LOGIC AND SYNTHESIS (IWLS'2012). BERKELEY, CA. **Proceedings...** [S.l.: s.n.], 2012.

MARTINS, M. G.; CALLEGARO, V.; RIBAS, R. P.; REIS, A. I. Efficient method to compute minimum decision chains of boolean functions. In: ACM. PROCEEDINGS OF THE 21ST EDITION OF THE GREAT LAKES SYMPOSIUM ON GREAT LAKES SYMPOSIUM ON VLSI. **Proceedings...** [S.l.], 2011. p. 419–422.

MARTINS, M. G.; RIBAS, R. P.; REIS, A. I. Functional composition: A new paradigm for performing logic synthesis. In: PROC. OF INT'L SYMPOSIUM ON QUALITY ELECTRONIC DESIGN (ISQED). **Proceedings...** [S.l.: s.n.], 2012.

MARTINS, M. G.; ROSA, L.; RASMUSSEN, A. B.; RIBAS, R. P.; REIS, A. I. Boolean factoring with multi-objective goals. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN (ICCD). **Proceedings...** [S.l.: s.n.], 2010.

MASGONTY, J.; CSERVENY, S.; ARM, C.; PFISTER, P.; PIGUET, C. Low-power low-voltage standard cell libraries with a limited number of cells. In: PATMOS. **Proceedings...** [S.l.: s.n.], 2001.

MASHAYEKHI, M.; LLAMAS, M.; CARRABINA, J. et al. Development of a standard cell library and ASPEC design flow for Organic Thin Film Transistor technology. In: PROC. OF CONF. ON DESIGN OF CIRCUITS AND INTEGRATED CIRCUITS (DCIS). **Proceedings...** [S.l.: s.n.], 2014.

MATOS, J.; REIS, A. Placed AIGs as a Logical-Physical Data-Structure for VLSI Circuit Design. In: PROC. OF INT'L WORKSHOP ON LOGIC AND SYNTHESIS (IWLS). **Proceedings...** [S.l.: s.n.], 2015.

MATOS, J. M. **Graph-Based Algorithms for Transistor Count Minimization in VLSI Circuit EDA Tools**. Dissertation (Master's degree on Microelectronics) — Universidade Federal do Rio Grande do Sul, 2014.

MATOS, J. M.; BACKES, M. H.; RITT, M.; RIBAS, R. P.; REIS, A. Mapping circuits with simple cells from xor-and-inverter graphs. In: PROC. OF INT'L WORKSHOP ON LOGIC AND SYNTHESIS (IWLS). **Proceedings...** [S.l.: s.n.], 2015.

MATOS, J. M.; NEUTZLING, A.; RIBAS, R. P.; REIS, A. A benchmark suite to jointly consider logic synthesis and physical design. In: PROCEEDINGS OF THE 2015 SYMPOSIUM ON INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN. **Proceedings...** [S.l.: s.n.], 2015.

MATOS, J. M.; RITT, M.; RIBAS, R.; REIS, A. Deriving Reduced Transistor Count Circuits from AIGs. In: PROC. OF SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI). **Proceedings...** [S.l.: s.n.], 2014.

MICHELI, G. de. Cell-based Logic Optimizations. In: Architecture Design and Validation Methods. **Proceedings...** 1. ed. [S.l.]: Springer-Verlag, 2000.

MICHELI, G. de. **Synthesis and optimization of digital circuits**. [S.l.]: Tata McGraw-Hill Education, 2003.

MISHCHENKO, A.; CHATTERJEE, S.; BRAYTON, R. Integrating logic synthesis, technology mapping, and retiming. In: PROC. OF INT'L WORKSHOP ON LOGIC AND SYNTHESIS (IWLS). **Proceedings...** [S.l.: s.n.], 2005.

MISHCHENKO, A.; CHATTERJEE, S.; BRAYTON, R. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In: PROC. OF DESIGN AUTOMATION CONF. (DAC). **Proceedings...** [S.l.: s.n.], 2006.

MOONEN, P. F.; YAKIMETS, I.; HUSKENS, J. Fabrication of transistors on flexible substrates: from mass-printing to high-resolution alternative lithography strategies. **Advanced materials**, Wiley Online Library, v. 24, n. 41, p. 5526–5541, 2012.

MOORE, G. Cramming More Components onto Integrated Circuits. **Electronics**, 1965.

NALAMALPU, A.; BURLESON, W. Repeater insertion in deep sub-micron cmos: ramp-based analytical model and placement sensitivity analysis. In: PROC. OF INTERNATIONAL SYMPOSIUM ON CIRCUIS AND SYSTEMS (ISCAS). **Proceedings...** [S.l.: s.n.], 2000.

NEUTZLING, A.; MATOS, J. M.; REIS, A. I.; RIBAS, R. P.; MISHCHENKO, A. Threshold logic synthesis based on cut pruning. In: COMPUTER-AIDED DESIGN (ICCAD), 2015 IEEE/ACM INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2015.

PAGLIARINI, S.; MARTINS, M.; PILEGGI, L. Virtual characterization for exhaustive dfm evaluation of logic cell libraries. In: IEEE. QUALITY ELECTRONIC DESIGN (ISQED), 2017 18TH INTERNATIONAL SYMPOSIUM ON. **Proceedings...** [S.l.], 2017. p. 93–98.

PAN, D. Z.; LIEBMANN, L.; YU, B.; XU, X.; LIN, Y. Pushing multiple patterning in sub-10nm: are we ready? In: ACM. PROCEEDINGS OF THE 52ND ANNUAL DESIGN AUTOMATION CONFERENCE. **Proceedings...** [S.l.], 2015. p. 197.

PAUL, S.; HONKOTE, V.; KIM, R.; MAJUMDER, T.; ASERON, P.; GROSSNICKLE, V.; SANKMAN, R.; MALLIK, D.; JAIN, S.; VANGAL, S. et al. An energy harvesting wireless sensor node for iot systems featuring a near-threshold voltage ia-32 microcontroller in 14nm tri-gate cmos. In: IEEE. VLSI CIRCUITS (VLSI-CIRCUITS), 2016 IEEE SYMPOSIUM ON. **Proceedings...** [S.l.], 2016. p. 1–2.

PAUL, S.; HONKOTE, V.; KIM, R. G.; MAJUMDER, T.; ASERON, P. A.; GROSSNICKLE, V.; SANKMAN, R.; MALLIK, D.; WANG, T.; VANGAL, S. et al. A sub-cm 3 energy-harvesting stacked wireless sensor node featuring a near-threshold voltage ia-32 microcontroller in 14-nm tri-gate cmos for always-on always-sensing applications. **IEEE Journal of Solid-State Circuits**, IEEE, v. 52, n. 4, p. 961–971, 2017.

PISTORIUS, J.; HUTTON, M.; MISHCHENKO, A.; BRAYTON, R. Benchmarking method and designs targeting logic synthesis for fpgas. In: PROC. OF INT'L WORKSHOP ON LOGIC AND SYNTHESIS. **Proceedings...** [S.l.: s.n.], 2007. v. 7.

PLAZA, S. M.; MARKOV, I. L.; BERTACCO, V. M. Optimizing nonmonotonic interconnect using functional simulation and logic restructuring. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)**, v. 27, n. 12, 2008.

POLI, R. E.; SCHNEIDER, F. R.; RIBAS, R. P.; REIS, A. I. Unified theory to build cell-level transistor networks from bdds. In: PROC. OF INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI). **Proceedings...** [S.l.: s.n.], 2003.

POSSANI, V.; MATOS, J.; BACKES, M.; JR, L. da R.; RIBAS, R.; REIS, A. Towards optimal area synthesis of small combinational circuits. In: PROC. OF INT'L WORKSHOP ON LOGIC AND SYNTHESIS (IWLS). **Proceedings...** [S.l.: s.n.], 2015.

POSSANI, V. N.; CALLEGARO, V.; REIS, A. I.; RIBAS, R. P.; MARQUES, F. S.; ROSA, L. S. da. Improving the methodology to build non-series-parallel transistor arrangements. In: INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2013 26TH SYMPOSIUM ON. **Proceedings...** [S.l.: s.n.], 2013.

POSSANI, V. N.; CALLEGARO, V.; REIS, A. I.; RIBAS, R. P.; MARQUES, F. de S.; ROSA, L. S. da. Graph-based transistor network generation method for supergate design. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 24, n. 2, 2016.

POSSANI, V. N.; MARQUES, F. S.; JUNIOR, L. S. da R.; CALLEGARO, V.; REIS, A. I.; RIBAS, R. P. Nsp kernel finder-a methodology to find and to build non-series-parallel transistor arrangements. In: INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2012 25TH SYMPOSIUM ON. **Proceedings...** [S.l.: s.n.], 2012.

PU, Y.; GYVEZ, J. P. D.; CORPORAAL, H.; HA, Y. An ultra-low-energy multi-standard jpeg co-processor in 65 nm cmos with sub/near threshold supply voltage. **IEEE Journal of Solid-State Circuits**, v. 45, n. 3, 2010.

QIU, X.; MAREK-SADOWSKA, M. Routing challenges for designs with super high pin density. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 32, n. 9, p. 1357–1368, 2013.

REIS, A.; MATOS, J. Physical awareness starting at technology-independent logic synthesis. In: REIS, A.; DRECHSLER, R. (Ed.). ADVANCED LOGIC SYNTHESIS. **Proceedings...** [S.l.]: Springer, 2017.

REIS, A.; REIS, R.; AUVERGNE, D.; ROBERT, M. The library free technology mapping problem. In: PROCEEDINGS OF THE IEEE ACM INTERNATIONAL WORKSHOP ON LOGIC SYNTHESIS. **Proceedings...** [S.l.: s.n.], 1997.

REIS, A.; ROBERT, M.; AUVERGNE, D.; REIS, R. Associating CMOS transistors with BDD arcs for technology mapping. **Electron. Lett.**, v. 31, n. 14, 1995.

REIS, A.; ROBERT, M.; REIS, R. Topological parameters for library free technology mapping. In: IEEE. INTEGRATED CIRCUIT DESIGN, 1998. PROCEEDINGS. XI BRAZILIAN SYMPOSIUM ON. **Proceedings...** [S.l.], 1998. p. 213–216.

REIS, A. I. Covering strategies for library free technology mapping. In: PROC. OF THE CONF. ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI). **Proceedings...** [S.l.: s.n.], 1999. p. 180–183.

REIS, A. I.; ANDERSON, O. C. **Library sizing**. 2011. US Patent 8,015,517.

REIS, A. I.; REIS, R.; AUVERGNE, D.; ROBERT, M. Library free technology mapping. In: VLSI: INTEGRATED SYSTEMS ON SILICON. **Proceedings...** [S.l.: s.n.], 1997.

RIBAS, R. P.; BAVARESCO, S.; SCHUCH, N.; CALLEGARO, V.; LUBASZEWSKI, M.; REIS, A. I. Contributions to the evaluation of ensembles of combinational logic gates. **Microelectronics Journal**, v. 42, n. 2, 2011.

RICCI, A.; MUNARI, I. D.; CIAMPOLINI, P. An evolutionary approach for standard-cell library reduction. In: PROC. OF GLSVLSI. **Proceedings...** [S.l.: s.n.], 2007.

ROSA, L. S.; WAGNER, F. R.; CARRO, L.; CARISSIMI, A. S.; REIS, A. I. Scheduling policy costs on a java microcontroller. **Lecture notes in computer science**, 2003.

ROSA, L. S. da; SCHNEIDER, F. R.; RIBAS, R. P.; REIS, A. I. Switch level optimization of digital cmos gate networks. In: PROC. OF INT'L SYMPOSIUM ON QUALITY OF ELECTRONIC DESIGN (ISQED). **Proceedings...** [S.l.: s.n.], 2009.

RUDELL, R. **Logic synthesis for VLSI design**. Thesis (PhD) — University of California, Berkeley, 1989.

SAXENA, P.; MENEZES, N.; COCCHINI, P.; KIRKPATRICK, D. A. Repeater scaling and its impact on cad. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)**, v. 23, n. 4, 2004.

SCHMIDT, J.; FISER, P. The case for a balanced decomposition process. In: IEEE. DIGITAL SYSTEM DESIGN, ARCHITECTURES, METHODS AND TOOLS, 2009. DSD'09. 12TH EUROMICRO CONFERENCE ON. **Proceedings...** [S.l.], 2009. p. 601–604.

SCHNEIDER, F.; RIBAS, R.; SAPATNEKAR, S.; REIS, A. Exact lower bound for the number of switches in series to implement a combinational logic cell. In: INT'L CONF. ON COMPUTER DESIGN (ICCD). **Proceedings...** [S.l.: s.n.], 2005. p. 357–362.

SENTOVICH, E. M.; SINGH, K. J.; LAVAGNO, L.; MOON, C.; MURGAI, R.; SALDANHA, A.; SAVOJ, H.; STEPHAN, P. R.; BRAYTON, R. K.; SANGIOVANNI-VINCENTELLI, A. **SIS: A System for Sequential Circuit Synthesis**. [S.l.], 1992.

SEO, J.; MARKOV, I.; SYLVESTER, D.; BLAAUW, D. On the decreasing significance of large standard cells in technology mapping. In: PROC. OF ICCAD. **Proceedings...** [S.l.: s.n.], 2008.

SHELAR, R.; SAPATNEKAR, S. BDD decomposition for delay oriented pass transistor logic synthesis. **IEEE Trans. on VLSI Systems**, v. 13, n. 8, 2005.

SILVA, D. N. da; REIS, A. I.; RIBAS, R. P. Cmos logic gate performance variability related to transistor network arrangements. **Microelectronics Reliability**, v. 49, n. 9, 2009.

SIRRINGHAUS, H.; WILSON, R.; FRIEND, R.; INBASEKARAN, M.; WU, W.; WOO, E.; GRELL, M.; BRADLEY, D. Mobility enhancement in conjugated polymer field-effect transistors through chain alignment in a liquid-crystalline phase. **Applied Physics Letters**, AIP Publishing, v. 77, n. 3, p. 406–408, 2000.

SMITH, J.; BAWOLEK, E.; LEE, Y.; O'BRIEN, B.; MARRS, M.; HOWARD, E.; STRNAD, M.; CHRISTEN, J. B.; GORYLL, M. Application of flexible flat panel display technology to wearable biomedical devices. **Electronics Letters**, IET, v. 51, n. 17, p. 1312–1314, 2015.

SOEKEN, M.; AMARU, L. G.; GAILLARDON, P.-E.; MICHELI, G. D. Exact synthesis of majority-inverter graphs and its applications. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, 2017.

STANGHERLIN, K. H. **Energy and Speed Exploration in Digital CMOS Circuits in the Near-Threshold Regime for Very-Wide Voltage-Frequency Scaling**. Dissertation (Master's degree on Computer Science) — Universidade Federal do Rio Grande do Sul, 2013.

STOK, L.; IYER, M. A.; SULLIVAN, A. J. Wavefront technology mapping. In: DESIGN AUTOMATION AND TEST IN EUROPE CONF. AND EXHIBITION (DATE). **Proceedings...** [S.l.: s.n.], 1999. p. 531–536.

SYLVESTER, D.; KEUTZER, K. Getting to the bottom of deep submicron ii: A global wiring paradigm. In: PROC. OF INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN (ISPD). **Proceedings...** [S.l.: s.n.], 1999.

TOGNI, J.; SCHNEIDER, F.; CORREIA, V.; RIBAS, R.; REIS, A. Automatic generation of digital cell libraries. In: PROC. OF SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI). **Proceedings...** [S.l.: s.n.], 2002.

WANG, A.; CALHOUN, B. H.; CHANDRAKASAN, A. P. **Sub-threshold design for ultra low-power systems**. [S.l.]: Springer, 2006.

WANG, L.-T.; CHANG, Y.-W.; CHENG, K.-T. T. **Electronic design automation: synthesis, verification, and test**. [S.l.]: Morgan Kaufmann, 2009.

WU, Q.; ZHANG, J.; QIU, Q. Design considerations for digital circuits using organic thin film transistors on a flexible substrate. In: PROC. OF INT'L SYMP. ON CIRCUITS AND SYSTEMS (ISCAS). **Proceedings...** [S.l.: s.n.], 2006.

XU, X.; CLINE, B.; YERIC, G.; YU, B.; PAN, D. Z. Systematic framework for evaluating standard cell middle-of-line robustness for multiple patterning lithography. **Journal of Micro/Nanolithography, MEMS, and MOEMS**, v. 15, n. 2, 2016.

XU, X.; PAN, D. Z. Toward unidirectional routing closure in advanced technology nodes. **IPSJ Transactions on System LSI Design Methodology**, Information Processing Society of Japan, v. 10, p. 2–12, 2017.

XU, X.; YU, B.; GAO, J.-R.; HSU, C.-L.; PAN, D. Z. Parr: Pin-access planning and regular routing for self-aligned double patterning. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, ACM, v. 21, n. 3, p. 42, 2016.

YANBIN, J.; SAPATNEKAR, S.; BAMJI, C. Technology mapping for high-performance static CMOS and pass transistor logic designs. **IEEE Trans. on VLSI Systems**, v. 9, n. 5, 2001.

YE, W.; YU, B.; PAN, D. Z.; BAN, Y.-C.; LIEBMANN, L. Standard cell layout regularity and pin access optimization considering middle-of-line. In: ACM. PROCEEDINGS OF THE 25TH EDITION ON GREAT LAKES SYMPOSIUM ON VLSI. **Proceedings...** [S.l.], 2015. p. 289–294.

ZHU, W.; YAO, K.; ZHANG, Z. Design and fabrication of a novel piezoelectric multilayer actuator by thick-film screen printing technology. **Sensors and Actuators A: Physical**, Elsevier, v. 86, n. 3, p. 149–153, 2000.