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

CALEBE MICAEL DE OLIVEIRA CONCEIÇÃO

Minimizing Transistor Count in Transistor Networks

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

Advisor: Prof. Dr. Ricardo Augusto da Luz Reis

Porto Alegre
Outubro 2019

CIP — CATALOGING-IN-PUBLICATION

Conceição, Calebe Micael de Oliveira

Minimizing Transistor Count in Transistor Networks / Calebe Micael de Oliveira Conceição. – Porto Alegre: PPGC da UFRGS, 2019.

98 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Advisor: Ricardo Augusto da Luz Reis.

1. EDA. 2. Logic Synthesis. 3. Library free. 4. Cell clustering. 5. Transistor network. 6. Transistor count. 7. Microelectronics. I. Reis, Ricardo Augusto da Luz. II. Título.

"Não to mandei eu? Esforça-te, e tem bom ânimo;
não temas, nem te espantes; porque o Senhor teu Deus
é contigo, por onde quer que andares." — (JOSUÉ 1:9)

ACKNOWLEDGEMENTS

(*Only in Portuguese. Thanks for your comprehension.*) Eu estava prestes a concluir a graduação em Ciência da Computação na Universidade Federal de Sergipe e, apesar da segurança com que realizei as atividades acadêmicas visando seguir carreira acadêmica, naquele momento eu estava com medo. Orei pedindo uma resposta clara e inequívoca sobre qual caminho seguir dali pra frente. A resposta veio em forma de aprovação, e pude escolher entre alguns dos melhores programas de pós-graduação do Brasil. Agradeço a Deus por ter guiado minhas decisões até aqui, e principalmente por ter me sustentado em cada necessidade.

"No trabalho, busque as pessoas boas, Calebe. Depois filtra por currículo". Esse foi um dos valiosos conselhos que recebi do meu querido orientador na graduação, o professor Luiz Brunelli. Sábias palavras. E Deus foi tão bom comigo que trouxe a mim o professor Ricardo Reis, que tem um coração ainda maior que seu extenso currículo. Reis, muito obrigado pelas oportunidades, e especialmente por não me deixar desistir. "Calebe, não está morto que peleia!". Dessa lição eu não vou esquecer.

Na UFRGS convivi com colegas brilhantes, por quem nutro carinho e admiração. Companheiros de laboratório, de viagens, de futebol, de saídas nos finais de semana, e sempre parceiros nas angústias que a pós-graduação é doutora em causar. Amigos que foram essenciais à minha adaptação na nova terra. Obrigado, meus amados. Queria muito que todos estivessem aqui para ler isso.

Agradeço a cada professor que cruzou meu caminho nessa longa jornada, seja em sala de aula, nos congressos, bancas, palestras, encontros, seminários... O conhecimento que vocês compartilharam pavimentaram meu caminho, e algumas intervenções foram luz em momentos de muita dúvida sobre a condução do trabalho. Agradeço ainda aos servidores técnico-administrativos da UFRGS, gentis, cuidadosos e solidários em cada necessidade que tive.

Aos estudantes do IFRS Restinga, onde comecei, aos estudantes do IFSUL Charqueadas, onde permaneço, muito obrigado! Talvez vocês não saibam, mas a empolgação, o respeito, o incentivo, o carinho, e o *feedback* sincero e imediato de vocês repunham a auto-confiança profissional que as agruras da pós-graduação sugavam de mim.

Essa tese não teria sido feita sem a importante contribuição de amigos. Em especial, agradeço a Graci pelas discussões e parceria no início desse trabalho, e pelo constante incentivo. Sou teu fã, e talvez não tenhas a real dimensão de como suas palavras foram essenciais em alguns momentos. Agradeço à Gisell pela parceria no momento em que a tese precisava tomar corpo. Eu não teria conseguido fazer muitos dos experimentos sem tua ajuda. Agradeço aos amigos de trabalho no IFSUL que, ao compartilharem suas experiências nos bastidores da pós-graduação, me ajudaram a não me sentir o pior dos seres, e entender que o caminho não é fácil pra ninguém. Em especial, agradeço a Carol por escutar meus desabafos, pelos conselhos, por ter se disposto a revisar os meus textos mesmo não sendo especificamente a sua área, e por toda torcida.

Não posso deixar de agradecer aos antigos gestores do instituto onde trabalho. Com o fim desse ciclo, mais um doutor entra para o quadro funcional da instituição. Agradeço a carga horária média de 20 horas-aula semanais, das mais elevadas dentre todos os campi; agradeço as regras limitantes de afastamento para docentes em probatório, que elevaram à condição de mero devaneio o meu anseio por me afastar para concluir essa etapa com tranquilidade. Essas situações apimentaram meu desafio, e tornam essa vitória ainda mais saborosa. Investi todas as férias e finais de semana que pude. Concluo o doutorado com a satisfação de ter conseguido chegar até aqui sem negligenciar nenhum aluno que precisou de mim. Nem um sequer. Agradeço ainda aos profissionais de saúde mental que me atenderam, e me ajudaram a transpor essa barreira.

O caminho não foi fácil. Paguei o preço por ter escolhido não adiar os planos de construir uma família, e não me arrependo. Agradeço à minha amada esposa Marilia por sua parceria, carinho, e amor. Ao meu filho Miguel, por iluminar até os dias mais sombrios com sua inocência e energia. Agora posso dispor sem culpa de mais tempo para brincar contigo, meu gurizinho! Aos meus sogros Genice e Paulo, e à minha cunhada Milene, agradeço pelo apoio de todas as horas. Agradeço aos meus irmãos Iuri e Jonathas, minha torcida mais fiel. Ao meu pai, de quem herdei o hábito de estudo. E, como não poderia deixar de ser, registro aqui um agradecimento todo especial à minha mãe, Ivete. Você foi a primeira a acreditar em mim, viveu o sonho junto comigo, e se sacrificou por ele. Nunca me faltou. Sei o quanto tens orado por mim. O Senhor ouviu as tuas preces. Eu te amo, minha baixinha!

ABSTRACT

The evolution of the Integrated Circuits Technology demands optimization of IC design. Nowadays, many circuits use much more transistors than necessary as a broad set of ASICs use a library of pre-designed cells. The small number of logic functions that a traditional cell library provides represents an inherent limitation in the optimization of the number of transistors in the circuit. This limitation directly influences the circuit performance. A library free design approach is necessary to obtain optimized circuits, using tools to allow the layout synthesis of any transistor network. The goal of this thesis is to develop a method to optimize the logical netlist of a circuit willing to reduce the number of transistors, connections, and vias. The optimized netlist serves as input to the layout synthesis tool. We post-process the original netlist generated in the traditional standard cell design flow systematically, replacing sets of cells by one new gate of equivalent logic generated on demand to reduce the number of transistors. We merge groups of connected combinational cells of unitary fanout into a new complex gate that is, in general, not available in the traditional cell library. The new gate has a custom transistor network that can be appropriately arranged and sized to fit the specific requirements of its location in the circuit. The experiments performed so far show that our method allows about 13% of reduction of the number of transistors in the entire circuit in comparison to netlists generated using other logic minimization tools. We also reduce the number of instances, contacts, and connections in the experiments we performed in 14%, 11%, and 10% on average, respectively, when compared to the netlist generated with a leading academic logic synthesis tool. We also investigate the impact of the proposed optimization in area and wirelength, achieving an estimated average reduction of 5% in the area and up to 14% reduction in total wirelength. These results evidence the optimization opportunities neglected in the standard cell design approach and show the advantages of library free synthesis.

Keywords: EDA. Logic Synthesis. Library free. Cell clustering. Transistor network. Transistor count. Microelectronics.

Minimizando o Número de Transistores em Redes de Transistores

RESUMO

A evolução da Tecnologia de Circuitos Integrados exige otimização do projeto do circuito. Atualmente, vários circuitos usam muito mais transistores do que o necessário, pois um amplo conjunto de circuitos ASICs utiliza biblioteca de células pré-projetadas. O número reduzido de funções lógicas que uma biblioteca de células tradicional fornece representa uma limitação inerente na otimização do número de transistores no circuito, influenciando diretamente as métricas usuais de desempenho do circuito, como área, dissipação de energia e atraso. Uma abordagem de projeto livre de bibliotecas é necessária para obter circuitos otimizados, usando ferramentas para permitir a síntese de layout de qualquer rede de transistores. O objetivo desta tese é desenvolver um método para otimizar a netlist lógica de um circuito de modo a reduzir o número de transistores, número de conexões e número de vias. A netlist otimizada serve como entrada para a ferramenta de síntese de layout. Nós pós-processamos a netlist original gerada no fluxo de design de célula padrão tradicional e sistematicamente substituímos conjuntos de células por uma nova porta com lógica equivalente, gerada sob demanda para reduzir o número de transistores. Consideramos a mesclagem de grupos de células conectadas de *fanout* unitário em uma nova porta complexa que normalmente não está disponível na biblioteca de células tradicional. A nova porta possui uma rede de transistores personalizada que pode ser adequadamente organizada e dimensionada para atender aos requisitos específicos de onde ela está localizada no circuito. Os experimentos realizados até o momento mostram que a abordagem proposta é capaz de reduzir o número de transistores em todo o circuito em até 13 % em comparação com netlists geradas usando outras ferramentas de minimização, independentemente do tamanho da biblioteca de células padrão usada inicialmente para sintetizar a netlist original. Também reduzimos o número de instâncias, contatos e conexões nos experimentos realizados em 14 %, 11 % e 10 % em média, respectivamente, quando comparados com a netlist gerada com uma ferramenta acadêmica líder em síntese lógica. Investigamos também o impacto da otimização proposta na área e comprimento de fio, alcançando uma redução média estimada de 5 % na área e de até 14 % no comprimento total de fio. Esses resultados evidenciam as oportunidades de otimização negligenciadas na abordagem de projeto com células padrão, e reforçam as vantagens do projeto livre de biblioteca.

Palavras-chave: EDA, síntese lógica, projeto livre de bibliotecas, agrupamentos de células, redes de transistores, contagem de transistores, microeletrônica.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	Application Specific Integrated Circuit
CMOS	Complementary Metal Oxide Semiconductor
EDA	Electronic Design Automation
IC	Integrated Circuit
FFC	Fanout Free Cone
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
LUT	Look Up Table
NBTI	Negative Bias Temperature Instability
NRE	Non Recurring Expense
NOCL	Nangate Open Cell Library
RTL	Register Transfer Level
SCCG	Static CMOS Complex Gate

LIST OF FIGURES

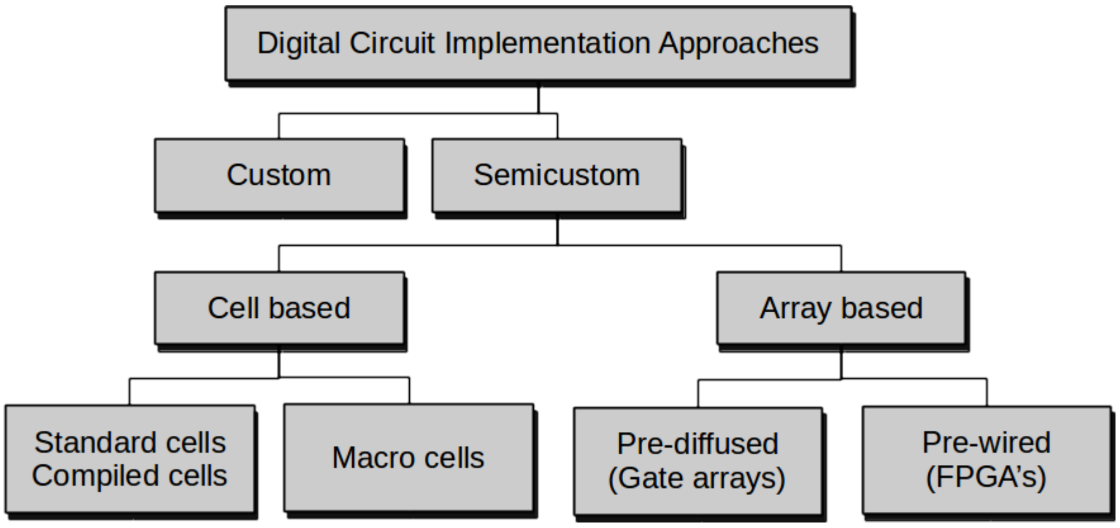
LIST OF TABLES

LIST OF ALGORITHMS

1 INTRODUCTION

In the last five decades, the world has experienced an increasing use of Integrated Circuits (ICs) in many and diverse consumer applications, notable by the considerable changes it has promoted in the way people interact with each other and with the objects. However, far beyond the details seen from a customer's point of view, the microelectronics industry has evolved, especially the design process, as a way to maintain such an increasing market. As presented in (RABAEY; CHANDRAKASAN; NIKOLIC, 2002), many implementation strategies for digital ICs have been adopted, whose choice is mainly driven by economic considerations. A taxonomy of these methodologies is presented in Figure 1.1, divided into two main classes: custom and semi-custom methodologies.

Figure 1.1: Overview of Implementation Approaches for Digital Integrated Circuits.



Source: (MICHELI, 1994) apud (RABAEY; CHANDRAKASAN; NIKOLIC, 2002)

The custom design methodology was the most traditional and the only option in the early days of IC design. Its main characteristics are a high design cost, a long *time to market* due to the time spent drawing mainly manually each layout mask, and, on the other hand, with a high degree of freedom to implement a layout. Due to its labor-intensive nature, the market share of custom design reduces from year to year as the design automation advances. Only the most performance-critical modules justify to have their components designed manually; otherwise, a library cell-based design is mainly used. (RABAEY; CHANDRAKASAN; NIKOLIC, 2002).

The so-called semi-custom methodologies are alternatives centered on the reuse of portions of previous designs, which are divided into the cell-based and array-based design. The array-based designs reuse several mask layers of regular fabric to considerably reduce costs, requiring only a limited set of extra processing steps or yet completely eliminating the processing, saving design costs, and production time with a trade-off in performance. Examples of them are the Gate Arrays and FPGAs, but this kind of digital design approach is beyond the scope of this thesis. In effect, this thesis discusses some of the main characteristics of cell-based design approaches and focuses on overcoming their limitations by optimizing their transistor network.

1.1 A historical perspective

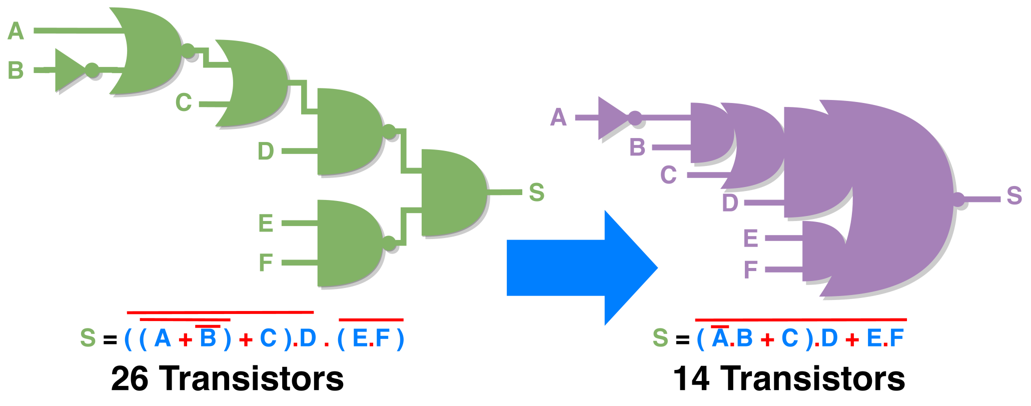
The physical design of integrated circuits was done only by hand, as a full custom design, till the end of the 1970s. By that time, the computer was used for drawing circuits, performing design and verification of such circuits, but there was no tool to automate the physical design (NEWTON, 1982). It was only in 1979 that Motorola 68000 was released, the first representative microprocessor to use regular blocks like ROMs (Read Only Memories) and Programmable Logic Arrays (PLAs), which were the first technological endeavors to automatize the physical design (REIS, 2008).

By the year of 1982, the standard cell design had already emerged as a prominent design methodology in which a library of custom-designed cells is used to implement a logic function. It standardizes the design entry level at the logic gate, and has the complexity of simple logic gates or flip-flops, besides being restricted to constant height in order to aid packing and ease of power distribution. Unlike the array-based approaches, standard cell layout involves the customization of all mask layers (NEWTON, 1982).

An extensive set of chip designs, mainly ASIC ones, uses a traditional standard cell design approach. It is a well-established flow based on the use of a cell library to implement the desired circuit, covering steps from translating a high-level description of the circuit into a netlist of logic gates available in the cell library. Many exciting challenges appear in the whole flow, like finding the best cell's placement to minimize the total wire length, ensuring that every connection of a circuit is routed through the available metal layers without s short circuit, generating a clock tree to feed the whole circuit with minimal skew. There are several mature EDA (Electronic Design Automation) tools that handle these problems. However, the focus of this thesis is in the late steps of the logic synthesis, working on deciding which cells will be part of the final logic gate netlist.

In a traditional standard cell design flow (like the one provided by major EDA vendors), a circuit description must fit the targeting cell library. A standard cell library has no more than 100 different logic functions, while, for example, there are 3057 possible logic functions with up to 4 (four) serial transistors (DETJENS et al., 1987). Therefore, it is clear that the technology mapping phase can be a source of worsening since the identified logic function in the early stages of logic synthesis must fit the provided limited set of functions. As shown in Fig. 1.2, the lack of specific logic functions in a cell library can lead to designs that use more transistors than it is needed.

Figure 1.2: Equivalent logic functions implemented with a set of basic logic gates and using an SCCG.


A
B
C
D
E
F
S
$S = \overline{(\overline{(A+\overline{B})} + C).D} . \overline{(E.F)}$
26 Transistors
A
B
C
D
E
F
S
$S = \overline{(\overline{A}.B + C).D + E.F}$
14 Transistors

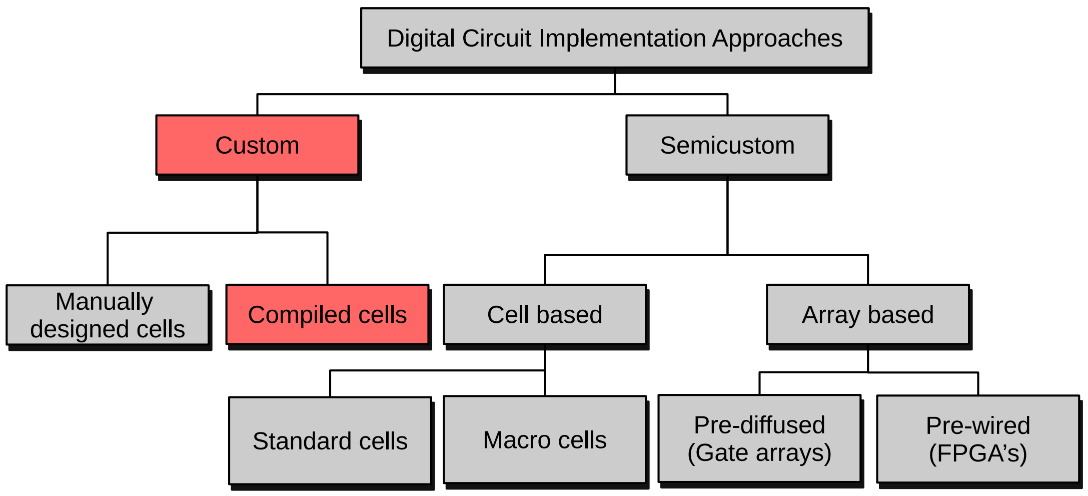
Source: own authorship

In modern technologies, mainly below 65nm, the power consumption due to leakage current became so important as the dynamic power consumption. This characteristic has inspired the industry to develop the FinFET and FDSOI technologies, (first introduced to 28nm and below), in order to keep attending the increasing demand for low power and energy-efficient devices (WEBER, 2017). However, for many applications, it is enough to use older and cheaper technology nodes (BAMPI; REIS, 2011), where reducing power consumption is also desired. Therefore, if we want to optimize power leakage for those applications, it is necessary to develop new methods capable of reducing the number of transistors needed to implement a specific function (REIS, 2011b). By allowing the use of other logic functions rather than the ones usually available in the cell library, it is also possible to reduce the number of wires (and vias) in a circuit, which is more than welcome, as routing is a critical step in a modern IC design.

The simple addition of new cells to the cell library in a traditional design flow would increase the complexity of the technology mapping algorithms; require the development of new ones – which not necessarily would find the best solutions –; and significantly increase the Non-Recurring Expense (NRE) for the library vendor, among other issues(MURGAI, 2015). As an alternative to the standard cell design methodology, the library free technology mapping uses a big virtual library instead of using a physical cell library (REIS et al., 1997)(BERKELAAR; JESS, 1988)(REIS, 1999). The logic synthesis optimization runs using a more limited representation of the cell characteristic, and it is assumed that any optimized logic netlist can be mapped into silicon using any network of transistors instead of a network of gates.

In Figure 1.1, the customized networks of transistors, also referred to in the related literature as Static CMOS Complex Gate (SCCG), appears as part of the *Compiled Cells*. This methodology is possible if we have access to tools that can generate the layout of any network of transistors, due to the complexity of handmade layout design. That is one of the first efforts of our research group that started in late 80's with TRANCA methodology. Several tools were developed in this scope, among which is ASTRAN, a layout automation tool capable of generating a cell layout from a netlist of transistors written in SPICE, with a quality comparable to handcrafted ones (POSSER et al., 2010) (ZIESEMER; REIS, 2014).

Figure 1.3: Implementation Approaches for Digital Integrated Circuits, including Compiled Cell approach.



Adapted from:(MICHELI, 1994) apud (RABAEY; CHANDRAKASAN; NIKOLIC, 2002)

From the taxonomic point-of-view previously shown in Figure 1.1, we can also add the Compiled Cell approach, as the modification shown in Figure 1.3. This approach can be seen as a mixture of Semi-custom and Custom designs methodologies, working as an automatized full custom approach. Looking at the transistors as the building block of a circuit enables to do continuous sizing of transistors instead of discrete sizing limited to the few available sizes, and allows to use different logic styles and different transistor network topologies (REIMANN; SZE; REIS, 2016; FLACH et al., 2014; POSSER et al., 2012). These options can address each contextualized demand of the inferred transistor arrangements of a logic function, either generated on-demand or by reuse. On the other hand, it is still possible to use tools and techniques from the traditional physical design steps of cell-based methodologies. Focusing on building transistor network instead of using pre-characterized logic gates has demonstrated advantages, like a reduction of leakage power and delay shown in (SCARTEZZINI; REIS, 2011) mainly due to the reduction in transistors count, as well as the possibility to explore different transistor network arrangements willing to optimize leakage power as shown in (TONFAT; FLACH; REIS, 2016).

As partially shown in our previous works (CONCEIÇÃO; POSSER; REIS, 2016; CONCEIÇÃO et al., 2017) (CONCEIÇÃO; REIS, 2019), this thesis presents in more detail our cell replacement methodology willing to improve transistor networks. The netlist resulting from a standard cell design methodology is refined through the replacement of a set of connected combinational gates of unitary fanout by a single Static CMOS Complex Gate (SCCG) of equivalent logic, focusing on the reduction of the transistor count of the whole circuit. The reference (ROY; BHATTACHARYA; BOPPANA, 2005) is also a related work, but the novelty of our technique relies on the reduction of transistor count as the performance parameter.

1.2 Justification

Cell libraries usually cover a reduced number of logic functions compared to all possible functions, which can be a primary source of de-optimization (REIS, 2011a). The way to solve this problem is somehow to provide new functions. Ways trying to overcome this situation can be found in literature, usually assuming that every needed gate would be generated on demand.

Previous works attempt to circumvent this limitation by providing new cells for a given circuit, either by generating a custom cell library – with cells generated on demand – for each circuit and then performing the traditional technology mapping, as proposed in (PILATO; FERRANDI; PANDINI, 2011); or by synthesizing the circuit targeting a size-able virtual library and then generating the gates as needed, as proposed in (REIS et al., 1997). This latter approach has been investigated and improved over the years, but it still faces the computational hardness for actually optimizing the whole circuit at once using such an extensive library. Indeed, these alternatives to the traditional standard cell design flow can lead to solutions with better parameters of design quality. As discussed in (REIS, 2011a), adding more and flexible cells allows more degrees of freedom to design the circuit, such as continuous gate sizing and different transistor arrangements. Nevertheless, it seems practical to take advantage of the maturity of the software tools for standard cell logic synthesis to generate an initial netlist, and only after to provide on-demand transistor networks to optimize small portions instead of handling the whole circuit.

1.3 Goal

The main goal of this thesis is to evaluate the effectiveness of the systematic replacement of groups of combinational cells in a synthesized netlist by a logically equivalent transistor network as a way to decrease the total number of transistors, wires, and vias.

As underlying goals, this thesis also aims to:

- measuring how academic and commercial logic synthesis tools perform regarding transistor count optimization parameter;
- developing an EDA tool that incorporates the proposed optimization;
- evaluating the technique regarding the number of connections and number of vias;
- estimating the impact of the proposed procedure over the length of the interconnections, circuit congestion, and the estimated area.

In more concrete terms, this thesis aims to answer the following questions regarding the proposed method:

- How many cells of unitary fanout a netlist of a circuit usually has?
- How are cells of unitary fanout placed in a circuit? Does it depend on the placement algorithm? How should be the placement algorithm to favor the proposed

technique?

- How the choice of the first cell library influences the netlist regarding the number of instances, wires, and transistors?
- How effective is the technique to reduce the number of transistors in a circuit?
- How much of the circuit is affected by the proposed cell merging technique?
- How it impacts the area initially occupied by the first cells? What is the impact of the entire circuit area?
- How the technique influences the connections of the circuit?

1.4 Hypothesis

By focusing on optimizing the overall number of transistors, improvements in the traditional quality parameters (area-power-delay) of the circuit can be achieved. We guess that a smaller number of transistors means less area and, by consequence, less energy spent to power the circuit. The proposed method intends to find in the circuit connected cells of unitary fanout and replace them by a transistor network. By replacing such a group of cells, the wires that interconnect them also vanishes, as well as their vias, thus improving routability and reliability.

There are works suggesting designers to avoid using large standard cells for new technologies, claiming that results in long wires, which are more susceptible to electromigration and antenna effects. That is debatable since we expect an overall reduction in the area when using fewer transistors, which would approximate sink and result in shorter wires. Also, using such cells is still a practical optimization solution for many technology nodes still operational, and it can be applied to newer technology nodes as soon as the researches overcome the inherent limitations in the fabrication process.

1.5 Contributions of the thesis

The main contributions of this thesis are

- A study about transistor count as quality parameter of both academic and commercial tools;
- A novel approach to achieve a library free design methodology of digital design;

- A systematic and validated procedure to reduce transistor count in a circuit; and
- A new tool for optimization of logic netlists to compose the TRANCA methodology of our research group, to work as an input to ASTRAN layout automation tool.

We organized this thesis as follows. The second chapter discusses some basic concepts and presents the previous related works. The third chapter focuses on to discuss the adopted modeling and the algorithms designed in the scope of this thesis. The fourth chapter presents a full and detailed example of the clustering and substitution procedure that we are proposing an optimization method. The experiments we have developed in the scope of this thesis to find answers to the hypothesis are presented in chapter five. We conclude our discussion about transistor count as a parameter to be optimized in chapter six.

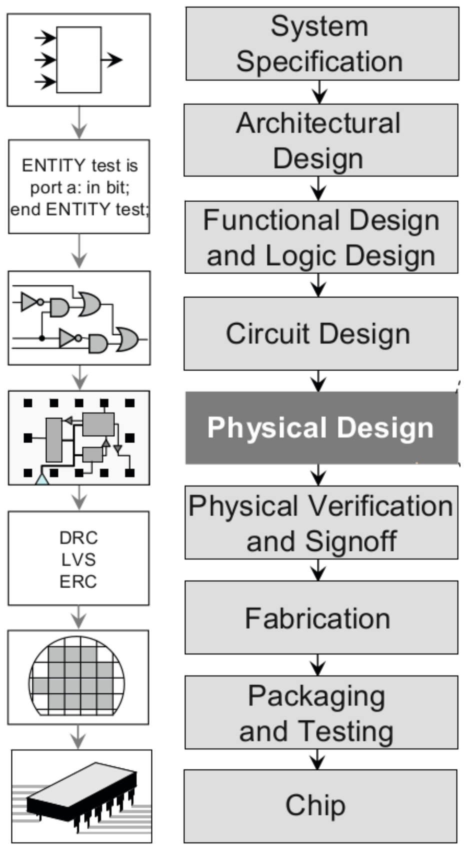
2 MAIN CONCEPTS AND PREVIOUS WORKS

This chapter presents the main concepts and definitions in the scope of this thesis. It also aims to review the previous works that make the way to state of the art and helps to characterize the innovation of this thesis.

2.1 Traditional Standard Cell Design Flow

In order to put the subjects related to this thesis proposal in a context, it is necessary to start by presenting and explaining the essential steps of the VLSI circuit design flow. It is shown on Figure 2.1 an organization of the general steps as presented in (KAHNG et al., 2011). It brings an overview of the significant steps of VLSI design and omits verification activities that usually run in parallel with the general flow, sometimes represented as return arcs between subsequent phases and eventually has overlaps between phases. However, the goal is to give an overview of the entire design flow, and not to exhaust the subject in all of its details.

Figure 2.1: The major steps in the VLSI circuit design flow.



Source: (KAHNG et al., 2011)

IIt all starts with the System Specification phase when circuit designers define the overall goal and high-level requirements of the system. The Architectural Design phase concerns to determine the basic architecture to meet the system specifications. The next Functional and Logic Design phase embraces the high-level behavior and connectivity of each module. It captures the specification of these characteristics in register-transfer level (RTL) using a hardware description language (HDL), of which Verilog and VHDL are the most representative examples. Logic synthesis tools are then applied to map the HDL description of the functionality into a netlist, composed of a list of signals and specific circuit elements such as standard cells from a given technology library, which is usually described in liberty format.

The next phase is the Circuit Design, in which some critical low-level elements must be designed at the transistor level, such as RAM blocks, I/O, high-speed functions (multipliers), whose correctness of circuit-level design is predominantly verified by circuit simulation tools such as SPICE. This makes room for the Physical Design phase that intends to instantiate the geometric representations of fixed shapes of all design components (cells, macros, gates, transistors, etc.) in spatial locations distributed in a flat area, connected with appropriate routing connections done in the available metal layers.

In order to ensure correct electrical and logical functionality of the previous step, Physical Verification is performed. As a result, layout modifications may be demanded, which has to be minimal and careful to bring no new problems. Once all the Signoff tasks are performed, the circuit design is ready to be sent to a foundry where it will be manufactured, packaged, tested, and sent back as a chip. Further discussions about these last phases are beyond the scope of this thesis.

Indeed, the design flow of the digital VLSI circuit is quite complicated. Design decisions have to be made and will impact some of the conflicting quality parameters of the resulting IC, such as performance, area, reliability, power dissipation, and yield (KAHNG et al., 2011). Also, as traditional circuit design is like a cascade methodology (therefore the cost of changes increases in late stages), it is vital to make the right decision in early stages in order to prevent problems in the last stages, when a fix is more costly and difficult. In other words, when optimization is done in an early stage of the design flow, it tends to give more significant results.

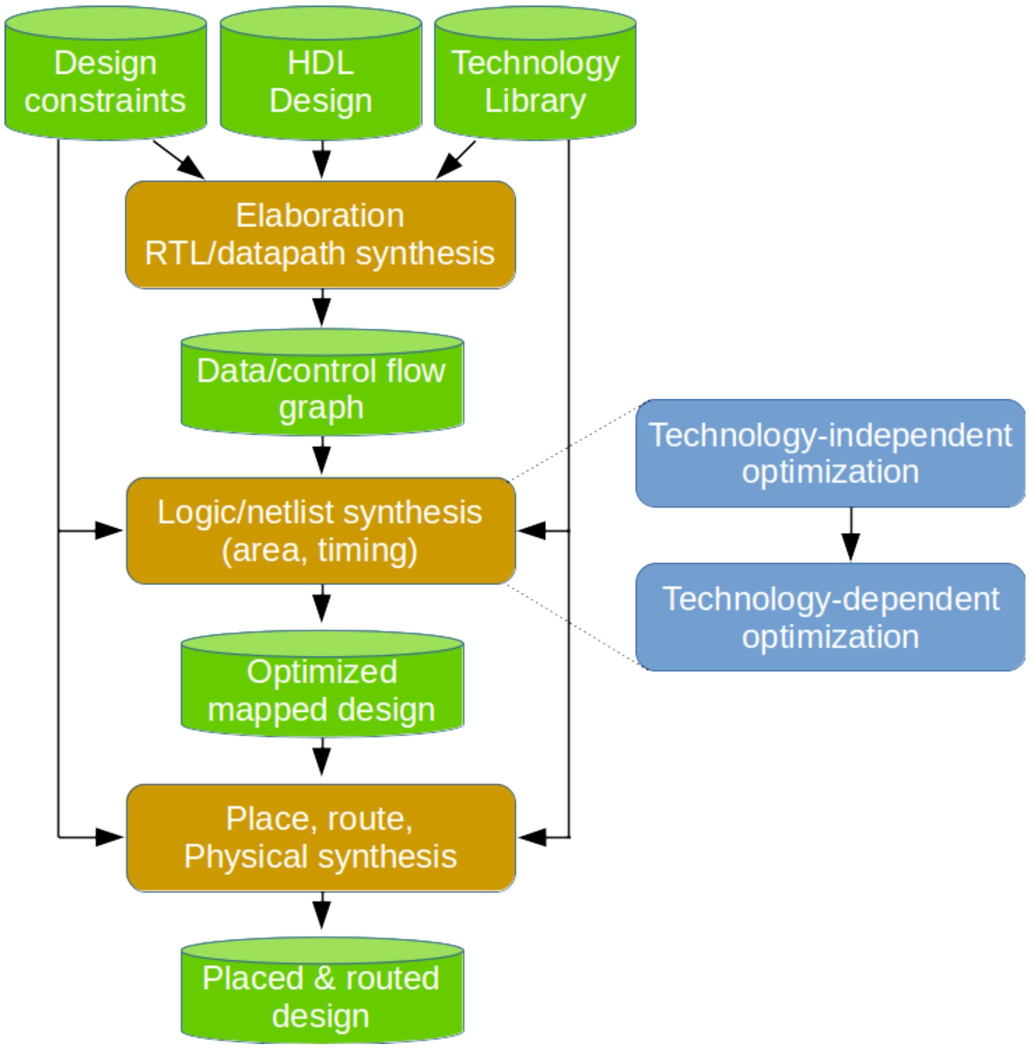
2.1.1 Logic Design Phase

The main goal of this thesis is in the logic synthesis phase of the digital integrated circuit design flow, specifically focused on the usage of compiled cells (see Figure 1.1). Going into details of Logic Design phase, since the writing of the desired functionality of the system into an HDL description is mostly a manual task, it matters to present the tasks performed by the Logic Synthesis tools, which are divided into two main steps: Technology Independent Mapping and Technology Dependent Mapping, as presented in Figure 2.2.

Technology-independent Optimization aims to derive an optimized netlist consisting of generic gates, using a general cost for design area and/or delay. This netlist serves as input to Technology-Dependent Optimization, which uses technology information such as the area and the delay from cells of a cell library and the design rules to derive a netlist composed of library gates satisfying the required area and delay objectives.

Exact technology mapping is an intractable problem under most practical scenarios, and a set of heuristic tasks called Postmaping Optimization is usually performed over the mapped netlist in some subsequent phase of the circuit design flow. Postmap-

Figure 2.2: Details of the Logic Synthesis flow.


Design constraints
HDL Design
Technology Library
Elaboration RTL/datapath synthesis
Data/control flow graph
Logic/netlist synthesis (area, timing)
Technology-independent optimization
Technology-dependent optimization
Optimized mapped design
Place, route, Physical synthesis
Placed & routed design

Source: (MURGAI, 2015)

ing transformations can improve the circuit characteristics (such as delay, area, power, routing congestion, signal integrity) before, during, or after the layout. Some examples are Gate Resizing, Fanout Optimization/Buffering, Gate Replication, Simple Gate Decomposition/Collapsing, Resynthesis and Remapping, and Pin Permutation (MURGAI, 2015).

Gate Resizing is an in-place optimization technique that consists in selecting the size of each gate such that some objective function is minimized without violating any constraint, which has minimal impact on placement and route of cells that can be applied during/after placement or post-routing when more accurate wire load and delay estimates become available.

Fanout optimization/Buffering seeks to optimally distribute a signal from the driver gate (source) to the fanout gates (sinks) using buffers, without violating the drive capacity of the source or those inserted buffers.

Like Buffering optimization, Gate Replication is a way to speeding up a design by redistributing the fanout load. It consists of replicating a gate *g* into *k* copies, and then partition the fanout gates of *g* among the *k* copies. It has the potential of reducing the delay through the circuit with an area penalty and an increase in fanout of fanin gates of *g*.

Simple Gate Decomposition/Collapsing refers to replace a multiple-input (at least three inputs) gate by two or more simpler gates present in the gate library. This technique keeps the design mapped after the transform, and works by identifying delay critical multiple input simple gates in the design and consider them for decomposition into two simple gates (like AND, OR, NAND, NOR, XOR or XNOR gates).Reversely, there are situations where simple gate collapsing can be done if the circuit timing or routing congestion improve.

In Resynthesis and Remapping technique, the idea is to identify regions of the design that are critical and resynthesize them using techniques such as tree height reduction, critical path resynthesis, logic minimization, among others. The combined cost (such as delay and area) of the new remapped region is compared to the first region, and a replacement is done if it has improved.

Pin permutation technique works by permuting the pins of a gate where input wires are assigned in order to find a permutation that minimizes delay. It makes sense because the arrival times of a pin may vary according to cell placement and wire routing for that signal, which cannot be known *a priori* during logic synthesis. Pin permutation does not disturb gate placement and affects routing minimally (only the connections to pins of the same gate have to be permuted).

Gate resizing, Gate Collapsing, and Resynthesis/Remapping techniques relies on a limitation: the new gate that will be used has to be available in the cell library. Those techniques might be improved if a gate specifically designed to that demand is available. In fact, probably the whole synthesis can benefit of an on demand cell generation, as we intend to discuss in this thesis.

2.2 Library free technology mapping

A traditional standard cell library provides about 150 different logic functions, with three or four different sizings each, and usually presents optimized versions for low power and high-performance designs (REIS, 2011a). This number is far below the number of possible functions that could be realized. It is shown in (DETJENS et al., 1987) a discussion about the possible number of logic functions that can build up using a number of transistors in series considering only the complementary topology of CMOS (*Complementary Metal Oxide Semiconductors*) transistors, which is presented in table 2.1. As it is further discussed in (SCHNEIDER, 2007), the usual number of logic cells in a cell library can look even smaller when other transistor network arrangements are taken into consideration.

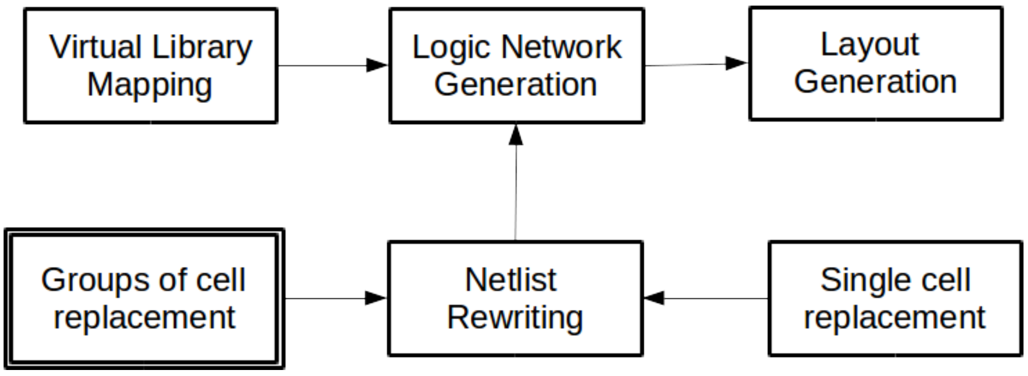
Table 2.1: Number of possible different functions using a limited number of stacked P and stacked N transistors.

		Number of stacked PMOS transistors				
Number		1	2	3	4	5
of	1	1	2	3	4	5
stacked	2	2	7	18	42	90
NMOS	3	3	18	87	396	1677
transis-	4	4	42	396	3503	28435
tors	5	5	90	1677	28435	125803

Source: (DETJENS et al., 1987)

As shown below, there are many works available in the literature that aims to relax the limitation in the number of cells in the traditional standard cell design flow by targeting compiled cells. The map presented in Figure 2.3 is proposed to organize such initiatives. All of these initiatives assumes that the needed cells will be generated on demand by a layout generator tool.

Figure 2.3: General activities of logic synthesis alternative to traditional standard cell flow



Source: own authorship

The automatic Layout Generation itself is an essential step of the library independent initiatives and the main focus of some related works. The ASTRAN tool is an example, which allows automatic layout generation for technologies down to 45 nm from its description in SPICE format, supporting the generation of any kind of transistor networks and continuous gate sizing (ZIESEMER; REIS, 2014). ASTRAN [1] is open-source and the most recent layout generator tool developed in UFRGS, in an effort that started in 1981 with TRANCA methodology, as it is well discussed in (ZIESEMER JR., 2014). This is a quite mature open source tool that generates layout comparable to handmade ones, and it has being applied in several works from different labs related to digital design, covering studies about gate sizing on digital designs (POSSER et al., 2010) up to generation of asynchronous cells (ZIESEMER et al., 2014). There are also other tools, like the commercial tool Library Creator[2] from Nangate Inc.

Virtual Library Mapping refers to the initiatives that map the design to a set of logic gates that will implement the circuit. It is very similar to traditional library mapping but targeting a virtual library specified in fewer details than the Liberty format, usually in a format called genlib [3] where only the logic function, the area of the cell, and each pin load and delays are specified. The circuit is initially described in an abstract data structure – also used as intermediary representation in traditional logic synthesis – such as Binary Decision Diagrams (BDDs) and their variants, And-Inverter Gates (AIGs), And-Or-Inverter Gates (AOIGs), Majority Gates, among others as it is discussed in (SILVA, 2017), and then a structural Verilog is output from the mapping phase. The works in this approach usually targets the whole synthesis to a virtual library. This kind of approach represents the first initiatives towards a physical design of transistor networks instead of cells (REIS et al., 1997) (MORAES et al., 2000) (GAVRILOV et al., 1997) (JIANG; SAPATNEKAR, 1999) (ONODERA; HASHIMOTO; HASHIMOTO, 2001) (XUE; AL-KHALILI; ROZON, 2004) (MARQUES et al., 2007). Maybe the most representative tools that support this kind of logic synthesis flow is the open source ABC tool, from University of California at Berkeley[4].

The Logic Network Generation is the focus of a considerable portion of the related works. Once a logic function is decided to be part of a circuit implementation, it is necessary to implement it in a physical way. This is the goal of the works we classify in

[1]Available in: https://github.com/aziesemer/astran/
[2]More in: http://www.nangate.com
[3]Specification available in: https://www.ece.cmu.edu/ ee760/760docs/genlib.pdf
[4]Available in: https://bitbucket.org/alanmi/abc

this category: given a logic function as input, derive a suitable transistor logic network among the possible topologies, sizings, and logic styles. Some authors use functional composition for factoring boolean functions regarding multi-objective goals, from which series-parallel transistor network can be directly obtained (MARTINS et al., 2010). Others minimize the number of transistor needed to implement a circuit by exploring non series-parallel transistor networks and non-planar network structures (POSSANI et al., 2016). Also, there are other works specifically tailored to generate transistor networks for multi-output logic functions with a minimal number of transistors (KAGARIS, 2016). There are also more recent works towards a layout aware transistor network generation, like in (SMANIOTTO et al., 2017) and (Cardoso et al., 2018), where the network generated using the technique presented in (POSSANI et al., 2016) is improved to avoid breaks in the diffusion wells.

In all the previous approaches for library free synthesis, no library of cells characterized *a priori* is used. In the Netlist Rewriting approaches, an already mapped circuit is taken as input by an additional task to the traditional design flow. The input netlist can be generated either by a synthesis based on a traditional standard cell library or by a synthesis based on a virtual library. Individual gates and/or portions in the resulting netlist are replaced by new cells explicitly generated to the identified context. For both fronts, it is still necessary to generate a suitable logic network for the new gates.

This idea can be found in some recent works developed in our research group. In (GUIMARãES; PUGET; REIS, 2015), individual gates are evaluated to be replaced by new automatically generated ones, using neural networks to decide when to perform the replacement. The approach of replacing groups of cells can be found in other works of our group (CONCEIÇÃO; POSSER; REIS, 2016) (CONCEIÇÃO et al., 2017) (SILVA, 2017) (CONCEIÇÃO; REIS, 2019), where groups of interconnected cells of unitary fanout are identified and replaced by a single and logically equivalent complex cell, as we will further discuss in more details.

Specifically, in (SILVA, 2017), a tool called LOMGAM (Logic Minimization By Gate Merging) was developed to investigate different parameters to decide when to stop grouping connected gates of unitary fanout in order to replace them. Three parameters are considered, one related to the number of stacked transistors (called QMTS), other related to a maximal number of inverters included by the De Morgan transformations, and a last one related to the number of cells in a logical cone (called GMI). A greedy algorithm is implemented to apply one parameter at a time, achieving a reduction of up to

11% in transistor count for ITC99 circuits benchmark, and showing that the best results are achieved when QMTS parameter is used to perform clustering. It lacks the use of the total number of transistors as a direct quality metric.

It also belongs to the Netlist Rewriting approach the work (ROY; BHATTACHARYA; BOPPANA, 2005), in which a clustering process identifies the best candidate regions in the design for local optimization, in a static timing analysis driven search, and then replace the clusters cells by new cells, which they call flex cells, created for each respective timing contexts. The provided result is about the number of time-violating paths and the number of instances achieved with the application of their technique. It differs from our work by the adopted criteria to perform clustering and by the objective as our goal is to reduce the number of transistors.

The idea of merging cells and replacing them with a new complex gate is also applied in (GHANE; ZARANDI, 2016) to mitigate NBTI effect (*Negative-bias Temperature Instability*) in digital circuits. It identifies NBTI susceptible nodes in critical and non-critical paths, and then NBTI-sensitive gates – and only them – are combined to their driver gates and replaced by a new complex gate with the same logic. The proposed method of generating the new complex gate removes most of the PMOS transistors that were under severe NBTI stress.

Regarding the goal of reducing the number of transistors, the work of (MATOS et al., 2014) presents a tool to reduce the number of transistors when performing technology mapping to a library composed only by simple cells (nand, nor, inverters, and xors cells). Also, in (AMARÚ; GAILLARDON; MICHELI, 2013) it is presented a tool capable of producing area-efficient results for mixed XOR-AND/OR dominated logic functions with a two steps synthesis process, one responsible for finding in the circuit portions with potential to optimization and the second step is the remap itself. They report a respective reduction in the number of transistors and the number of devices of 18% and 9.2% on average compared to the state-of-the-art academic and commercial synthesis tool.

As we discuss further in the next chapters of this thesis, we present a netlist rewriting technique based on a cell grouping and replacement method to seek optimization of the transistor count.

3 MODELING AND ANALYSIS OF THE ALGORITHMS

In this chapter, we provide formal modeling of the problem, and we show the main algorithms used in this thesis, other auxiliary algorithms, as well as their analysis of complexity. We start by presenting the basic definitions and data structures, then we present and discuss the algorithms we developed to realize the proposed investigation of this thesis. We finish by discussing some underlying analysis of complexity.

3.1 Definitions

Let A a generic set and $|A|$ its size. We define $D(V, E)$ as a directed graph (digraph) where V is a set of vertices and E is a set of directed edges, as declared in Definition 3.1 and Definition 3.2.

Definition 3.1. V is a nonempty set of vertices $v_1, v_2, \ldots, v_n$ so that:

$$V = \{v_i \mid i \in [\,1, |V|\,]\ \}, V \neq \emptyset.$$

Definition 3.2. E is a set of directed edges (arcs) identified by an ordered pair of vertices (v_i, v_j), so that:

$$E = \{(v_i, v_j) \mid v_i, v_j \in V, (v_i, v_j) \neq (v_j, v_i)\}$$

In a digraph, each ordered pair $(v_i, v_j) \in E$ has only one direction from v_i to v_j, and we say that the arc (v_i, v_j) is divergent from v_i and convergent to v_j. We also say that v_i is the initial vertex of the arc (v_i, v_j), while v_j is its final vertex. The *input degree* of a vertex v_i is the number of edges convergent to v_i, while the *output degree* of v_i is the number of edges divergent from v_i. A vertex with a null input degree is called a *source*, and a vertex with a null output degree is called a *sink*. Consider the degree of the digraph as the highest input/output degree of its vertices.

A tree is a particular type of graph that is connected and has no cycles. An oriented tree, also called a *root tree* is a directed graph tree with a specified root vertex, such that each non-root vertex is the initial vertex of exactly one arc, and the root is the initial vertex of no arc (KNUTH, 1997). Therefore, for this definition, the root is a sink. The definition of root tree is necessary to the problem we are modeling because, generally speaking, this is the kind of structure we are interested to find in a circuit as we are modeling.

3.2 Adopted Modeling

We represent the netlist of a circuit as a digraph. Starting from a plain netlist (a netlist without internal modules), we can build an equivalent digraph with the procedure shown in Algorithm 1.

Algorithm 1 Building Digraph Algorithm

```
 1: function BUILDDIGRAPH(Netlist n)
 2:     let D(V,E) an empty digraph
 3:     V.add(new Vertex(input))                    ▷ A source vertex
 4:     V.add(new Vertex(output))                   ▷ A sink vertex
 5:     for all Gate g: n.getGates() do
 6:         V.add(new Vertex(g))
 7:     end for
 8:     for all Wire w: n.getNets() do
 9:         DRIVER $\leftarrow$ V.get(w.getDriver())
10:         for all Gate g: w.getSinks() do
11:             SINK $\leftarrow$ V.get(g)
12:             E.add(new Edge(DRIVER,SINK))
13:         end for
14:     end for
15:     return D(V,E)
16: end function
```

Notice that in the resulting digraph from Algorithm 1, all inputs in netlist are represented by a single source vertex, while all outputs in netlist are represented as a single sink vertex. It starts by adding a vertex for each gate from the netlist. Finally, it adds an arc for each wire linking the two corresponding vertices of each driver and sink gates. The direction of the edge is the same as the signal propagation in the circuit.

We are particularly interested in rooted trees that may appear in this digraph as sub-graphs. We call it a *group*, which corresponds to a set of connected gates of unitary fanout. Regarding its graph representation, a group is further defined as in Definition 3.3.

Definition 3.3. A group is a subgraph $D'(V', E')$ of $D(V, E)$, where $V' \subset V$ and $E' \subset E$, for which both the connectivity and uniqueness properties are valid. We stand connectivity and uniqueness as follows:

Connectivity) For each vertex v_i from V', exists an edge of E' divergent from v_i that links v_i to a different vertex $v_j \in V'$. Formally:

$$\forall v_i \in V', \exists (v_i, v_j) \in E', v_j \in V', v_i \neq v_j$$

Uniqueness) The output degree of each vertex v_i of V' is 1. That is, exists only one divergent edge from v_i. It is also valid for output vertex, whose divergent edge connects it to a vertex in the digraph netlist, but outside the group. Formally, it is valid that:

$$\forall v_i \in V', \exists | (v_i, v_j) \in E'$$

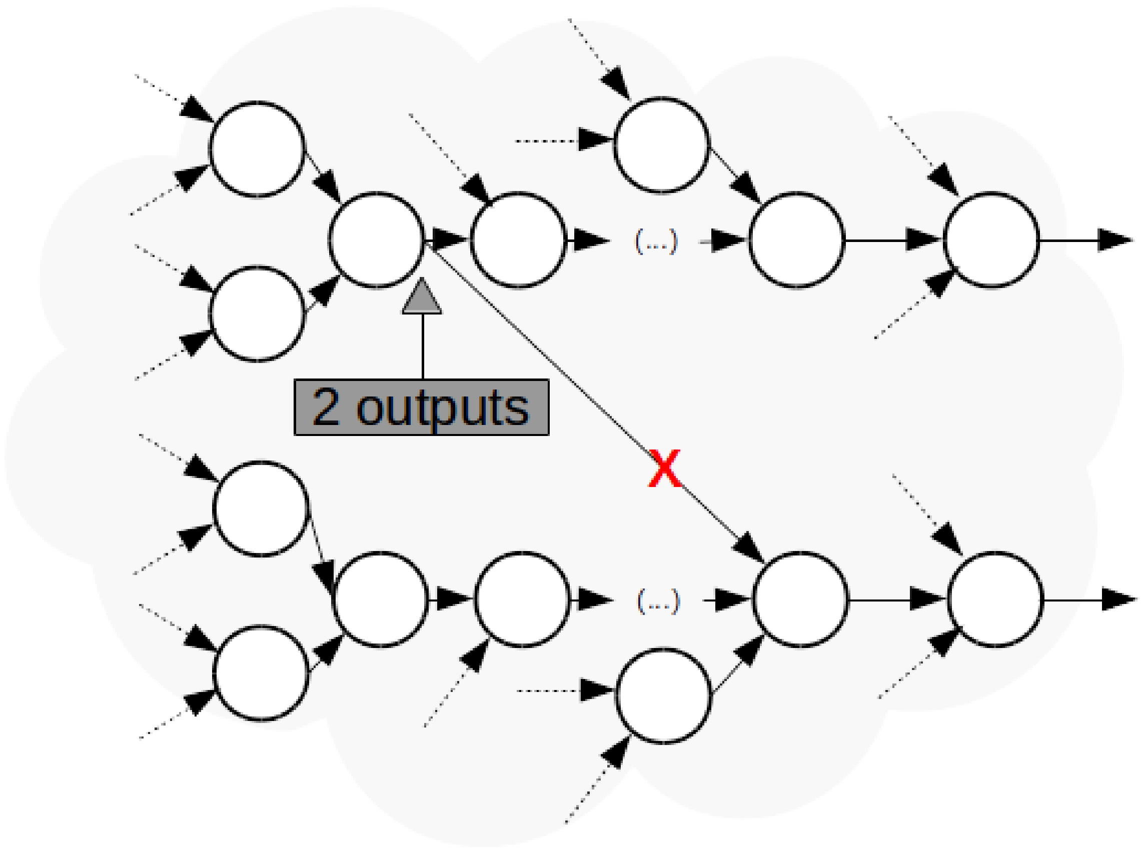
We call an input of the group an arc that is convergent to a vertex in the group, but is divergent from a vertex outside the group. And we call the output arc the one that is divergent from a vertex in the group but convergent to a vertex outside the group.

The root vertex of an identified group corresponds to the output vertex of a group. It follows that a group always has only one output, which can be easily demonstrated by contradiction. Assume that a group has more than one output gate (i.e., two root vertices). Both root vertices are not directly connected to each other; otherwise, they would violate the uniqueness principle. Also, no other vertex could be connected to both outputs for the same reason. Therefore, this group, with two output vertices, would violate the connectivity principle, as illustrated in Figure 3.1. The full edges are the internal edges of the group, and the dotted edges connect the vertices in the group with other vertices from the digraph netlist.

Also, a group does not contain any cycle. It also can be demonstrated by contradiction. Assume that a group contains a cycle. To have that cycle, a vertex v_i should have an output arc connected to another vertex that drives v_i (directly or not), called a feedback arc. In consequence, since v_i has output degree 1 by definition, unless v_i is an output vertex, the group is unconnected, thus violating its definition. However, if v_i is an output edge, the feedback edge is in the last vertex of the cluster chain, so such group has either no output, thus violating the claim that any group has always one output vertex, or v_i has two divergent edges, which violates the uniqueness principle. Figure 3.2 illustrates this claim.

Another data structure also based on the concept of rooted trees appears in works related to FPGA mapping minimization (CONG; DING, 1994), known as Fanout Free Cones (FFC). This structure differs from the one we are using because the FFC structure allows a cell with fanout bigger than one since its sink cells are inside the FFC structure. Also, the authors adopt the notion of bounded FFC that is related to the maximum number of inputs of a boolean function of a node, which is needed in order to fit the number of inputs allowed by the FPGA Look Up Tables (LUTs). For the ASIC application we are proposing, the number of stacked transistors demanded to implement the logic function of an identified group is a more valuable parameter.

Figure 3.1: A group has no cycle.

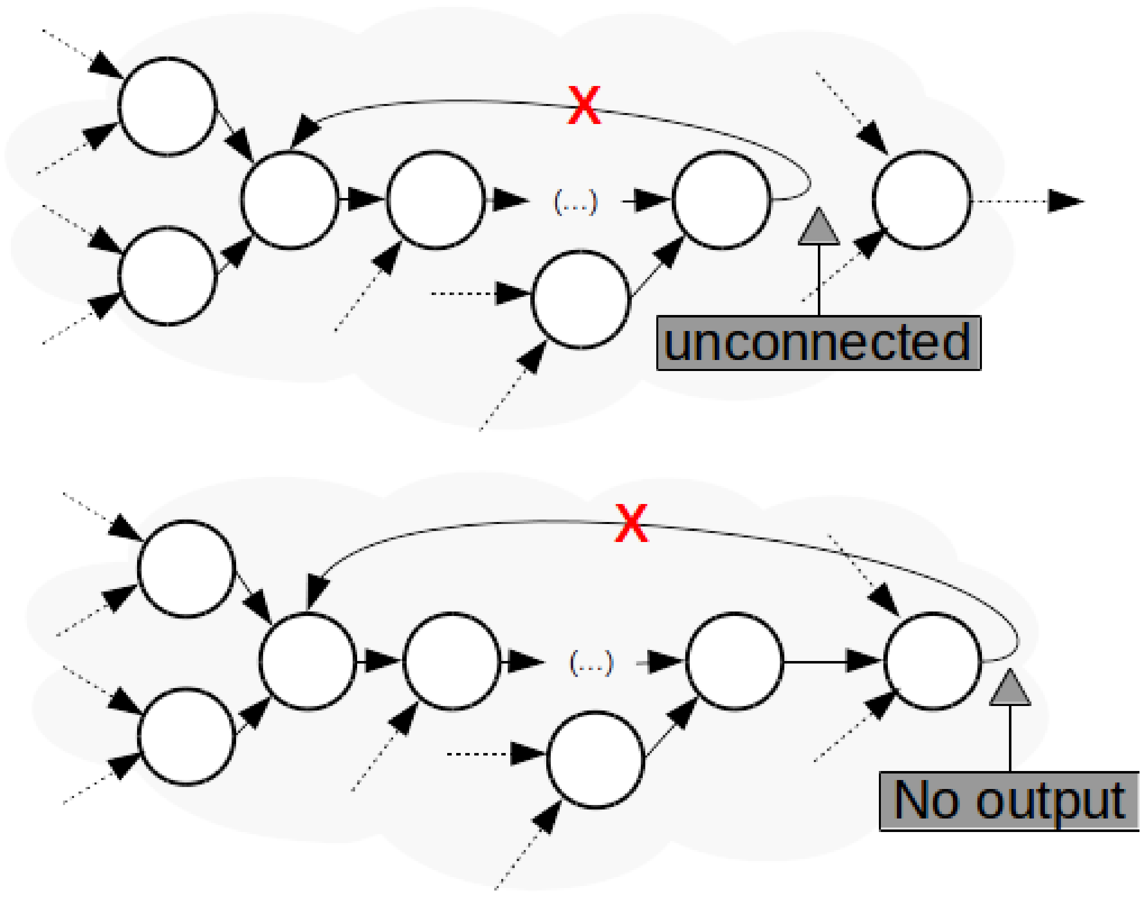


Source: own authorship

Algorithm 2 Extracting the function of the group

```
function BUILDFUNCTION(Group g)
    INPUTS = g.getInputs()
    while g.getvertices().size()>1 do
        for all Input i: INPUTS do
            for all Vertex v: i.getSinks() do
                FUNCTION = v.getFunction()
                for all Literal lit: FUNCTION.getLiterals() do
                    FUNCTION.replaceLiteral(lit, v.getInputConnectedTo(var))
                end for
                v.setOutput(FUNCTION)
            end for
            g.removeAllVertices(i.getSinks())
        end for
        INPUTS = g.getInputs()
    end while
    return FUNCTION
end function
```

Figure 3.2: Graphic demonstration that a group has no cycles.



Source: own authorship

Since each gate of a group corresponds to a specific boolean function, we can compose them to find a single boolean function that represents the whole group. We assign a literal to every input arc of the group. The function can be constructed following the procedure shown in Algorithm 2. It works by incrementally replacing each literal in the boolean function of the gate by the aggregate function of the gate that drives the literal's corresponding input, until it reaches the wires that are inputs and output for the group.

3.3 Proposed Methodology

We wrote three main algorithms to identify groups in a digraph representation of a netlist. We call them UNBOUNDED, BOUNDED, and OPTIMIZED procedures. The first procedure is presented in Algorithm 3. The Unbounded algorithm is, in essence, a breadth first search algorithm that starts with a seed, which is a gate of unitary fanout, and generates a maximal group that is bounded by cells with fanout bigger than one. In other words, this algorithm generates a group as large as possible. However, the new complex gate that should be generated to replace the group of cells may not be feasible due to a recommended limitation in the number of stacked transistors (SCHNEIDER, 2007). On the other hand, this algorithm is quite useful in helping to set an upper bound for the amount of optimization our technique is able to achieve.

Algorithm 3 Maximal group finding Algorithm

```
 1: function UNBOUNDEDGROUPING(Gate seed)
 2:     gateQueue.add(seed)
 3:     while !gateQueue.allVisited() do
 4:         first = gateQueue.getFirst()
 5:         first.setVisited()
 6:         group.add(first)
 7:         for all Gate g : first.getConnectedGates() do
 8:             if g.getFanout() == 1 then
 9:                 if !g.isVisited() then
10:                     gateQueue.add(g)
11:                 end if
12:             end if
13:         end for
14:     end while
15:     return group
16: end function
```

Restrictions can be added to the UNBOUNDED algorithm in order to impose limitations during the grouping procedure, resulting in the greedy BOUNDED procedure presented in Algorithm 4. In this version, we set restrictions in the number of serial transistors (S_LIMIT) and in the total number of transistors (T_LIMIT) the new SCCG can have. So, for a cell of unitary fanout given as seed, the algorithm outputs a group of cells connected to the seed whose the new complex gate that will replace them is always feasible.

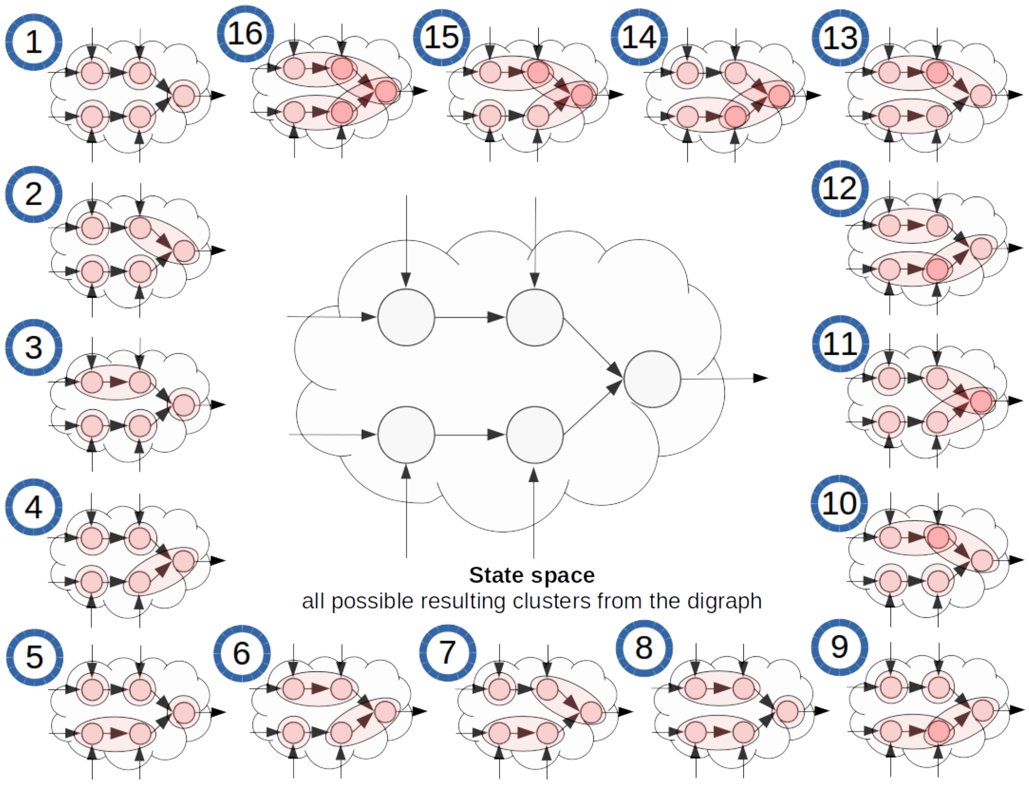
Algorithm 4 Restricted grouping Algorithm

```
function BOUNDEDGROUPING(Gate seed)
    gateQueue.add(seed)
    while !gateQueue.allVisited() do
        first = gateQueue.getFirst()
        first.setVisited()
        temp = group
        temp.add(first)
        function = buildFunction(temp)
        if function.getNumOfSerialT() > S_LIMIT then
            continue
        end if
        if function.getNumOfT() > T_LIMIT then
            continue
        end if
        group.add(first)
        for all Gate g : first.getConnectedGates() do
            if g.getFanout()!= 1 then
                continue
            end if
            if g.isVisited() then
                continue
            end if
            gateQueue.add(g)
        end for
    end while
    return group
end function
```

On the other hand, the greedy approach adopted in Algorithm 4 presents final solutions that are not necessarily optimized regarding the number of transistors, although leading to SCCGs that are feasible to be implemented, as it is shown in (CONCEIÇÃO et al., 2017). That is because different groups of connected cells can be selected to compose a group to be replaced by a new SCCG, depending on the seed.

To illustrate the optimization demanded, Figure 3.3 shows the state space of a problem instance with five gates, with all possible group combinations. Each group com-

Figure 3.3: State space of a problem instance.



Source: own authorship

bination we call a *cover*. The group of connected cells of unitary fanout is presented in the center of the figure, while all the possible covers are presented in the borders. The challenge is to select the cover that will lead to the smallest number of transistors. Each smaller group of cells in a cover is a candidate to be replaced by a single SCCG with an equivalent function. An ideal algorithm should consider all the state space to select the best cover.

We designed the Algorithm 5 as part of the OPTIMIZED procedure to select groups of cells. It presents an exhaustive routine to find all possible covers of a given group of cells. The auxiliary procedures *unite*, *combine*, *uniteCover*, and *combineCover* are presented in Algorithm 6. It works bottom-up recursively, applying the procedures *unite* and *combine* to combine the set cover of each child gate with the parent gate, and then applying the procedures *uniteCovers* and *combineCovers* on these resulting set of covers of all children vertices to find the combinations of the subgroup rooted in the parent vertex.

For a better understanding of the procedure to find all possible covers, Figure 3.4 illustrate its general behavior. The labels marked in the figure indicate the result of the corresponding procedure. The algorithm receives as input a rooted tree (i.e., group)

Algorithm 5 Algorithm for exhaustive generation of all covers

```
function FINDALLCOVERS(Gate root)
    if root.isLeaf() then
        group.add(root);                    ▷ create a unitary group with root
        cover.add(group);                   ▷ the trivial cover
        allCovers.add(cover);               ▷ only one possible cover
        return allCovers;
    else
        for all Gate child : root.getChildren() do
            childAllCovers = findAllCovers(child);    ▷ combining root with possible covers of child
            for all Set< Set<Gate> > childCover: childAllCovers do
                united.add(unite(childCover, root));
                combined.add(combine(childCover, root));
            end for
            if allCovers.isEmpty() then     ▷ happens for the first child
                allCovers.addAll(united);
                allCovers.addAll(combined);
            end if
            for all Set< Set<Gate> > temp: allCovers do    ▷ now combining with combination of previous child
                for all Set< Set<Gate> > u: united do
                    mixed.add(uniteCovers(temp, u));
                end for
                for all Set< Set<Gate> > c: combined do
                    mixed.add(combineCovers(temp, c, root));
                end for
            end for
            allCovers = mixed;
        end for
        return allCovers;
    end if
end function
```

Algorithm 6 Auxiliary algorithms for exhaustive generation of all covers

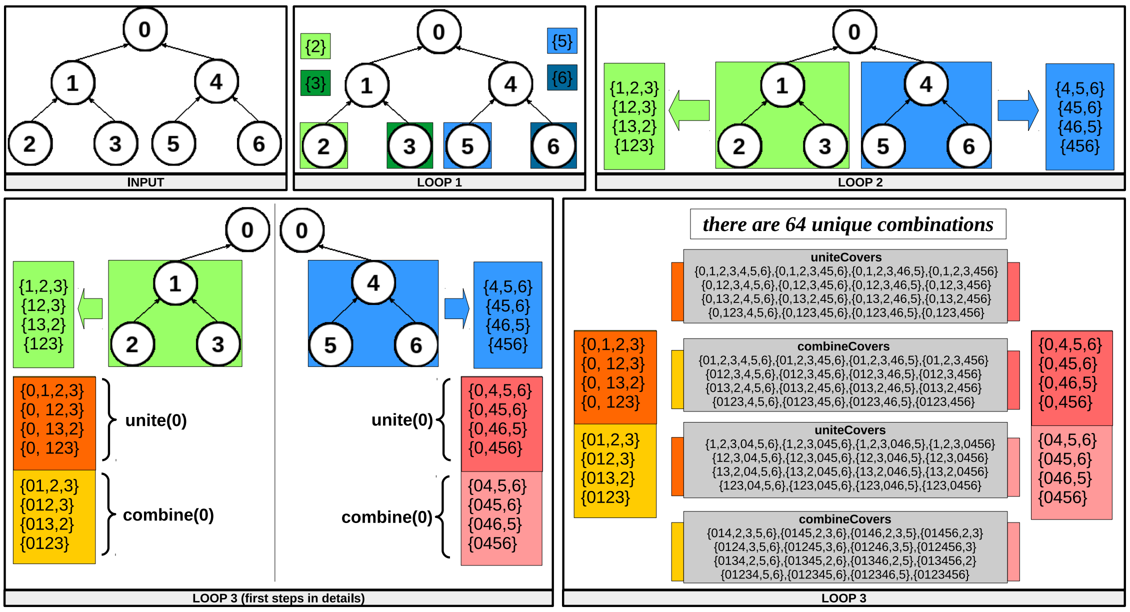
```
function UNITE(Set< Set<Gate> > cover, Gate root)
    group.add(root)                      ▷ create a unitary group with root
    cover.add(group)                     ▷ add root's group in the cover
end function
function COMBINE(Set< Set<Gate> > cover, Gate root)
    for all Set<Gate> group : cover do   ▷ search in all groups in cover...
        for all Gate g : group do        ▷ which one contains a root's child.
            if root.hasChild(g) then
                group.add(root)          ▷ then add root to that group
                return cover
            end if
        end for
    end for
end function
function UNITECOVERS(Set< Set<Gate> > rootCover, Set< Set<Gate> > childCover)
    rootCover.addAll(childCover);        ▷ just unite both covers
    return rootCover;
end function
function COMBINECOVERS(Set< Set<Gate> > rootCover, Set< Set<Gate> > childCover, Gate root)
    for all Set<Gate> groupRoot : rootCover do
        if groupRoot.contains(root) then
            for all Set<Gate> groupChild : childCover do
                if groupChild.contains(root) then
                    groupRoot.addAll(groupChild)
                    combinedCover.add(groupRoot)
                else
                    combinedCover.add(groupChild
                end if
            end for
        else
            combinedCover.add(groupRoot)
        end if
    end for
    return combinedCover
end function
```

of size 7 with output in vertex in 0 (i.e. rooted in 0). In the first loop, the algorithm associates the trivial cover to the leave vertices 2, 3, 5 and 6. In the second loop, the algorithm merges the covers of leave vertices with their respective parents using unite and combine operations. So, for vertex 2 and parent 1, the possible covers are vertex 1 and 2 separated (as a result of *union* operation, we write 1,2), and vertex 1 and 2 as a group (as a result of *combine* operation, we write 12). As the set of covers in 1 is empty, these two covers become the partial cover of vertex 1. In the same way, the algorithm applies the *unite* operation over the unitary set of covers of vertex 3 and the parent vertex 1 (resulting in cover 1,3), and then apply the *combine* operation over them (resulting in cover 13). Now each partial cover must be combined with each other partial cover of 1 using *uniteCover* and *combineCover* operations, as a Cartesian product of the two sets. We obviously eliminate repeated vertices, thus resulting in the covers 1,2,3, 13,2, 12,3 and 123. The same procedure can be extended to find all possible covers for sub-tree rooted in node 4.

Looking at the box with the first steps of LOOP 3 in Figure 3.4, the partial covers in orange and red show the result of *unite* operation over vertex 0 and the covers of sub-trees rooted in vertex 1 and vertex 4, respectively. Then the *combine* operation is applied over vertex 0 and covers of sub-trees rooted in vertex 1 and in vertex 4 to generate partial covers in yellow and pink, respectively. The final step is to combine these covers as expressed in the box labeled LOOP 3, to generate all possible covers for the groups, as the algorithm achieves the root vertex 0. There are 64 possible covers for this example, among which the best is the one that leads to the smallest total number of transistors when their groups of cells are replaced by one new cell.

When selecting the best cover, it is also necessary to consider the number of inverted signals demanded by a given cover. The number of inverted signals in the new SCCG can influence the number of transistors, since some inverting gates may have to be inserted to guarantee the logical integrity of the replacement. We can avoid this insertion by looking in the circuit for the inverted signals demanded by the new gate, or by inverting the logic of the signal driving cells up to reach a temporal barrier. Algorithm 7 considers these issues when it takes the result of Algorithm 5 as input to identify which cover leads to the smallest number of transistors. As we further discuss in the next section, the number of possible covers increases exponentially with the number of vertices in the group. Nevertheless, we demonstrate in this thesis that it still can be useful to find optimal solutions when the number of vertices is small.

Figure 3.4: Steps of the procedure to find all possible covers of a groups of seven connected cells.



Source: own authorship

Algorithm 7 Algorithm to find best cover

```
1:  function FINDBESTCOVER(Set< Set< Set<Gate> > > allCovers)
2:      smallest = ∞
3:      for all Set <Set<Gate> > cover : allCovers do
4:          numOfT = 0
5:          for all Set<Gate> group : cover do
6:              function = buildFunction(group)
7:              numOfT = numOfT + function.getNumOfT()
8:              for all Signal s : function.getInvertedInputs do
9:                  if !global.circuit.contains(s) then
10:                     numOfT += 2*function.getNumOfInvertedInputs()
11:                 end if
12:             end for
13:         end for
14:         if numOfT < smallest then
15:             smallest = numOfT
16:             best = cover;
17:         end if
18:     end for
19:     return best
20: end function
```

We combine the tree UNBOUNDED, BOUNDED, and FindAllCovers procedures into a new algorithm to group cells, which is shown in Algorithm 8. The OPTIMIZED algorithm works by first finding the maximal group of connected gates of unitary fanout for a given seed, and either applying the BOUNDED algorithm when the number of cells is bigger then MAX_NODES parameter, or performing the EXHAUSTIVE algorithm otherwise. The value of this parameter will depend on a trade-off between execution speed and achieving better results.

Algorithm 8 Optimized algorithm for grouping cells

```
 1: function OPTIMIZEDGROUPING
 2:     all = findAllUnitaryFanoutGates()        ▷ returns the set of gates of unitary fanout
 3:     while !all.empty() do
 4:         seed = all.getFirst()
 5:         group = unboundedGrouping(seed)
 6:         if group.size() <= MAX_NODES then
 7:             root = group.getRoot();
 8:             best = findBestCover(getAllCovers(root))
 9:             for all Set <Set<Gate> > cover : best do
10:                 allGroups.addAll(cover)
11:                 for all Set<Gate> group : cover do
12:                     all.removeAll(group)
13:                 end for
14:             end for
15:         else
16:             group = boundedGrouping(seed)
17:             allGroups.add(group)
18:             all.removeAll(group)
19:         end if
20:     end while
21:     return allGroups
22: end function
```

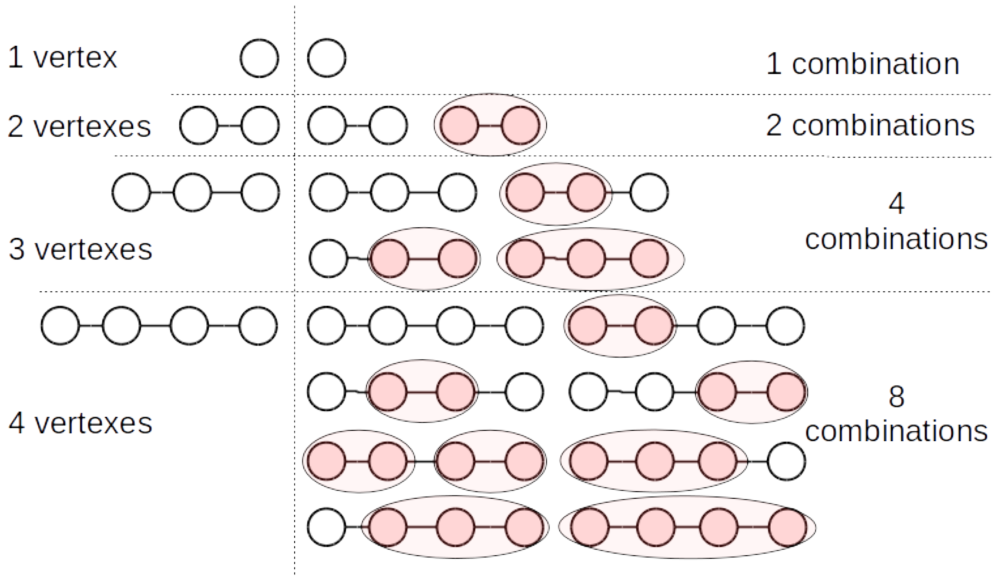
Finding the best combination can lead to a circuit that uses fewer transistors than the set of gates it replaces in the circuit, as well as fewer wires and, consequently, fewer vias when doing the physical design.

3.4 Analysis of Complexity

The number of combinations of a group of gates of size N with maximum input degree equals to d is $O(d^{N-1})$. That can be proved by structural induction. To start this analysis, consider a group whose maximum input degree is two (remember that all vertices have output degree one, by definition). Also, without loss of generality, consider the digraph of the cluster without the direction of its edges. That the cluster will look like a trivial tree of size N.

Figure 3.5 shows all possible combinations in a group with up to four vertices. The possibilities include not to group any vertex, group all vertices together, or group some portions of them. We can notice that the number of combinations for these cases is equal to 2^{N-1}, where N is the number of vertices in the tree. Take this as the base step of the demonstration.

Figure 3.5: All possible combinations of clusters with up to 4 vertices.



Source: own authorship

As an inductive hypothesis, assume that the equation 2^{N-1} is also valid as a superior asymptotic limit ($O(2^{N-1})$) to the number of possible combinations of vertices in all groups up to size N. Now, consider the group of size $N+1$. This additional vertex will combine with each combination in two possible ways: either by grouping with its adjacent cell (or group of cells) or by sticking together as a single cell without a group. In other words, the addition of a new cell in the group will double the possible combinations. By the inductive hypothesis, the number of combinations in a group of size N is 2^{N-1}, so the number of combinations in a group of size $N+1$ is $2 \times 2^{N-1} = 2^N$

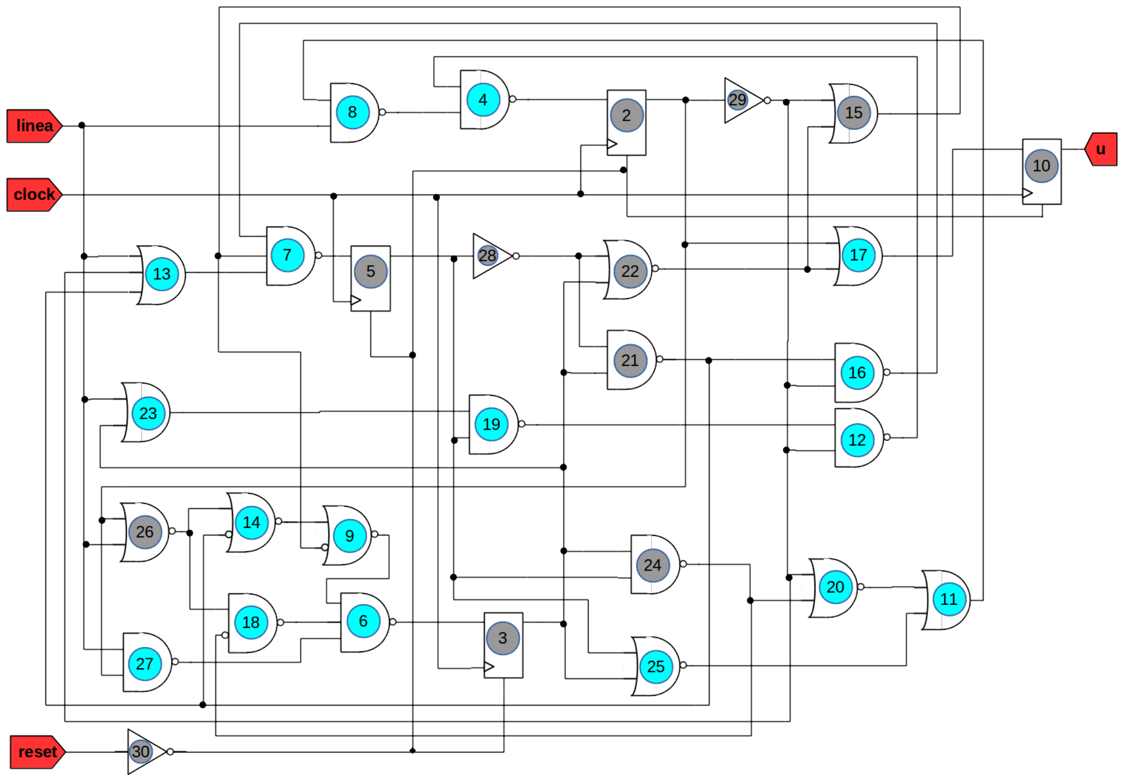
Finally, it is essential to relax the assumptions made at the beginning of the analysis to include groups of input degree bigger than two. Notice that a vertex with degree d in the group is possible to combine with as many cells as its degree. It explains the d as the basis of the equation in the asymptotic function. Notice yet that it is a quite pessimistic estimation since hardly all the vertices in a cluster will have the same high degree. Despite of this fact, the state space can be huge as the number of cells in a cluster increases.

4 A FULL EXAMPLE OF THE PROPOSED METHOD

In this chapter, we show in detail an example of the proposed optimization performed over the B02 circuit from the ITC99 benchmark, as well as some elementary discussions about the proposed cell merging methodology.

Figure 4.1 shows the plot of the original netlist generated by a commercial synthesis tool targeting a 180 nm vendor library, while its equivalent digraph view of the circuit is in Figure 4.2. The gates in the circuit are labeled and highlighted with the same colors as in the digraph, where light blue nodes are gates with unitary fanout, and the gray vertexes represent gates with fanout bigger than one or represent flip-flops. The red nodes represent inputs and outputs of the circuit.

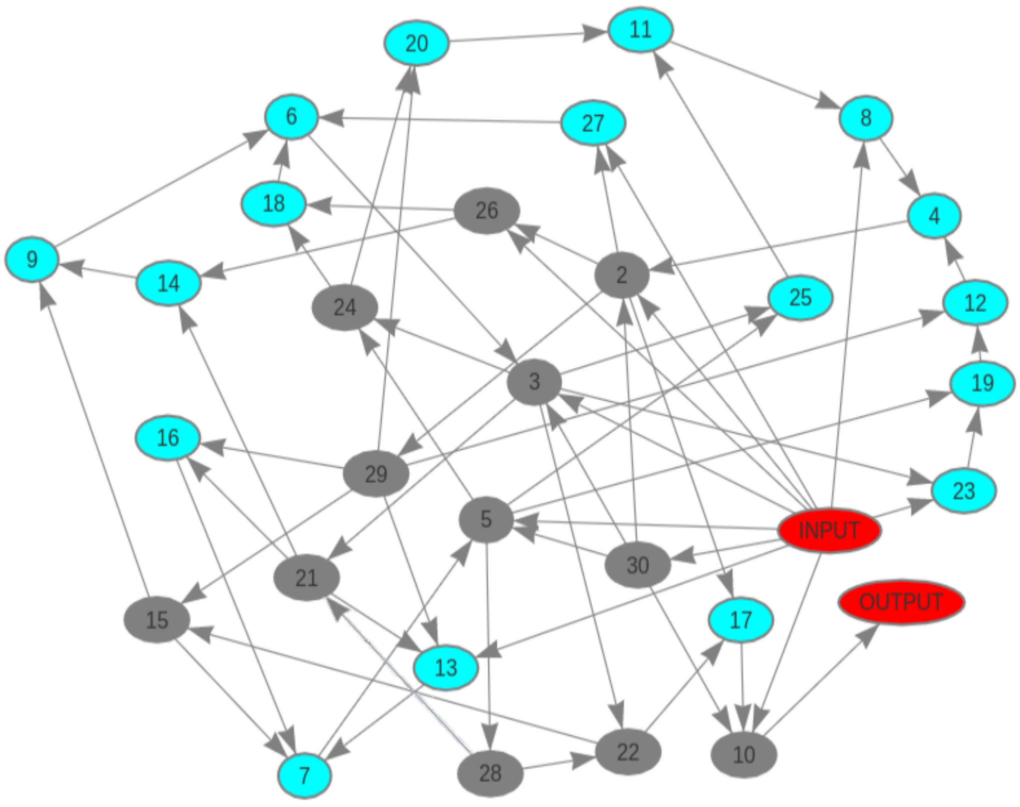
Figure 4.1: Plot of a netlist of B02 circuit from ITC99 benchmark



Source: own authorship

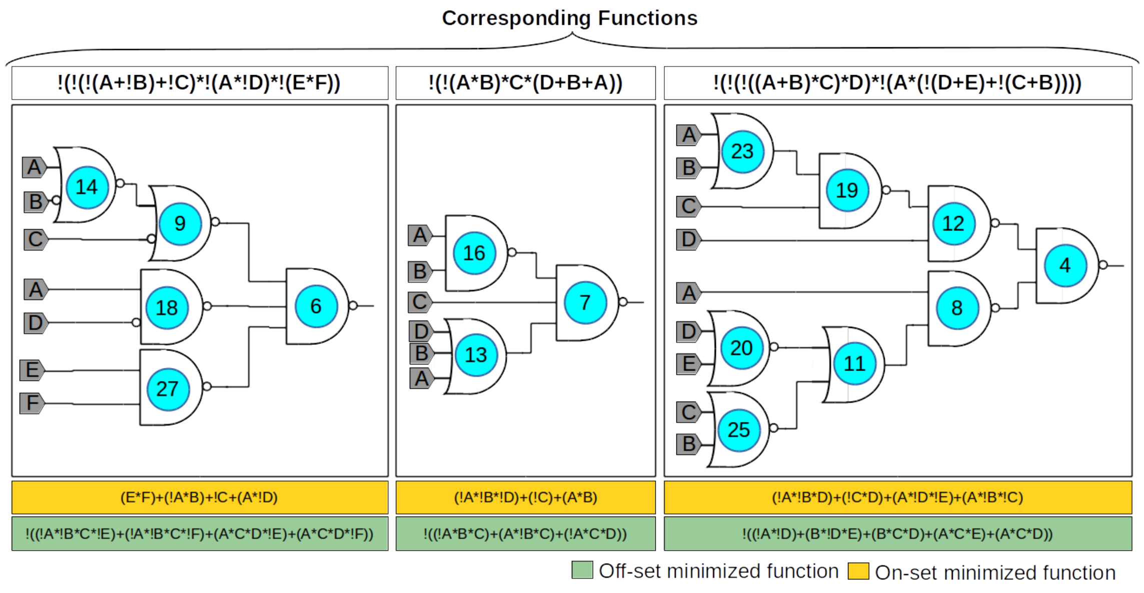
There are, in essence, four groups of connected combinational gates of unitary fanout in this circuit: an unitary group formed by gate 17; one formed by gates 6, 9, 14, 18 and 27; a third formed by gates 7, 13 and 16; and the biggest one formed by gates 4, 8, 11, 12, 19, 20, 23, 25. Except for the unitary group, these portions of the circuit are shown in Figure 4.3.

Figure 4.2: Digraph of the B02 circuit netlist from ITC99 benchmark



Source: own authorship

Figure 4.3: Identified portions of connected unitary fanout gates from B02


Corresponding Functions
!(!(!(A+!B)+!C)*!(A*!D)*!(E*F))
!(!(A*B)*C*(D+B+A))
!(!(!((A+B)*C)*D)*!(A*(!(D+E)+!(C+B))))
A
B
C
14
9
A
D
18
6
E
F
27
A
B
C
D
B
A
16
13
7
A
B
C
D
23
19
12
4
A
D
E
C
B
20
25
11
8
(E*F)+(!A*B)+!C+(A*!D)
(!A*!B*!D)+(!C)+(A*B)
(!A*!B*D)+(!C*D)+(A*!D*!E)+(A*!B*!C)
!((!A*!B*C*!E)+(!A*!B*C*!F)+(A*C*D*!E)+(A*C*D*!F))
!((!A*B*C)+(A*!B*C)+(!A*C*D))
!((!A*!D)+(B*!D*E)+(B*C*D)+(A*C*E)+(A*C*D))
🟩 Off-set minimized function
🟨 On-set minimized function

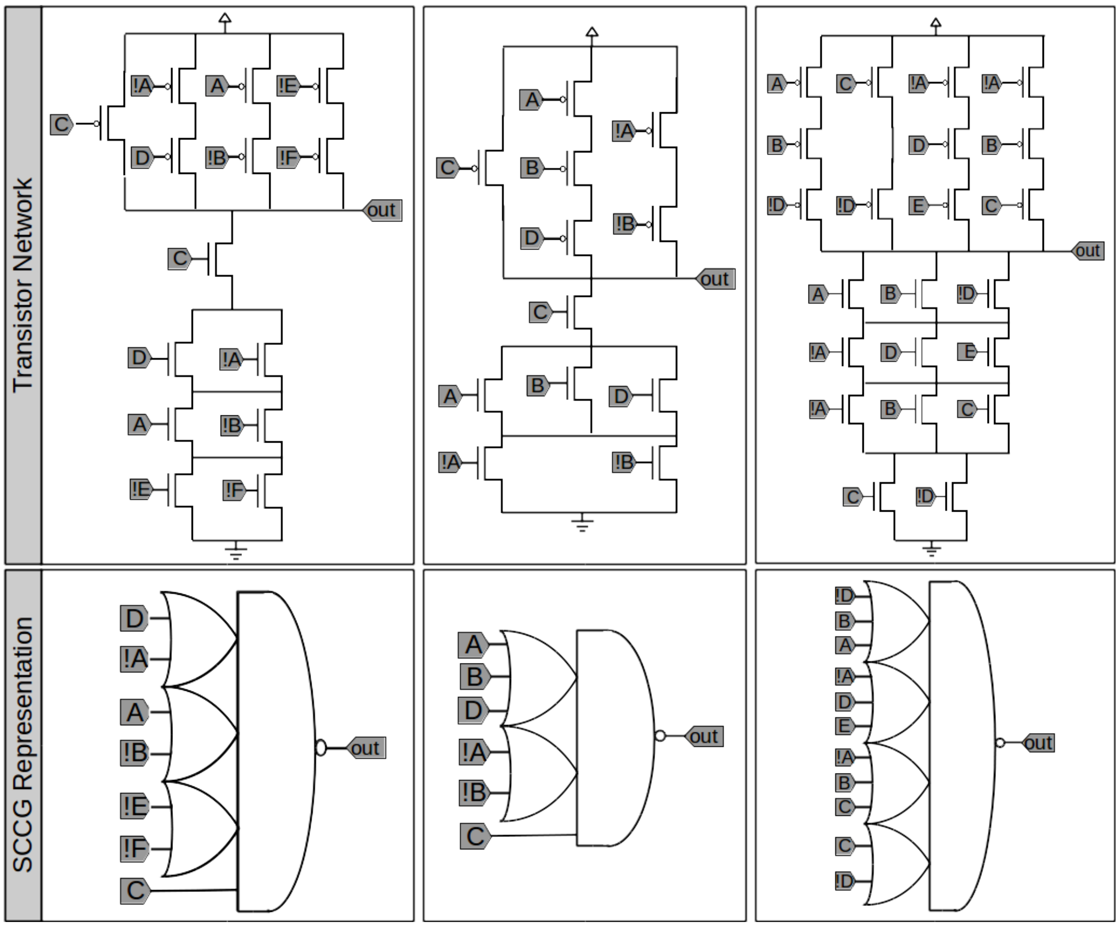
Source: own authorship

For a given group, the corresponding equation is extracted and then minimized using boolean factoring, targeting to reduce the number of literals (MARTINS et al., 2010).

We consider both on-set and off-set equations during minimization in order to get the expression with the smallest number of literals, as it will lead to a smaller transistor network – the aim of this thesis. Next, we build a transistor network for this expression. Sizing this transistor network is an important issue, but, for simplification, we keep all transistors with the smallest size allowed by the adopted vendor technology.

The identified groups and their respective expressions derived from on-set and off-set equations are shown in Figure 4.3 while resulting in transistor networks and their respective SCCG representation are shown in Figure 4.4. In this example, we consider the on-set equations for every group.

Figure 4.4: Transistor networks and their respective SCCG for the groups in B02



Source: own authorship

Notice that some of the inputs of the new gates are inverted from its first polarity. So, in order to guarantee the logical equivalence of the resulting circuit with the original, some options are possible to handle this situation:

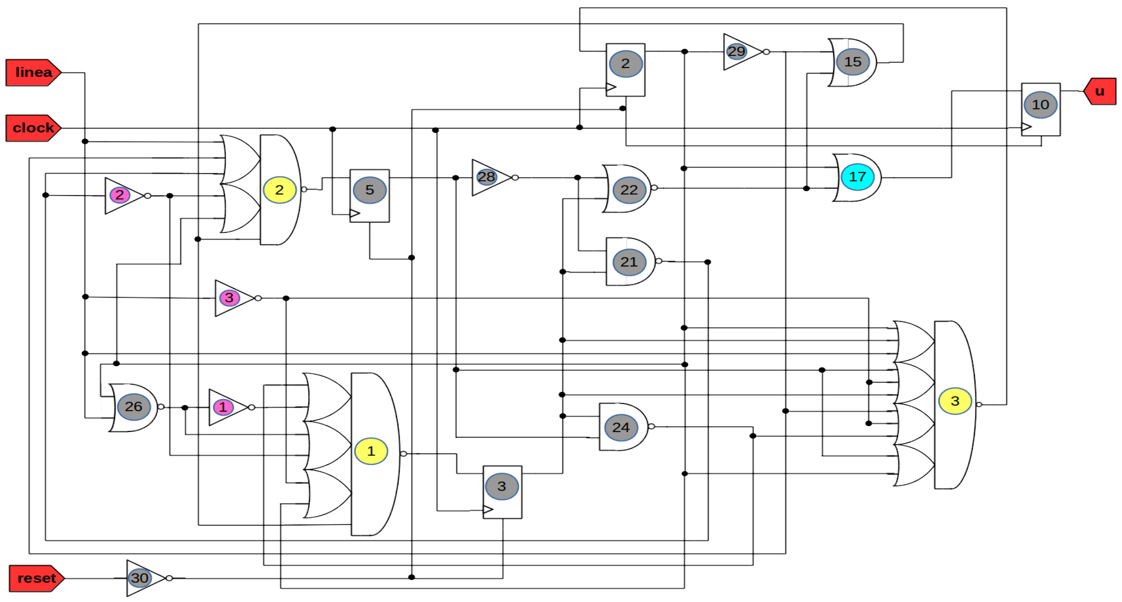
- if such an inverted signal is available in the circuit, which is possible if it drives (or it is driven by) an inv gate, then the signal can come from there;

- if it is driven by a flipflop, so the signal can come from its inverted Q output;
- If none of the previous possibilities are true, then a new inv gate is inserted.

Another possibility would be to perform De Morgan transformations in the logic cone that drives that signal until reaching the primary inputs or temporal barriers. However, it can be a hard computational task since the gates that drive the inputs of the group necessarily have fanout bigger than one, and it could interfere in the polarity of other inputs of this group and other groups in the circuit, in a cascade effect. So, we do not perform this option.

When the new gates presented in Figure 4.4 replace the circuit portions shown in Figure 4.3, it results in the circuit shown in Figure 4.5, where gates marked in yellow are the new SCCGs, and the pink ones are extra inverters added to guarantee the logic equivalence. The signal *linea* drives SCCG number three with both polarities so that an inverter is inserted. The same situation occurs to the output of gate twenty-nine, but in this case, no inverter is necessary since its input is used. Similar situations occur when replacing the other SCCGs, and new inverters are added only when it is strictly necessary.

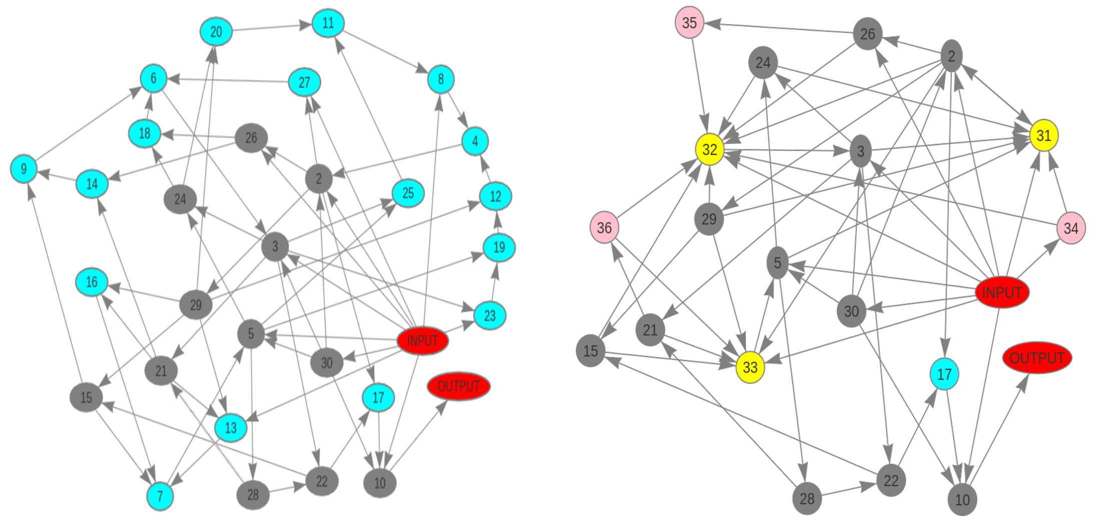
Figure 4.5: Circuit after proposed transformations



Source: own authorship

It is also necessary to notice that inverters are demanded to negate some signals in the original circuit. In Figure 4.6 it is shown the digraphs of the circuit before and after the optimization procedures described, where yellow vertexes are SCCGs, and pink vertexes are extra inverters that were added.

Figure 4.6: A digraph of a circuit before (left) and after (right) the merging procedure being applied

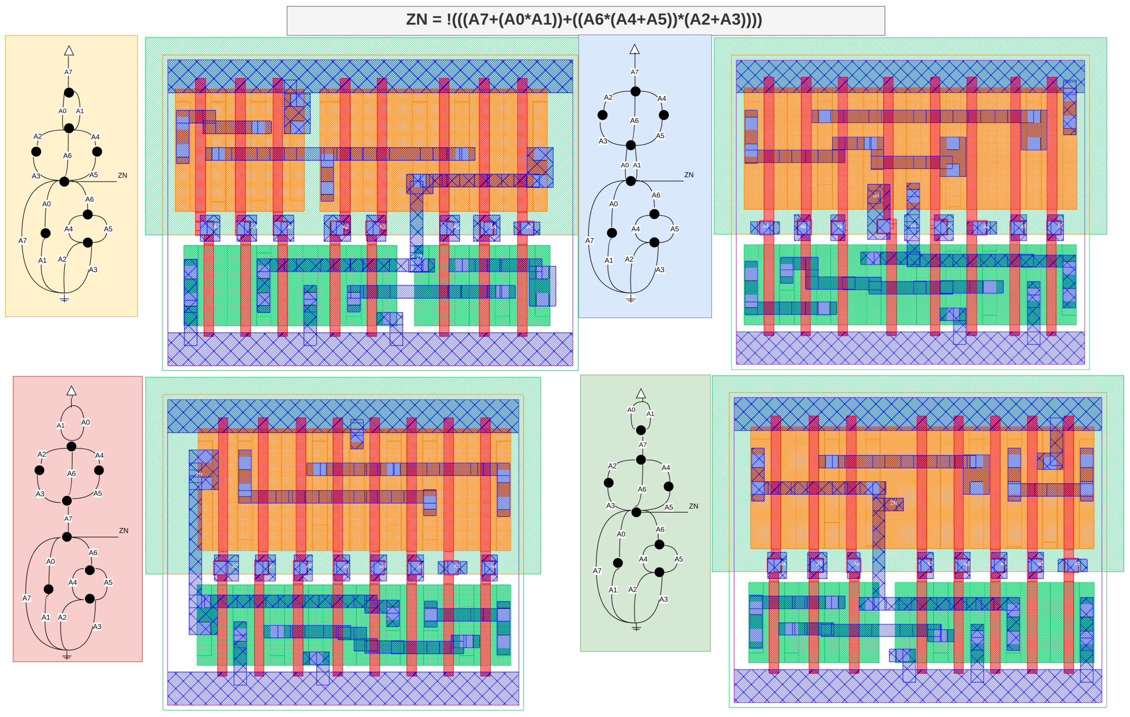


Source: own authorship

The group substitutions made for this final circuit not necessarily are the best choices to lead them to a minimal amount of transistors. It was not the case in this simple example, but replacing the whole group of cells by a single gate may lead to a network with an undesirable structure, for instance, with more than four stacked transistors. Also, it is possible that merging and replacing subsets of the maximal group instead of merging and replacing the whole group can lead to a solution with fewer transistors, since the number of demanded inverters due to inverted inputs influences the total transistor count.

Nevertheless, by doing the substitutions shown, the transistor count in the whole circuit is reduced by 21%, and the number of wires segments is reduced by 32%. Notice that all the remaining wires and cells in the circuit remain untouched. The proposed optimization procedure enables the exploration of distinct transistor arrangements and different layouts, as exemplified in Figure 4.7. The optimization opportunity is dependent on an efficient layout generator tool to make the exploration feasible during design time. These explorations are beyond the scope of this thesis, however.

Figure 4.7: Example of layouts automatically generated from distinct transistor arrangements for the same function.



Source: own authorship

5 EXPERIMENTS AND RESULTS

We organize the set of experiments and results done by us into six sections. The first aims to describe the setup of the experiments we perform in order to make the assumptions clear and help those who want to reproduce them. The second group corresponds to a preliminary analysis, focusing on studying the potential impact of the application of our technique. The third one focuses on evaluating the proposed technique regarding the parameters we want to optimize, in the broader scope of the feasibility of the proposal. In the fourth one, we compare the algorithms we proposed, collecting and presenting results about the parameters we optimize. The fifth section aims to evaluate the gains in physical synthesis, showing estimations of the impact of our technique over area and connectivity. In the sixth section, we compare our technique with other similar works regarding the performance parameters we have adopted.

The experiments presented here were performed using our EDA tool developed to support this thesis. The tool was developed in C++ and used external libraries like the API (*Application Program Interface*) from ABC to perform some minor logic transformations; the Open Source Liberty parser from E-Tools; and an open-source Verilog parser[1] developed with flex and YACC; besides other reading, converting and writing tools we wrote for genlib, eqn and SPICE file formats. Figure 5.1 summarized the whole automatic procedure as it is implemented in this thesis proposal.

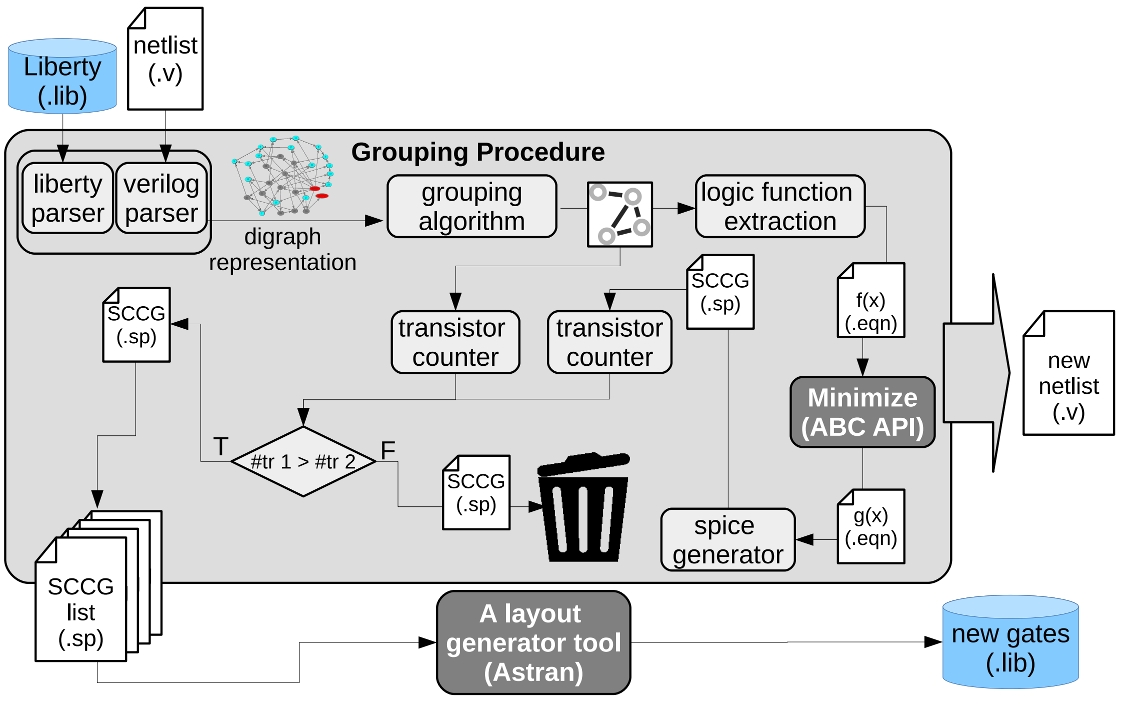
5.1 Setup of the experiments

The experiments presented in this chapter used cell libraries in genlib format[2], whose contents we describe as follows. These libraries appear in the related literature, and they are also available in the ABC download package. We sort the libraries by the number of logic functions, and we can roughly say that each library contains all the functions of its predecessor.

- **minimal.genlib**: logic0, logic1, inv1, nand2, nor2 (5 logic functions);
- **nand-nor.genlib**: logic0, logic1, inv1, nand2, nand3, nand4, nor2, nor3, nor4 (9 logic functions);
- **22.genlib**: logic0, logic1, inv1, nand2, nor2, aoi21, oai21, oai22, aoi22 (9 logic

[1]Available in https://github.com/ben-marshall/verilog-parser
[2]Specification available in https://www.ece.cmu.edu/ ee760/760docs/genlib.pdf

Figure 5.1: General steps, intermediary files, data structures, and outputs of the proposed method



Source: own authorship

functions);

- **33.genlib**: logic0, logic1, and all possible functions using up to 3 (three) stacked transistors (89 logic functions);
- **44.genlib**: logic0, logic1, and all possible functions using up to 4 (four) stacked transistors (3505 logic functions);

Besides those libraries, we use cadence.genlib library, which contains the set of gates from Cadence Design Systems generic library. We wrote a Liberty counterpart for all libraries previously mentioned based on the Nangate Open Cell Library 45 nm (we refer as NOCL45nm) functional liberty file, by keeping in the new liberty file the cell declaration of the minimum size for each logic function in the related .genlib, together with one declaration of a D type flip-flop. Also, an additional .genlib file was written for the library NOCL45nm, containing only its combinational cells (90 cells, 33 logic functions) except for the arithmetic ones, since .genlib format does not support cells with two outputs.

The circuits B01 to B12 from ITC99 benchmark[3] were selected to perform the proposed experiments. We choose this benchmark due to the small sizes of the circuits,

[3]Available in <https://github.com/squillero/itc99-poli>

to favor a more detailed analysis. The description of each circuit is in Table 5.1. Unless it is explicitly mentioned, every input netlist is generated using ABC logic synthesis tool using the following steps:

- read the target library file in *genlib* format using the command *read_library*;
- read the circuit description in *bench* format as provided in the benchmark repository using the command *read*;
- synthesize the circuit using the command *resyn*;
- map to the cell library using the command *map -s*;
- perform the command *fraig_sweep* to identify functionally equivalent nodes and reduce the AIG representation;
- repeat the last two steps 10 times in order to incrementally have better results from ABC tool;
- export the result to structural Verilog format using the command *write*.

The Verilog file serves as input to our tool, together with the library in .lib format. It is necessary to highlight that ABC tool reports a structural netlist with a behavioral block *always* sensible to the rising edge of the clock signal, and therefore our structural Verilog parser replaces each signal assignment on it by an instance of a D type flip-flop.

Table 5.1: Description of ITC99 circuits

Name	**ORIGINAL FUNCTIONALITY**
B01	FSM that compares serial flows
B02	FSM that recognizes BCD numbers
B03	Resource arbiter
B04	Compute min and max
B05	Elaborate the contents of a memory
B06	Interrupt handler
B07	Count points on a straight line
B08	Find inclusions in sequences of numbers
B09	Serial to serial converter
B10	Voting system
B11	Scramble string with variable cipher
B12	1-player game (guess a sequence)

Source: Benchmark git repository[4].

5.2 Preliminary analysis

The set of experiments presented in this section aim to evaluate the general characteristics of the circuits and tools as they are submitted to the general design flow. We analyze the usual number of unitary fanout gates in a circuit, and how this number behaves as the size of the circuit increases.

5.2.1 Regarding cells of unitary fanout in a standard cell circuit

We want to answer some questions about the characteristics of circuits generated using traditional standard cell design flow, regarding the cells of unitary fanout. More specifically, we want to answer: how many cells of unitary fanout are in a circuit? How are they placed in a circuit? Does it depends on the cell placement tools?

How many cells of unitary fanout?

We analyze the profile of all circuits available in IWLS 2005 Benchmark (ALBRECHT, 2005) to count the number of cells with unitary fanout. We show in Figure 5.2 the characteristics of each circuit organized by the number of cells.

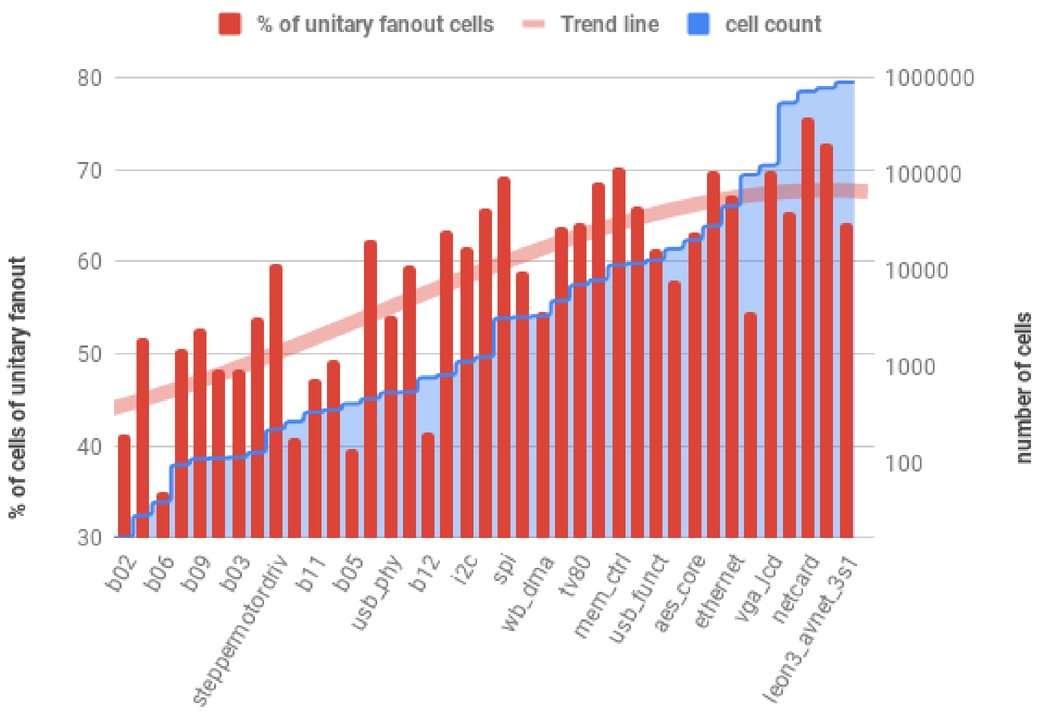
The average number of cells with unitary fanout in the circuits is over 58%. As we can see, the trend line shown in red indicates that this number tends to increase when bigger circuits are considered, showing that our approach has the potential to affect most cells of the circuit.

How cells of unitary fanout are placed?

In this experiment, we compare the relative distance of two kinds of interconnections: the distance between pairs of cells both with unitary fanout, and the distance between pairs of cells whose fanout of one of them is bigger than one. We use the placement results of three different types of placement tools as defined in (MARKOV; HU; KIM, 2015). We take the (x,y) position of each cell as reported in .pl file from bookshelf format generated by the tools and calculate their euclidean distance. We use Dragon (a simulated annealing based placement tool), Capo (a min-cut placement tool), and a NTUPlace3 (an analytic placement tool based on non-linear optimization).

The plots of the distribution of wirelengths of nets linking two cells of unitary

Figure 5.2: Unitary fanout cells count in circuits from ITC99, Opencores and Gaisler benchmarks



Source: own authorship

Table 5.2: Average distance between cells from B01 to B12 circuit according to placement done with Dragon, Capo and NTUPlace, divided into average distances between two cells of unitary fanout (D1) and average distances between two other cells (DN)

Ckt	**Dragon 3.01**			**Capo 10.5**			**NTUplace3-LE**		
	D1 (μm)	**DN (μm)**	**%**	**D1 (μm)**	**DN (μm))**	**%**	**D1 (μm)**	**DN (μm)**	**%**
b01	2,18	3,41	56,5	1,48	2,99	101,8	2,60	3,73	43,3
b02	1,75	2,69	53,7	2,27	2,96	30,2	3,27	3,85	17,8
b03	2,06	4,65	125,9	1,79	3,76	110,0	2,39	4,55	90,7
b04	2,74	5,71	108,6	2,52	5,62	122,8	2,31	6,76	191,8
b05	2,06	4,24	105,6	2,10	4,44	111,7	2,18	4,68	114,5
b06	2,66	3,51	32,2	1,87	2,50	33,4	3,09	3,59	16,4
b07	2,78	5,62	102,2	2,30	4,80	109,1	2,01	5,73	184,7
b08	2,36	4,13	74,8	2,28	3,63	59,3	2,22	4,52	103,7
b09	2,03	4,77	134,5	1,93	4,13	113,4	2,39	4,72	97,1
b10	2,60	4,24	63,5	2,33	4,31	84,6	2,71	4,68	72,3
b11	2,84	5,60	97,4	2,60	5,28	103,5	2,42	6,13	153,4
b12	2,94	6,08	107,0	2,69	5,90	119,2	3,13	6,36	103,3
Average	2,42	4,55	88,5	2,18	4,19	91,6	2,56	4,94	99,1

fanout (D1) and the wirelengths of the other nets (DN) according to placement done with three different types of placer tools are presented in Figure 5.3, Figure 5.4 and Figure 5.5, respectively. We can see that the wires linking cells of unitary fanout (qty_1) are shorter than the other cells in the circuit (qty_n), meaning that these cells are likely to be neat each other no matter which of the three kinds of the selected placement tool is used.

In Table 5.2.1, we show that the average distance between two cells of unitary fanout is almost half the average distance between two other cells. It means that our proposed approach tends to merge cells that would be placed together, thus smoothing the influence of our method over the total wirelength.

5.2.2 About the choice of initial cell library

Here we are interested in knowing how the choice of the first cell library influences the netlists regarding the quality parameters we adopted as goals in this thesis.

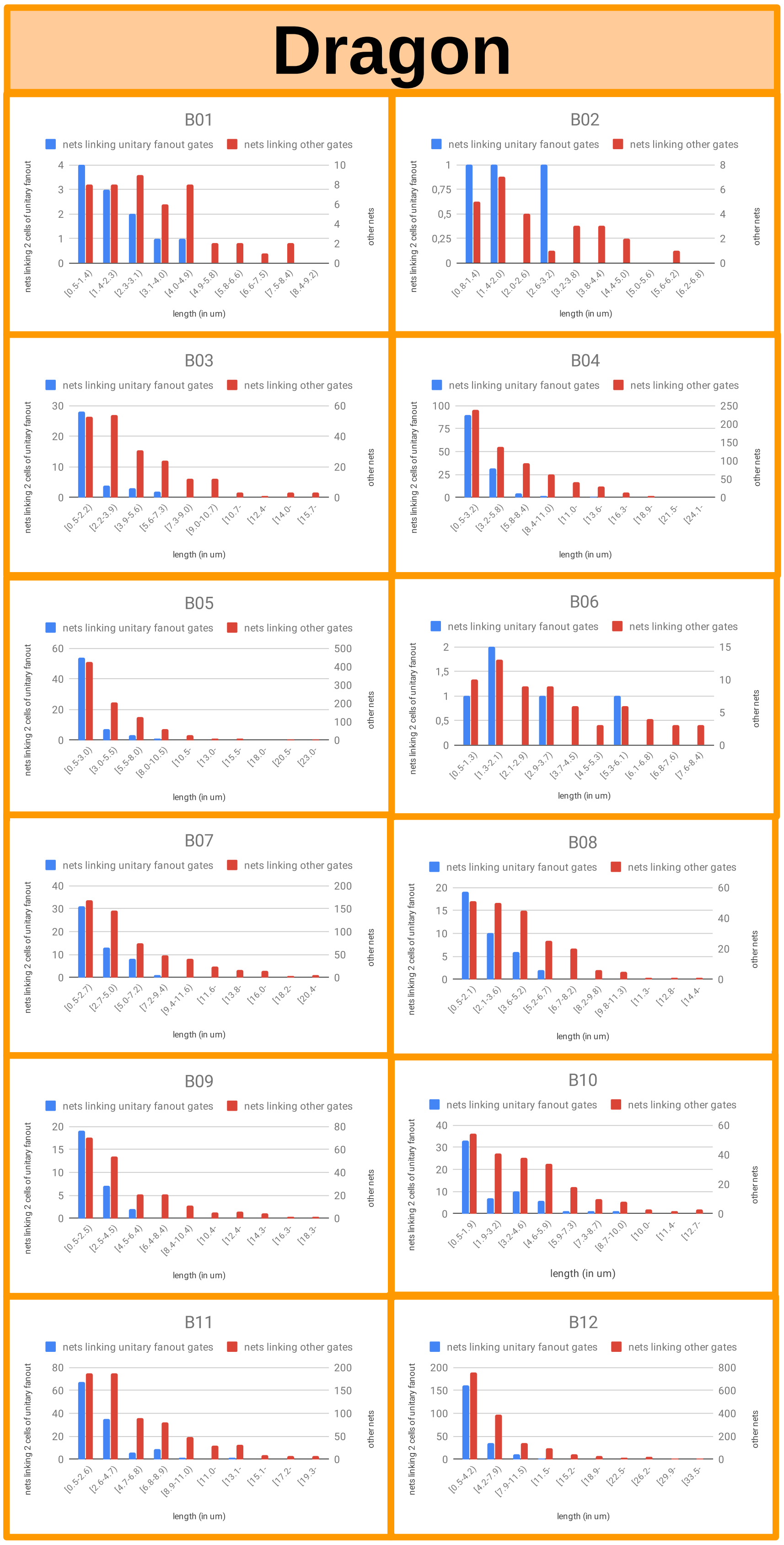
How it influences the number of instances, wires, and transistors?

We synthesized the circuits from the ITC99 benchmark using the logic synthesis tool ABC from Berkeley, targeting the cell libraries specified at the beginning of this chapter. We aim to evaluate the characteristics of the generated netlists regarding instances, wires, and transistors.

In Figure 5.6, it is shown a plot of the average number of transistors, wires, and instances of the ITC99 benchmark circuits as a function of the mentioned cell libraries ordered by the number of logic functions available. As the genlib format only has the logic function of the gates, we considered the number of transistors as twice the number of inputs in the function.

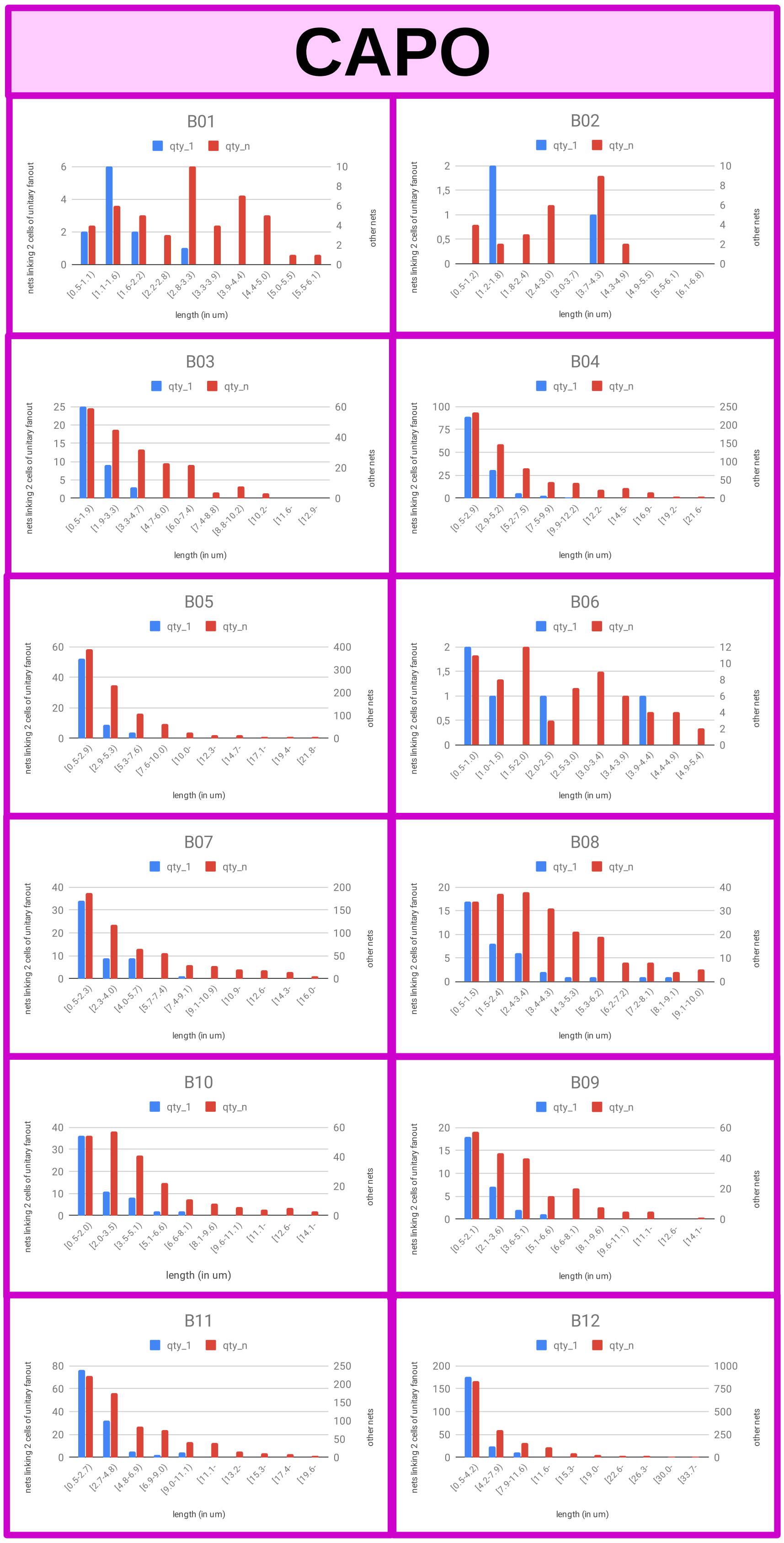
We can see that the ABC tool tries to minimize the number of instances in the final netlist, which indeed decreases considerably as the number of available logic functions increases. The number of wire segments and the number of transistors also decreases, but at a lower rate.

Figure 5.3: Plot of the distribution of net length estimation divided into nets linking two unitary fanout cells and other nets from circuit B01 to B12, according to Dragon placer.



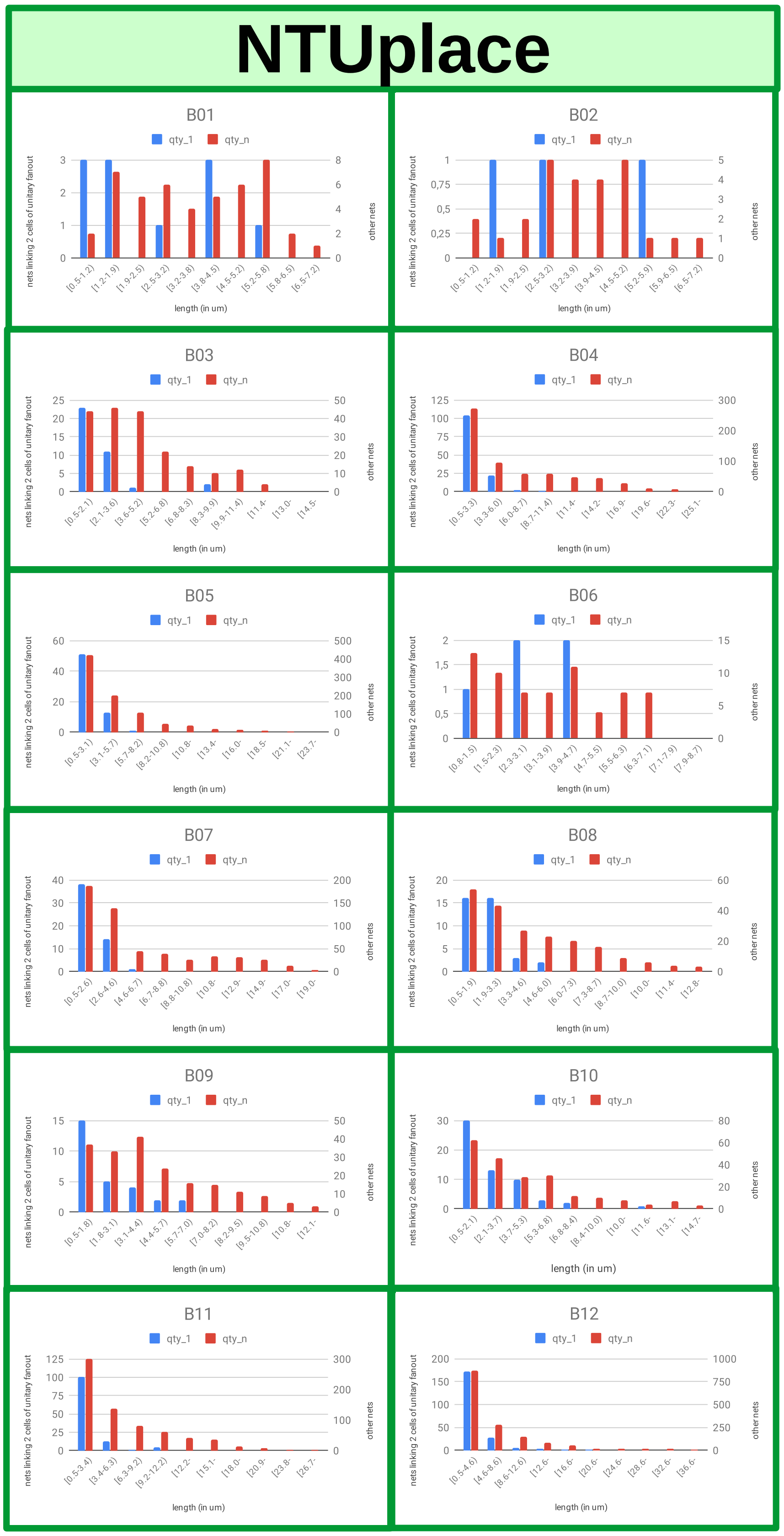
Source: own authorship

Figure 5.4: Plot of the distribution of net length estimation divided into nets linking two unitary fanout cells and other nets from circuit B01 to B12, according to Capo placer.



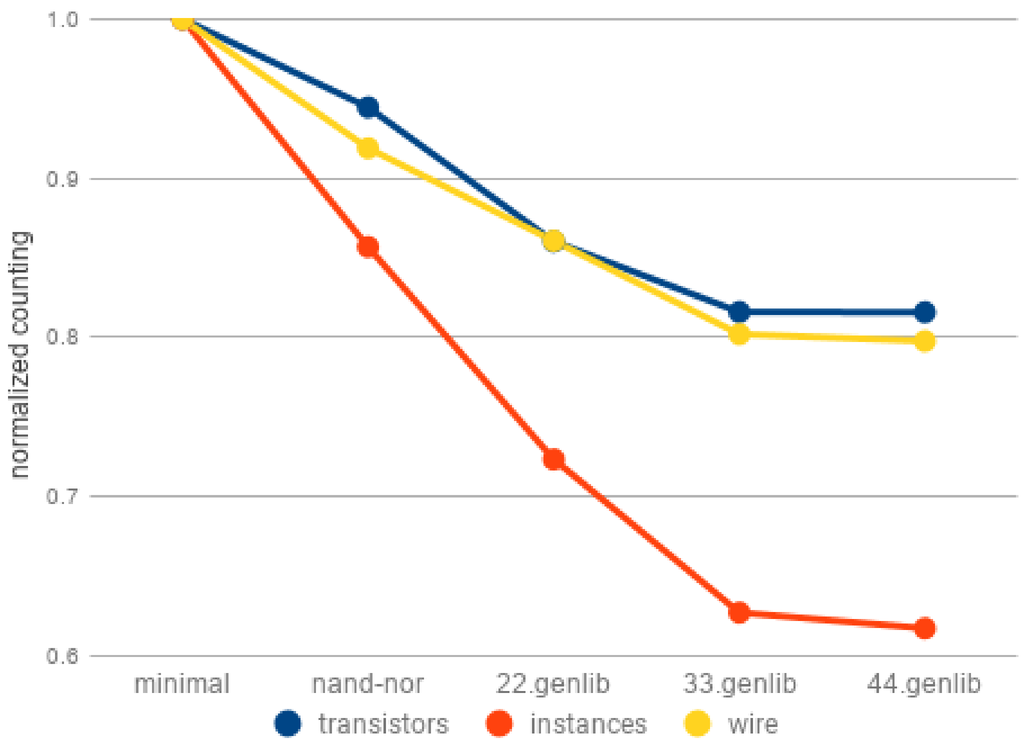
Source: own authorship

Figure 5.5: Plot of the distribution of net length estimation divided into nets linking two unitary fanout cells and other nets from circuit B01 to B12, according to NTU placer.



Source: own authorship

Figure 5.6: Plot of the number of instances, wires segments, and transistors found in average on circuits from ITC99 benchmark synthesized to cell libraries of increasing number of cells, normalized by the counting in netlists generated targeting minimal.genlib.



Source: own authorship

5.3 Evaluating the proposed algorithms

This section is about the analysis of the proposed technique, covering its potential of improvement of a circuit netlist in terms of transistors, wires, and instances count, and at which circumstances it performs better. We apply the algorithms presented in Chapter 3 in different scenarios using the ITC99 benchmark of small circuits in order to construct a deeper understanding of how our technique works.

5.3.1 UNBOUNDED algorithm

Despite being naive from a practical point-of-view, as mentioned in Chapter 3 of this thesis, the UNBOUNDED algorithm (Algorithm 3) can be valuable to establish an upper bound for the optimization that our technique can provide. Here we evaluate how our technique behaves for different input netlists generated to target different cell libraries using ABC tool.

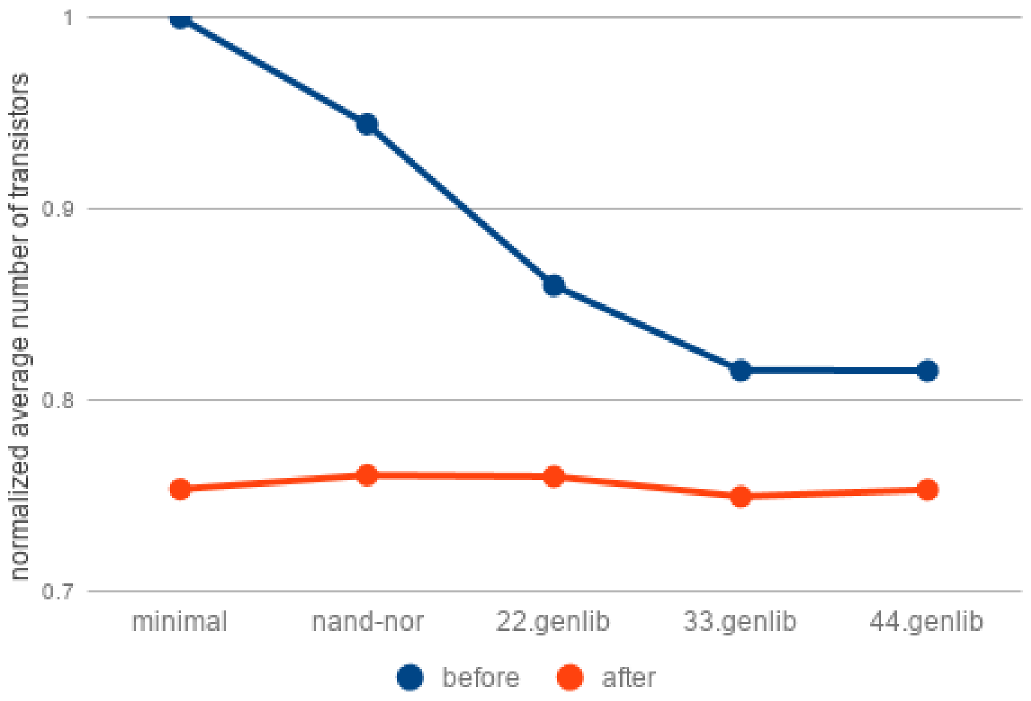
How many optimizations the cell merging technique can provide?

In our post-processing method, the new cell will replace a group of connected cells only if it needs fewer transistors than the original group of cells so that the whole circuit remains with the same number of transistors in the worst scenario after applying our technique. The first result set is shown in Figure 5.7, where the data is normalized by the average number of transistors in the synthesis of the ITC99 benchmark circuits to the *smallest* library.

As can be seen, the results in transistor count are almost the same independently of the initial library. So, we can say that our technique can reduce the number of transistors regardless of the size of the initial library used as a target for the input netlist, which can save computational effort spent when considering bigger libraries.

The previous analysis indicates that our proposed technique may be a better way to provide more logic functions in a library free effort, then when performing logic synthesis with ABC targeting a big virtual library. To confirm this claim, we counted the number of instances obtained before and after the application of the optimization procedure we propose. We show a plot of these results in Figure 5.8. As we can notice, the average number of instances after the application of the technique remains almost constant even

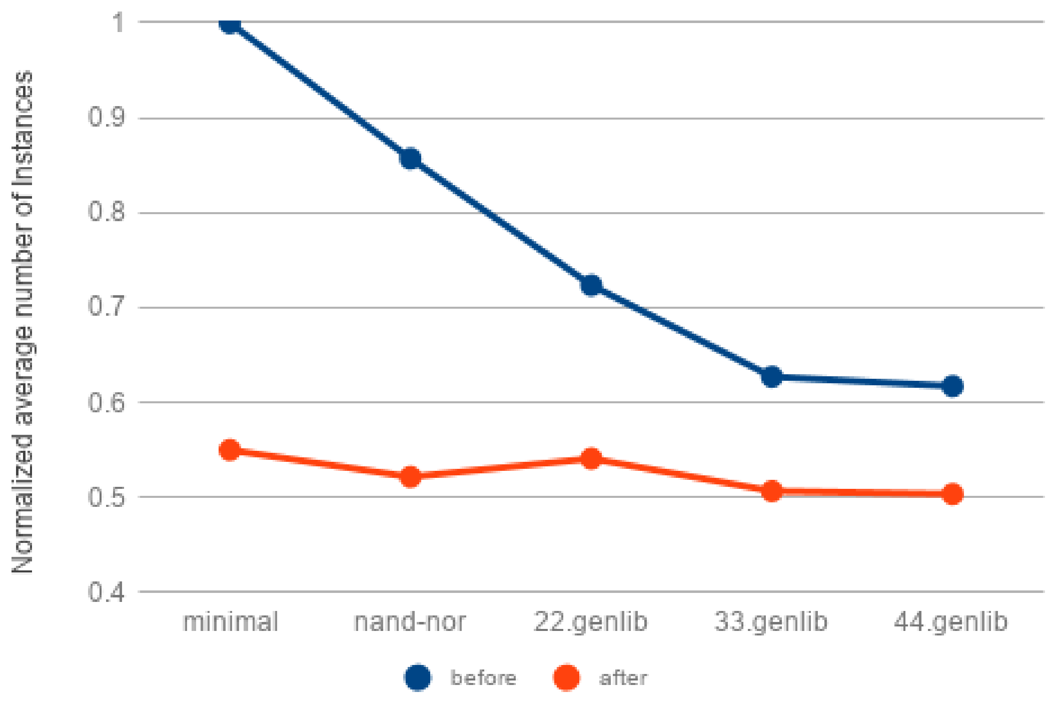
Figure 5.7: Plot of the average number of transistors before and after the application of the proposed merging technique over ITC99 benchmark circuits synthesized to different libraries, normalized by minimal.genlib average before values.



Source: own authorship

though the size of the initial targeting library differs. Therefore, we can say that our proposed optimization technique is library free so that it could take as input a technology independent netlist.

Figure 5.8: Plot of the average number of instances before and after the application of the proposed merging technique over ITC99 benchmark circuits synthesized to different libraries, normalized by minimal.genlib average before values.



Source: own authorship

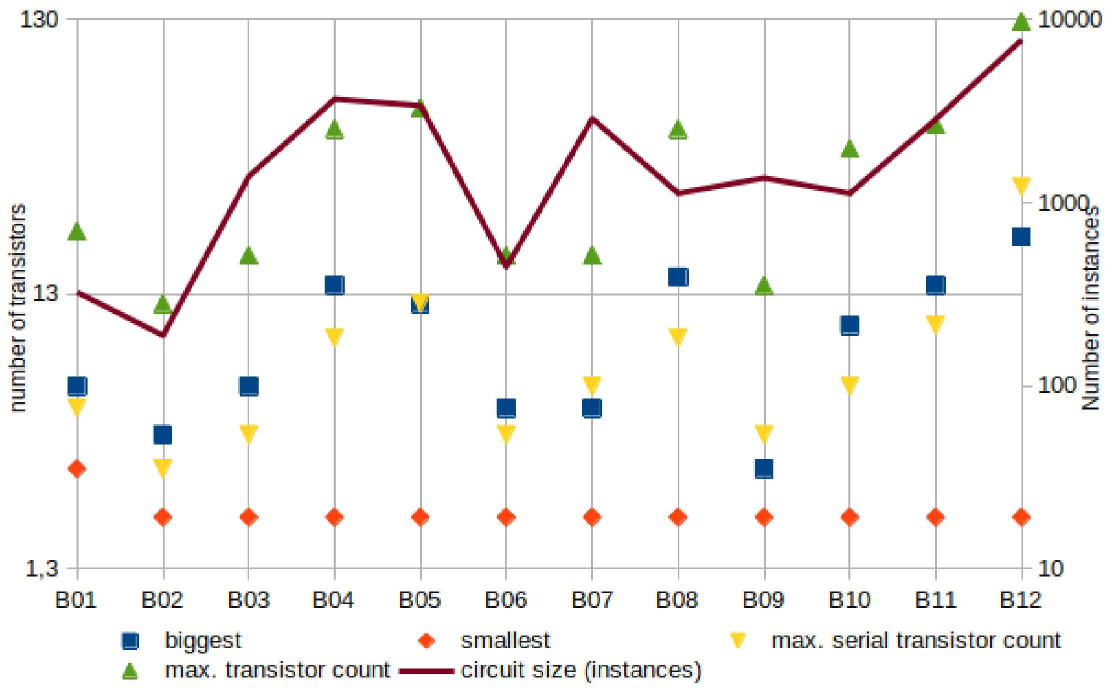
It is important to remember, however, that those results were obtained by applying the UNBOUNDED algorithm, a greedy version of our merging technique that has many practical problems, as it was already discussed. Nevertheless, it gives a good indication that our technique may generate results as good as a leading academic logic synthesis tool in terms of the number of instances, but with a smaller number of transistors.

What are the limitations of adopting the UNBOUNDED algorithm?

From the experiments presented so far, we can say that our approach has the potential to generate a circuit with fewer transistors than a synthesis targeting an extensive library, but keeping an equivalent number of instances. However, the results of the UNBOUNDED merging algorithm cannot be considered in practice. The chart in Figure 5.9

presents a plot with the size of the biggest and the smallest group of cells replaced, the maximum number of serial transistors and the maximum transistor count in a new cell for input circuits initially synthesized to cadence.genlib using ABC. We can see that it leads to new gates with unfeasible parameters, like 32 serial transistors for the B12 circuit.

Figure 5.9: Plot of biggest and smallest group of cells, maximum transistor count and transistors in series, and circuit size of each circuit after using UNBOUNDED algorithm



Source: own authorship

5.3.2 BOUNDED algorithm

The UNBOUNDED algorithm has greedy behavior that merges all cells in every maximal group to generate a single complex cell to replace them. By neglecting the recommended number of stacked transistors and avoiding to insert new inverters to assure logic equivalence, it assumes the best scenario in every gate replacement promoted by our technique. Therefore, we use the results of this algorithm to present the optimization limit we can provide.

The goal of these experiments is to evaluate the proposed merging methodology using a feasible way to generate new cells, aiming to avoid the undesired cells resulting from the unbounded algorithm. In this way, the BOUNDED procedure is applied, which was presented in Algorithm 4.

This BOUNDED algorithm adds limitations in the total number of transistors and in the number of serial transistors of the resulting new cell as a condition to keep merging cells of the original netlist. Therefore, two constraints to the previous merging method are implemented as conditional statements: the limiting number of serial transistors (S_LIMIT parameter) – in both pull up and pull down –, and the total number of transistors (T_LIMIT parameter).

Such modifications in the initial algorithm still lead to a greedy solution, as it starts from a seed and explores the neighborhood looking for connected cells of unitary fanout, making a local decision of merging the neighbor cell, and it does not backtrack. It still does not delivers optimal cell grouping in terms of transistor count, but it is fast and sufficient to bring realistic new cells as a result of the proposed merging process.

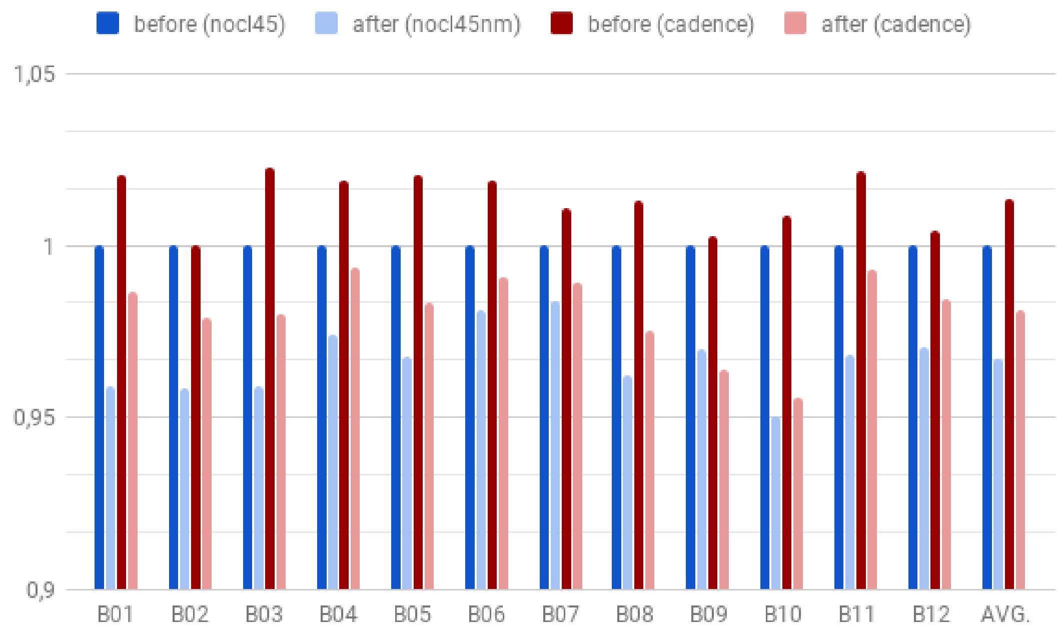
The general procedure must guarantee logical equivalence between the transformations. In this sense, when an input of the new cell is inverted, the procedure must search the circuit looking for that inverted signal. If it is not present, additional inverters have to be inserted, and its transistors are taken into account when deciding whether to replace the original group of cells by the new one.

In the results presented herein, the merging technique was applied with the new cells limited to 4 serial transistors and 12 transistors in total. The ITC99 circuits were synthesized with the ABC logic synthesis tool. The libraries nocl45nm.genlib and cadence.genlib were chosen as a target due to having a more practical set of functions among the libraries adopted in this thesis. Both library files were updated with the parameters the genlib format supports extracted from the NOCL liberty file.

Figure 5.10 shows a plot with the compared values of transistor count in two netlists. Notice that we always reduce the number of transistors, 3.3% for nocl45nm library, and 1.9% for cadence library. As expected, we observe a smaller reduction in the number of transistors when compared to the BOUNDED algorithm due to the imposed restrictions. Also, in the chart shown in Figure 5.11, we can see an average reduction of 15% for the netlist synthesized to nocl45nm library and 9% for the netlist generated to cadence library. We reduce the number of instances in both netlists, even considering the insertion of inverters to assure logic equivalence.

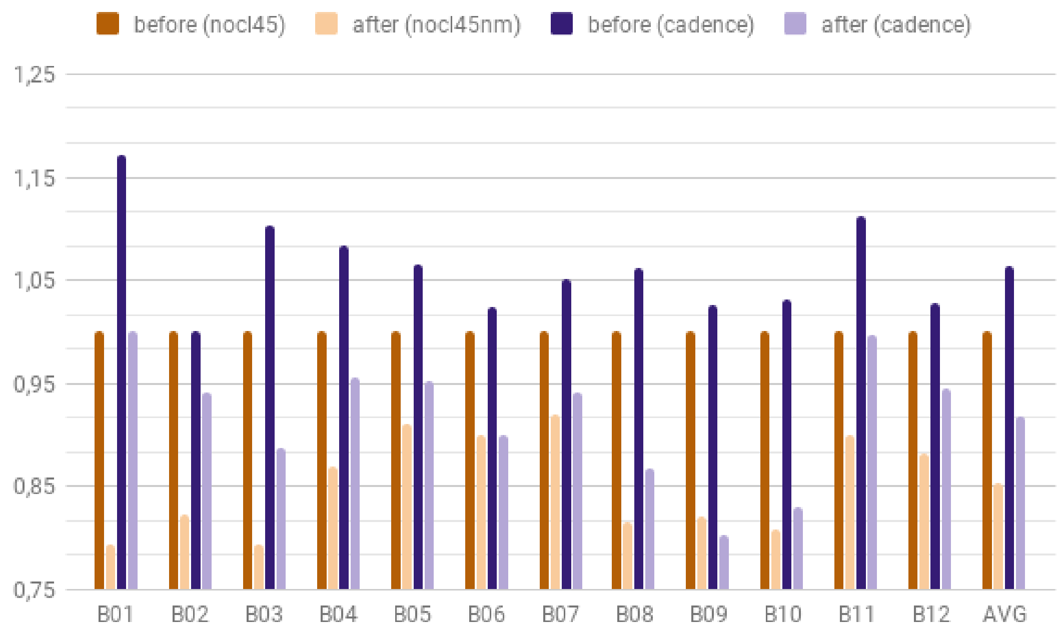
In Figure 5.12 and in Figure 5.13, we show a comparison between the count of wire segment and contacts before and after the application of BOUNDED algorithm for

Figure 5.10: Plot of the number of transistors before and after the application of the BOUNDED algorithm over ITC99 benchmark circuits synthesized to nocl45nm.genlib and cadence.genlib libraries, detailed by circuit, normalized by nocl45nm.genlib before values



Source: own authorship

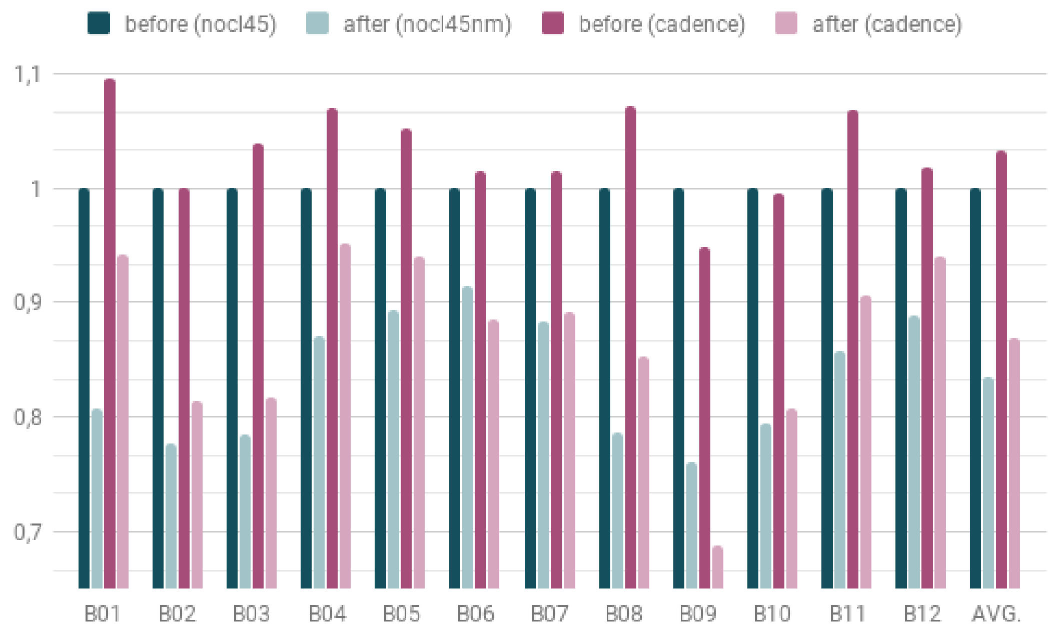
Figure 5.11: Plot of the number of instances before and after the application of the BOUNDED algorithm over ITC99 benchmark circuits synthesized to nocl45nm.genlib and cadence.genlib libraries, detailed by circuit, normalized by nocl45nm.genlib before values



Source: own authorship

gate merging, respectively. We can observe an average reduction in the number of wires and in the number of contacts of about 17% in both netlists, with a respective peak of 31% and 23% in the case of circuit b09. These decreases can impact routability and area, since fewer contacts mean fewer vias, whose area represents a portion of the total circuit area.

Figure 5.12: Plot of the number of wire segments before and after the application of the BOUNDED algorithm over ITC99 benchmark circuits synthesized to nocl45nm.genlib and cadence.genlib libraries, detailed by circuit, normalized by nocl45nm.genlib before values.



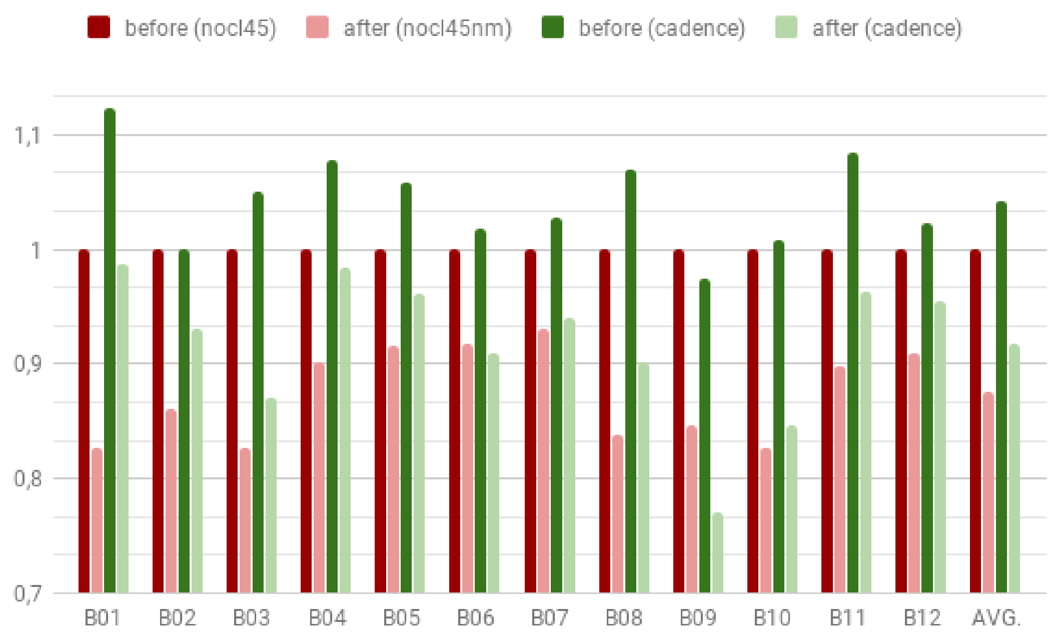
Source: own authorship

Another relevant parameter to observe is the number of inverted inputs the new cells have and the number of inverters that were inserted due to lack of the demanded signal in the circuit. We show in Figure 5.14 the proportion between these two data. We apply the heuristics presented in the previous chapter to find those signals in the circuit, so that we can avoid inserting a new inverter in 90% of the cases for nocl45nm netlists and 88% for cadence netlists, in average.

5.4 Comparing the BOUNDED and OPTIMIZED Algorithm

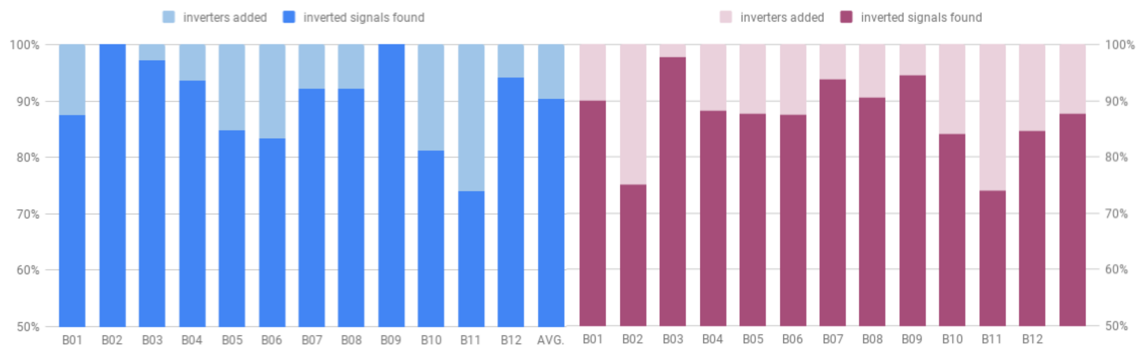
As discussed in Chapter 3, the BOUNDED algorithm is not optimal in terms of the number of transistors. Then it presents the OPTIMIZED algorithm that combines the

Figure 5.13: Plot of the number of contacts before and after the application of the BOUNDED algorithm over ITC99 benchmark circuits synthesized to nocl45nm.genlib and cadence.genlib libraries, detailed by circuit, normalized by nocl45nm.genlib before values.



Source: own authorship

Figure 5.14: Plots of the ratio between inverted signals found in the circuit and inverter insertion demanded by the SCCGs in the ITC99 benchmark circuits initially synthesized to NOCL45nm.genlib (left) and to cadence.genlib (right)



Source: own authorship

UNBOUNDED algorithm to find the whole group of connected cells, and then it explores all possible grouping to find the one that demands the smallest number of transistors. As the number of possible grouping increases exponentially, we set a limit for the size of the group of connected cells (MAX_NODES) in which the algorithm will explore all possibilities. For groups of connected cells bigger than this limit, the OPTIMIZED algorithm uses the BOUNDED algorithm in order to save execution time.

In this experiment, we apply algorithms BOUNDED and OPTIMIZED over the netlist B01 to B12 initially synthesized using ABC targeting the nocl45nm.genlib library. The goal is to compare the algorithms, and show that the OPTIMIZED algorithm achieves results more near to the limit we achieved by applying the UNBOUNDED algorithm over the same set of the netlist.

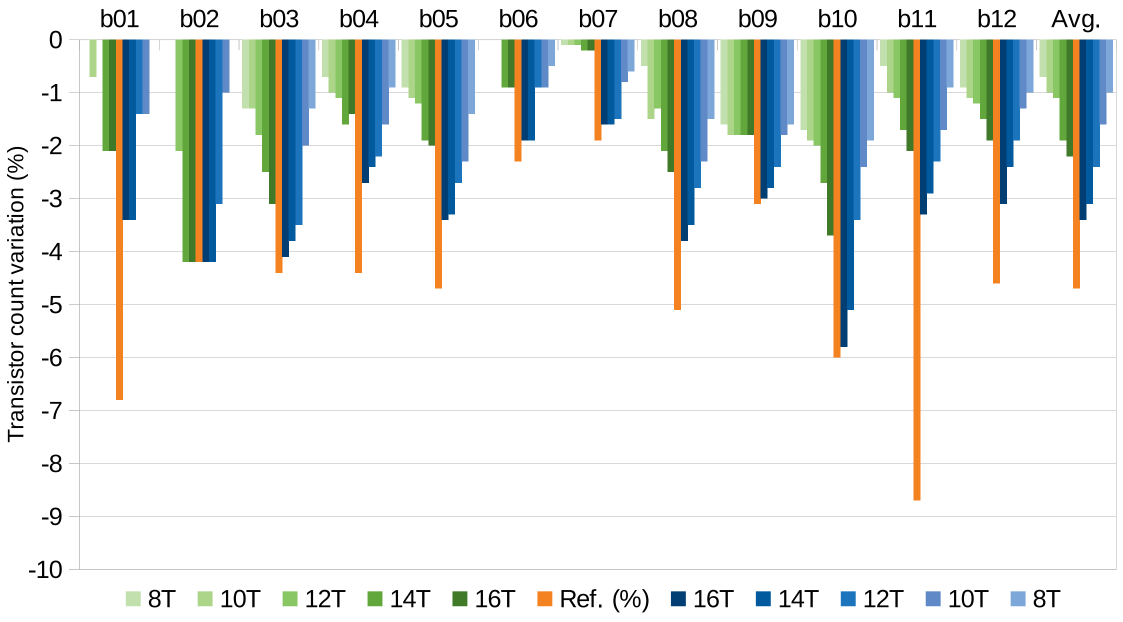
We compare the algorithms taking as reference values the results from UNBOUNDED algorithm. We set the OPTIMAL algorithm to take ten as the maximum number of cells in the identified group of connected cells to perform the exhaustive searching for all possible groups. The number was arbitrarily chosen to ensure an efficient execution time in the computer hardware available to run the experiments. We vary for both algorithms from 8, 10, 12, 14, and 16 as the limiting number of transistors the new cells could have, and we set 4 as the maximum number of serial transistors allowed in the pull-up and pull-down of the new cells. We discard any possible new gate whose parameters are beyond these values.

5.4.1 Transistor count

Table 5.3 shows details about transistor count improvements achieved with both algorithms for these parameters. We can see that the OPTIMIZED algorithm always leads to smaller transistor count than the BOUNDED algorithm, in some cases achieving values very close or equal to the reference values. For all scenarios, the resulting number of transistors is equal or smaller than the original count since we only replace cells when the new one demands a smaller number of transistors, including the inverters it may demand.

As we can observe better in Figure 5.15, both algorithms achieve a more significant decrease in the total number of transistors when bigger cells are allowed. Also, we can see that the reduction in transistor count obtained with OPTIMIZED algorithm is necessarily incremental as we increase the maximum number of transistors allowed for the new cells. The same observation is valid for BOUNDED algorithm in most cases, but its non-optimal grouping becomes evident in circuits b01, b04, and b08 where, for some values, allowing bigger cells lead to more soft reductions in transistor count.

Figure 5.15: Variation in transistor count in each circuit with bounded method (left) and optimal method (right), compared to the reference value (middle).



Source: own authorship

Table 5.3: Transistor count Variation (%), comparing BOUNDED and OPTIMIZED algorithms.

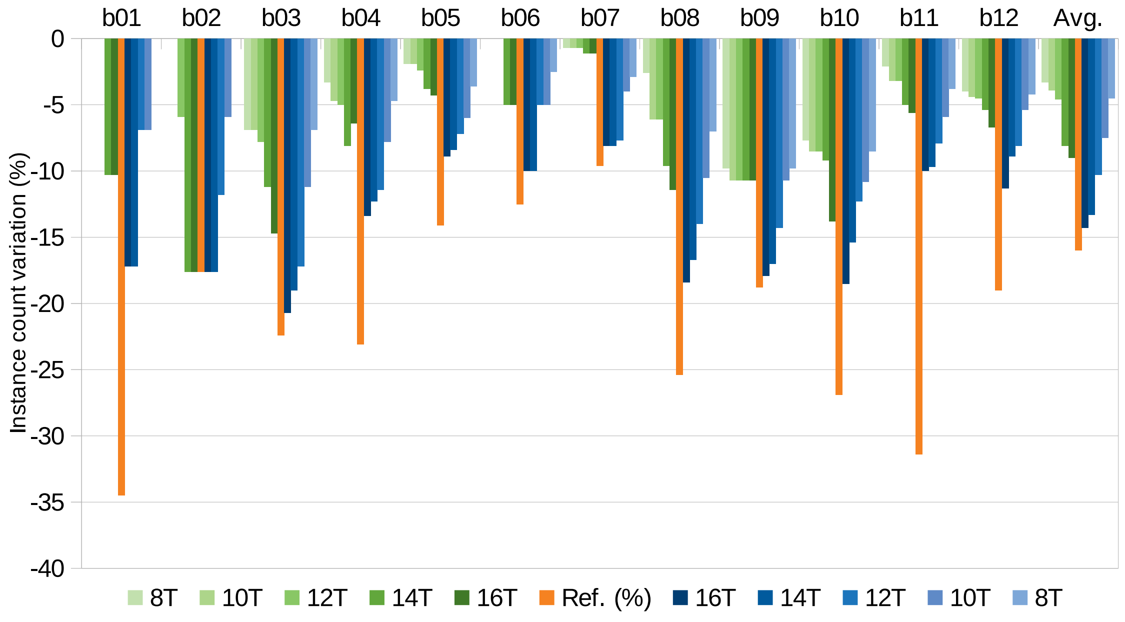
	Original	BOUNDED Algorithm					OPTIMIZED Algorithm				
CKT	Count	8T	10T	12T	14T	16T	8T	10T	12T	14T	16T
b01	292	0	-0,7	0	-2,1	-2,1	0,0	-1,4	-1,4	-3,4	-3,4
b02	192	0	0	-2,1	-4,2	-4,2	0,0	-1,0	-3,1	-4,2	-4,2
b03	1.414	-1,3	-1,3	-1,8	-2,5	-3,1	-1,3	-2,0	-3,5	-3,8	-4,1
b04	3.916	-0,7	-1	-1,1	-1,6	-1,4	-0,9	-1,6	-2,2	-2,4	-2,7
b05	3.078	-0,9	-1,1	-1,2	-1,9	-2	-1,4	-2,3	-2,7	-3,3	-3,4
b06	428	0	0	0	-0,9	-0,9	-0,5	-0,9	-0,9	-1,9	-1,9
b07	2.766	-0,1	-0,1	-0,1	-0,2	-0,2	-0,6	-0,8	-1,5	-1,6	-1,6
b08	1.212	-0,5	-1,5	-1,3	-2,1	-2,5	-1,5	-2,3	-2,8	-3,5	-3,8
b09	1.336	-1,6	-1,8	-1,8	-1,8	-1,8	-1,6	-1,8	-2,4	-2,8	-3,0
b10	1.174	-1,7	-1,9	-2	-2,7	-3,7	-1,9	-2,4	-3,4	-5,1	-5,8
b11	2.780	-0,5	-1	-1,1	-1,7	-2,1	-0,9	-1,7	-2,3	-2,9	-3,3
b12	7.302	-0,9	-1,1	-1,2	-1,5	-1,9	-1,0	-1,3	-1,9	-2,4	-3,1
AVG.	2.158	-0,7	-1,0	-1,1	-1,9	-2,2	-1,0	-1,6	-2,4	-3,1	-3,4

Source: own authorship

5.4.2 Instance count

We are also interested to see how the algorithms behave regarding the number of instances. Table 5.4 shows the original number of instances in the input netlists generated by ABC targeting the nocl45nm library. It shows an average reduction of 14.3% from the transistor count of the initial netlist with the OPTIMIZED algorithm with new cells up 16 transistors, which is quite close to the reference value of 16%. The data is also represented graphically in Figure 5.16. Although we achieve the reference reduction with both algorithms for B02 circuit, the results achieved with OPTIMIZED algorithm are closer the reference value for most circuits.

Figure 5.16: Variation in instance count in each circuit with bounded method (left) and optimal method (right), compared to the reference value (middle).



Source: own authorship

Another interesting data we want to evaluate is how our technique behaves regarding transistor count when compared to netlist generated using ABC synthesized to its largest virtual cell library, the 44.genlib. The Table 5.5 presents this counting, applying our OPTIMIZED algorithm set to 16 transistors as a maximum in the new cell to two initial netlists generated with ABC, one targeting 44.genlib and the other targeting nocl45nm.genlib. As it can be noticed, our technique minimizes transistor count in both cases and achieves the best results when the input netlist is generated targeting the

Table 5.4: Instance count variation (%), comparing BOUNDED and OPTIMIZED algorithms.

	Original	BOUNDED Algorithm					OPTIMIZED Algorithm				
CKT	Count	8T	10T	12T	14T	16T	8T	10T	12T	14T	16T
b01	29	0,0	0,0	0,0	-10,3	-10,3	0,0	-6,9	-6,9	-17,2	-17,2
b02	17	0,0	0,0	-5,9	-17,6	-17,6	0,0	-5,9	-11,8	-17,6	-17,6
b03	116	-6,9	-6,9	-7,8	-11,2	-14,7	-6,9	-11,2	-17,2	-19,0	-20,7
b04	359	-3,3	-4,7	-5,0	-8,1	-6,4	-4,7	-7,8	-11,4	-12,3	-13,4
b05	417	-1,9	-1,9	-2,4	-3,8	-4,3	-3,6	-6,0	-7,2	-8,4	-8,9
b06	40	0,0	0,0	0,0	-5,0	-5,0	-2,5	-5,0	-5,0	-10,0	-10,0
b07	272	-0,7	-0,7	-0,7	-1,1	-1,1	-2,9	-4,0	-7,7	-8,1	-8,1
b08	114	-2,6	-6,1	-6,1	-9,6	-11,4	-7,0	-10,5	-14,0	-16,7	-18,4
b09	112	-9,8	-10,7	-10,7	-10,7	-10,7	-9,8	-10,7	-14,3	-17,0	-17,9
b10	130	-7,7	-8,5	-8,5	-9,2	-13,8	-8,5	-10,8	-12,3	-15,4	-18,5
b11	341	-2,1	-3,2	-3,2	-5,0	-5,6	-3,8	-5,9	-7,9	-9,7	-10,0
b12	778	-4,0	-4,4	-4,5	-5,4	-6,7	-4,2	-5,4	-8,1	-8,9	-11,3
AVG.	227	-3,3	-3,9	-4,6	-8,1	-9,0	-4,5	-7,5	-10,3	-13,3	-14,3

Source: own authorship

smallest netlist.

Table 5.5: Transistor count achieved with OPTMIZED algorithm limiting new gates to 16 transistors, compared to netlists generated by ABC

Ckt	44.genlib				nocl45nm.genlib			
	ABC	**OPT (16T)**	**%**	**runtime**	**ABC**	**OPT (16T)**	**%**	**runtime**
B01	158	152	-3,80	00:00:05	144	134	-6,94	00:00:06
B02	78	72	-7,69	00:00:02	76	72	-5,26	00:00:01
B04	2048	1944	-5,08	00:02:50	1868	1.766	-5,46	00:01:21
B05	2066	1972	-4,55	00:00:35	2004	1.924	-3,99	00:03:36
B06	180	170	-5,56	00:00:03	176	168	-4,55	00:00:02
B07	1488	1400	-5,91	00:00:25	1326	1.282	-3,32	00:00:17
B08	584	542	-7,19	00:02:45	576	530	-7,99	00:00:23
B09	548	484	-11,68	00:00:18	504	466	-7,54	00:00:11
B10	668	634	-5,09	00:00:22	605	600	-0,83	00:00:48
B11	1908	1812	-5,03	00:12:23	1748	1.672	-4,35	00:13:13
B12	3846	3628	-5,67	00:09:25	3762	3.596	-4,41	00:06:14
AVG	1233,8	1164,5	-6,1	00:02:55	1337,6	1279,8	-5,0	00:02:37

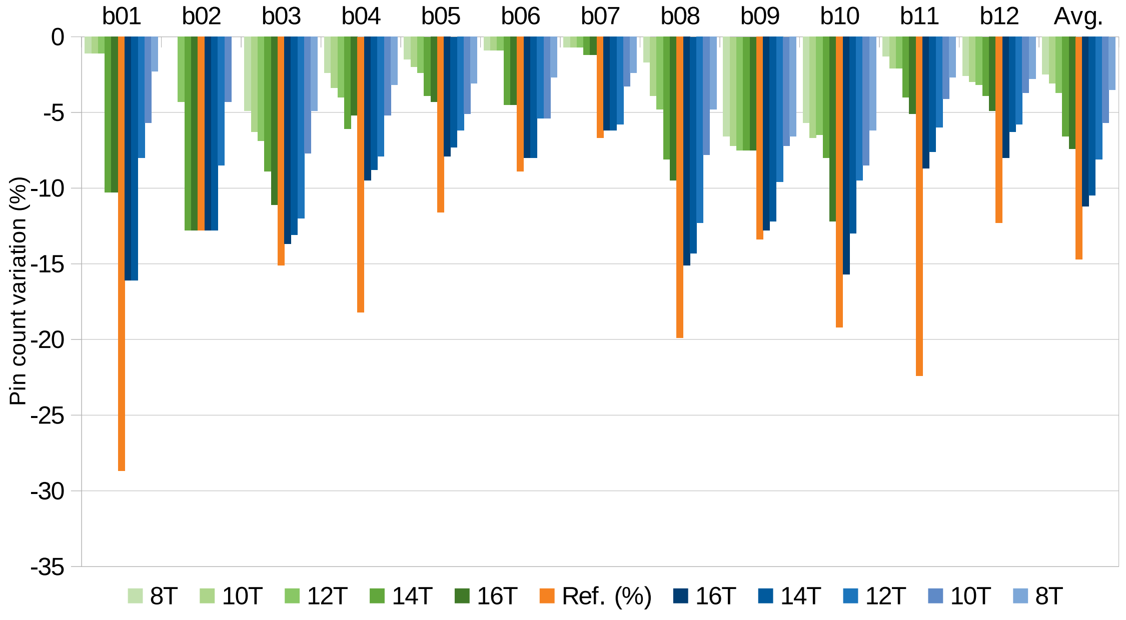
In the results of both libraries shown in Table 5.5, the average runtime was about 2 minutes in a notebook Intel i7 with 8GB of RAM, running Ubuntu 18.04 for x86_64 architecture. Although a short time, we can see it tends to increase when considering bigger circuits. We identify as necessary future work to reduce the runtime of the algorithm.

5.4.3 Net and Contact count

Although using transistor count as a performance parameter, the solution we propose can achieve desirable results, verifiable by comparing the netlists before and after its application. So, in the following charts and tables, we present a comparison of the net count, contact count, and wire segment count achieved with both algorithms. Reducing these parameters is an apparent secondary gain of reducing transistor count. As we can see, in all parameters, the best results are achieved using the OPTIMIZED algorithm, which is incrementally better when we increase the maximum number of transistors allowed in the new cells.

By looking at Figure 5.17 and Table 5.6, we see that we achieve an average reduction of 15% in the number of contacts in the circuit, with a maximum of 28% and a minimal of 6%. These values are 75% of the reduction limit established by the UNBOUNDED algorithm. Reducing the number of contacts also means fewer vias in the final layout, whose area in the final layout is not negligible.

Figure 5.17: Variation in contact count in each circuit with bounded method (left) and optimal method (right), compared to the reference value (middle).



Source: own authorship

Table 5.6: Contact count variation (%), comparing BOUNDED and OPTIMIZED algorithms.

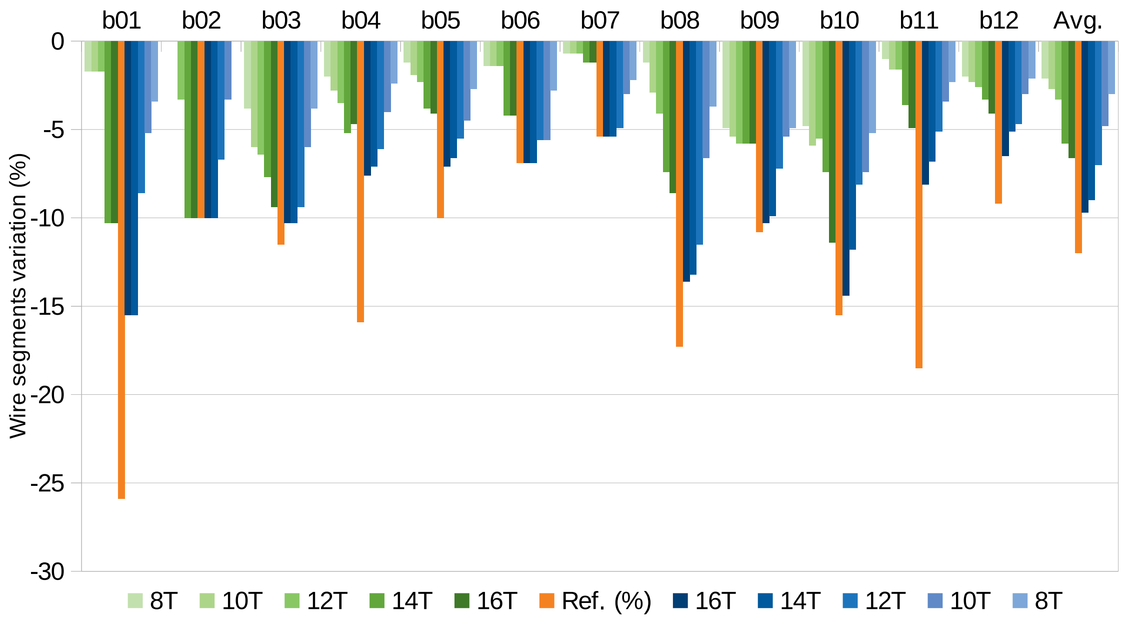
	Count	BOUNDED Algorithm					OPTIMIZED Algorithm				
CKT		8T	10T	12T	14T	16T	8T	10T	12T	14T	16T
b01	87	-1,1	-1,1	-1,1	-10,3	-10,3	-2,3	-5,7	-8,0	-16,1	-16,1
b02	47	0	0	-4,3	-12,8	-12,8	0,0	-4,3	-8,5	-12,8	-12,8
b03	350	-4,9	-6,3	-6,9	-8,9	-11,1	-4,9	-7,7	-12,0	-13,1	-13,7
b04	1.108	-2,4	-3,4	-4	-6,1	-5,2	-3,2	-5,2	-7,9	-8,8	-9,5
b05	1.307	-1,5	-2	-2,4	-3,9	-4,3	-3,1	-5,1	-6,2	-7,3	-7,9
b06	112	-0,9	-0,9	-0,9	-4,5	-4,5	-2,7	-5,4	-5,4	-8,0	-8,0
b07	866	-0,7	-0,7	-0,7	-1,2	-1,2	-2,4	-3,3	-5,8	-6,2	-6,2
b08	357	-1,7	-3,9	-4,8	-8,1	-9,5	-4,8	-7,8	-12,3	-14,3	-15,1
b09	335	-6,6	-7,2	-7,5	-7,5	-7,5	-6,6	-7,2	-9,6	-12,2	-12,8
b10	401	-5,7	-6,7	-6,5	-8	-12,2	-6,2	-8,5	-9,5	-13,0	-15,7
b11	1.139	-1,3	-2,1	-2,1	-4	-5,1	-2,7	-4,1	-6,0	-7,6	-8,7
b12	2.488	-2,6	-3	-3,2	-3,9	-4,9	-2,8	-3,7	-5,8	-6,3	-8,0
AVG.	716	-2,5	-3,1	-3,7	-6,6	-7,4	-3,5	-5,7	-8,1	-10,5	-11,2

Source: own authorship

By applying the proposed merging technique, we can also have an average reduction in the number of nets and wire segments, respectively, in 21% and 12%. Figure 5.19 and Figure 5.18 exhibit graphically the data about net and wire segment count, while details are shown in Table 5.8 and Table 5.7, respectively. The number of nets decreases more significantly because the right focus of our technique is in the nets with only one segment. Again, the OPTIMIZED algorithm achieves the results most approximated to the limits observed by the UNBOUNDED algorithm. It is acceptable to expect that the average wire length in the circuit will increase after applying our technique, as we reduce the number of short wire segments.

For all considered parameters, the best results are achieved with OPTIMIZED algorithm adjusted to limit in 16 the number of transistors in the new cells and set to perform the exhaustive search described in Algorithm 5 when the group of connected cells of unitary fanout is at most 10. As a reference for comparison, a D-type flip-flop of unitary dimension in Nangate Open Cell Library has 28 transistors. A cell with 16 transistors would not be the biggest in the resulting circuit. By using these parameters, we could achieve at least 67% (net count) and, at most 89% (instances) of the optimization estimated with the UNBOUNDED algorithm, on average. Regarding transistor count, we achieved with this algorithm and parameters 71% of the measured upper limit of improvement. However, the BOUNDED algorithm runs ten times faster than the OPTIMIZED procedure. Therefore, the challenge with the OPTIMIZED algorithm is to improve its runtime, whose bottleneck

Figure 5.18: Variation in wire segments count in each circuit with bounded method (left) and optimal method (right), compared to the reference value (middle).



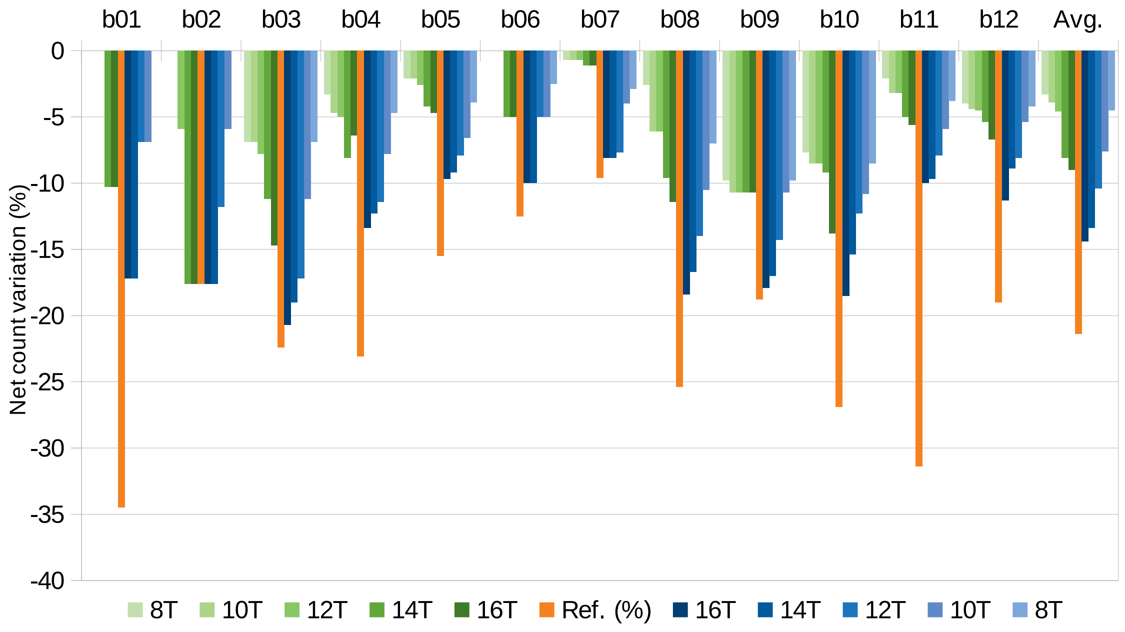
Source: own authorship

Table 5.7: Wire segments count variation (%), comparing BOUNDED and OPTIMIZED algorithms.

CKT	Count	BOUNDED Algorithm					OPTIMIZED Algorithm				
		8T	10T	12T	14T	16T	8T	10T	12T	14T	16T
b01	57	-1,7	-1,7	-1,7	-10,3	-10,3	-3,4	-5,2	-8,6	-15,5	-15,5
b02	30	0,0	0,0	-3,3	-10,0	-10,0	0,0	-3,3	-6,7	-10,0	-10,0
b03	225	-3,8	-6,0	-6,4	-7,7	-9,4	-3,8	-6,0	-9,4	-10,3	-10,3
b04	734	-2,0	-2,8	-3,5	-5,2	-4,7	-2,4	-4,0	-6,1	-7,1	-7,6
b05	915	-1,2	-1,9	-2,3	-3,8	-4,1	-2,7	-4,5	-5,5	-6,6	-7,1
b06	71	-1,4	-1,4	-1,4	-4,2	-4,2	-2,8	-5,6	-5,6	-6,9	-6,9
b07	590	-0,7	-0,7	-0,7	-1,2	-1,2	-2,2	-3,0	-4,9	-5,4	-5,4
b08	240	-1,2	-2,9	-4,1	-7,4	-8,6	-3,7	-6,6	-11,5	-13,2	-13,6
b09	212	-4,9	-5,4	-5,8	-5,8	-5,8	-4,9	-5,4	-7,2	-9,9	-10,3
b10	258	-4,8	-5,9	-5,5	-7,4	-11,4	-5,2	-7,4	-8,1	-11,8	-14,4
b11	790	-1,0	-1,6	-1,6	-3,6	-4,9	-2,3	-3,4	-5,1	-6,8	-8,1
b12	1.676	-2,0	-2,3	-2,6	-3,3	-4,1	-2,1	-3,0	-4,7	-5,1	-6,5
AVG.	483	-2,1	-2,7	-3,3	-5,8	-6,6	-3,0	-4,8	-7,0	-9,0	-9,7

Source: own authorship

Figure 5.19: Variation in net count in each circuit with bounded method (left) and optimal method (right), compared to the reference value (middle).



Source: own authorship

Table 5.8: Net count variation (%), comparing BOUNDED and OPTIMIZED algorithms.

	Count	BOUNDED Algorithm					OPTIMIZED Algorithm				
CKT		8T	10T	12T	14T	16T	8T	10T	12T	14T	16T
b01	29	0,0	0,0	0,0	-10,3	-10,3	0,0	-6,9	-6,9	-17,2	-17,2
b02	17	0,0	0,0	-5,9	-17,6	-17,6	0,0	-5,9	-11,8	-17,6	-17,6
b03	116	-6,9	-6,9	-7,8	-11,2	-14,7	-6,9	-11,2	-17,2	-19,0	-20,7
b04	359	-3,3	-4,7	-5,0	-8,1	-6,4	-4,7	-7,8	-11,4	-12,3	-13,4
b05	381	-2,1	-2,1	-2,6	-4,2	-4,7	-3,9	-6,6	-7,9	-9,2	-9,7
b06	40	0,0	0,0	0,0	-5,0	-5,0	-2,5	-5,0	-5,0	-10,0	-10,0
b07	272	-0,7	-0,7	-0,7	-1,1	-1,1	-2,9	-4,0	-7,7	-8,1	-8,1
b08	114	-2,6	-6,1	-6,1	-9,6	-11,4	-7,0	-10,5	-14,0	-16,7	-18,4
b09	112	-9,8	-10,7	-10,7	-10,7	-10,7	-9,8	-10,7	-14,3	-17,0	-17,9
b10	130	-7,7	-8,5	-8,5	-9,2	-13,8	-8,5	-10,8	-12,3	-15,4	-18,5
b11	341	-2,1	-3,2	-3,2	-5,0	-5,6	-3,8	-5,9	-7,9	-9,7	-10,0
b12	778	-4,0	-4,4	-4,5	-5,4	-6,7	-4,2	-5,4	-8,1	-8,9	-11,3
AVG.	224	-3,3	-3,9	-4,6	-8,1	-9,0	-4,5	-7,6	-10,4	-13,4	-14,4

Source: own authorship

is in the exhaustive search it performs. A possible way to deal with this situation in future works is to use a lookup table to avoid processing twice the same arrangement of gates in a group. It is also possible to implement this part to support parallel execution, as the procedure to find the number of transistors of a new cell candidate to replace groups of cells in the original netlist is entirely independent.

5.5 Estimated impact in physical synthesis

Before we start to present the experiments realized in the scope of this evaluation, it is necessary to make clear that a serious analysis of the impact of our technique toward physical design should take into account much more details on physical aspects of the new cells we include in the circuit, such as gate sizing, extraction, and characterization, as we discuss in (CONCEIÇÃO; REIS, 2019). However, there are many related works already presented in the previous chapters of this thesis, some of them developed in our research group, showing the advantages of adopting SCCGs also when facing the challenges from the physical domain. So, the results presented in this section are only estimations.

We only use the OPTIMAL algorithm set to perform exhaustive search in up to 10 cells in the experiments presented from now on. Although not being the main focus of the thesis, we want to estimate the impacts of our technique over the circuit area. To do this, we use the Astran tool mostly with its default parameters (without tuning) to re-generate the layout of all combinational cells with a single output present in 45nm Nangate Open Cell Lilbrary (NOCL) in order to make a fair comparison.

The original .bench description of the circuits from the ITC99 benchmark is synthesized to the nocl45nm.genlib we described in the introduction of this chapter. The new cells that modify the original netlist by our technique are also generated with Astran, mostly using its default parameters, always with minimal dimensions. We assume that sizing is a future step, out-of-scope of the transistor count optimization procedure we are proposing in this thesis. Notice that we neglect possible area savings due to decreasing via count, although it could reduce the total area in the circuit even more. Therefore, the reported area of the circuits is given by the cell area.

5.5.1 Area analysis

We analyze 60 different netlists generated by setting the parameter of the OPTMIZED algorithm to allow new SCCGs with up to 8, 10, 12, 14, and 16 transistors, for 12 different circuits. It demanded to use ASTRAN to generate a layout for the 1230 new cells, which would be a challenging task to do without a layout design automation tool. We report the dimension of the new SCCGs to get the data we show in this section.

Affected Area

We report on Table 5.9 the affected area of the original netlist as the sum of the area of each cell replaced by our merging technique. As we can see, the technique affects an increasing portion of the circuit as we allow bigger SCCGs, achieving up to 29% when limiting the new cells to 16 transistors.

Table 5.9: Affected circuit area when using OPTIMAL algorithm, varying the maximum number of transistors in the new cells.

Ckt	8 tr.	10 tr.	12 tr.	14 tr.	16 tr.
b01	5,6%	12,5%	13,5%	23,4%	23,4%
b02	0,0%	0,0%	17,0%	20,6%	20,6%
b03	7,3%	10,5%	**20,7%**	21,8%	22,2%
b04	5,3%	9,2%	13,4%	14,0%	15,7%
b05	6,7%	12,0%	14,5%	16,8%	17,3%
b06	4,9%	10,5%	10,5%	13,0%	13,0%
b07	3,5%	4,9%	11,2%	11,9%	11,9%
b08	8,0%	13,1%	18,8%	20,4%	21,2%
b09	**9,8%**	9,8%	13,9%	17,7%	19,1%
b10	9,5%	**15,5%**	19,8%	**27,9%**	**29,6%**
b11	5,8%	9,4%	15,5%	19,8%	22,1%
b12	5,7%	8,0%	12,6%	15,0%	18,9%
MAX.	9,8%	15,5%	20,7%	27,9%	29,6%

Area savings

If we compare the cell area of the new cell (together with the inverters it demands) with the total area of the replaced group of cells, we can see that our SCCGs occupy an area 25% smaller on average, as shown in Table 5.10. This reduction in the small portions of the circuit affected by our technique results in an overall reduction in the area of the whole circuit.

Table 5.10: Compared cell area (in μm^2) of the new SCCGs and the group of cells they replace.

	8 transistors		10 transistors		12 transistors		14 transistors		16 transistors	
Ckt	group area	(%)	group area	(%)	group area	(%)	group area	(%)	group area	(%)
b01	3,3	0,0	7,3	-20,4	7,9	-11,3	13,8	-25,0	13,8	-25,0
b02	0,0	0,0	0,0	0,0	6,3	-31,0	7,6	-37,3	7,6	-37,3
b03	19,6	-26,0	28,1	-33,0	55,5	-28,6	58,5	-28,0	59,6	-30,4
b04	41,1	-30,3	71,2	-28,9	104,0	-27,5	108,9	-27,8	122,1	-27,1
b05	44,6	-26,5	79,8	-25,1	96,7	-25,9	111,8	-27,0	114,8	-27,4
b06	4,0	-18,5	8,7	-17,2	8,7	-20,7	10,8	-26,4	10,8	-25,0
b07	19,0	-31,8	27,1	-28,7	61,8	-21,5	65,4	-21,3	65,4	-22,7
b08	19,2	-32,8	31,1	-26,9	44,7	-22,1	48,5	-21,6	50,6	-26,0
b09	25,4	-32,4	25,4	-35,5	36,1	-30,7	45,9	-28,8	49,5	-26,3
b10	22,7	-30,3	37,3	-24,9	47,4	-25,2	66,7	-26,4	70,9	-26,9
b11	34,3	-24,5	56,0	-26,7	92,2	-20,3	117,9	-21,1	131,4	-21,6
b12	83,8	-31,6	117,8	-28,1	185,7	-24,0	221,2	-27,0	278,6	-26,2
AVG.	26,4	-29,1	40,8	-27,8	62,2	-24,6	73,1	-25,8	81,3	-25,9

When we measure the cell area of the whole circuit before and after applying our merging technique, we can see in Table 5.11 that we can achieve up to 7,9% reduction, and about 5.3% in average when we performed OPTIMIZED algorithm adjusted to generate new cells with up to 16 transistors. This reduction can be even more significant when we consider the area saved due to fewer pins, as it reduces the number of vias.

Table 5.11: Compared cell area (in μm^2) of the netlists before and after applying our gate merging technique

Ckt	before	8 transistors		10 transistors		12 transistors		14 transistors		16 transistors	
		Measure	%	Measure	%	Measure	%	Measure	%	Measure	%
b01	58,8	58,8	0,0	57,3	-2,5	57,9	-1,5	55,3	-5,9	55,3	-5,9
b02	37,0	37,0	0,0	36,2	-2,0	35,0	-5,3	34,1	-7,7	34,1	-7,7
b03	268,2	263,1	-1,9	259,0	-3,5	252,4	-5,9	251,9	-6,1	250,1	-6,8
b04	775,6	763,1	-1,6	755,1	-2,7	747,0	-3,7	745,3	-3,9	742,5	-4,3
b05	665,2	653,4	-1,8	645,2	-3,0	640,2	-3,8	635,0	-4,5	633,8	-4,7
b06	82,7	82,0	-0,9	81,2	-1,8	80,9	-2,2	79,9	-3,4	80,0	-3,3
b07	549,6	543,5	-1,1	541,8	-1,4	536,3	-2,4	535,7	-2,5	534,8	-2,7
b08	238,2	231,9	-2,6	229,8	-3,5	228,3	-4,1	227,7	-4,4	225,0	-5,5
b09	259,7	251,4	-3,2	250,6	-3,5	248,6	-4,3	246,4	-5,1	246,7	-5,0
b10	239,6	232,7	-2,9	230,3	-3,9	227,6	-5,0	222,0	-7,4	220,6	-7,9
b11	595,5	587,2	-1,4	580,6	-2,5	576,8	-3,1	570,7	-4,2	567,1	-4,8
b12	1.473,8	1.447,3	-1,8	1.440,7	-2,2	1.429,2	-3,0	1.414,1	-4,1	1.400,9	-4,9
AVG.	437,0	429,3	-1,6	425,7	-2,7	421,7	-3,7	418,2	-4,9	415,9	-5,3

5.5.2 Connection analysis

We also want to analyze how our technique influences circuit wiring. More than count how many nets or how many wire segments, as we have already reported, we want to estimate how it works for total wirelength in the circuit. To measure it, we generated bookshelf descriptions for every netlist used in this analysis. Then we place the cells in the circuit using placement tools like Dragon and Capo. Both tools use Half Perimeter Wire Length (HPWL) as a model to estimate wirelength.

Total wirelength analysis

We report in Table 5.12 the measurements made in the original input netlists and in the netlists obtained by applying our technique for different sizes allowed for the new SCCG gates, as we did in the previous area analysis. Different from our expectations, we can see that the total wirelength reported by the tools has increased for some circuits and parameter configurations. The total wirelength measures vary from -18% in b02 to +14% in b04 for Dragon placer, and from -21% in b01 circuit up to +47% in b12 for Capo placement tool.

Table 5.12: Compared total wirelength (in μm) of the circuits before and after applying our technique

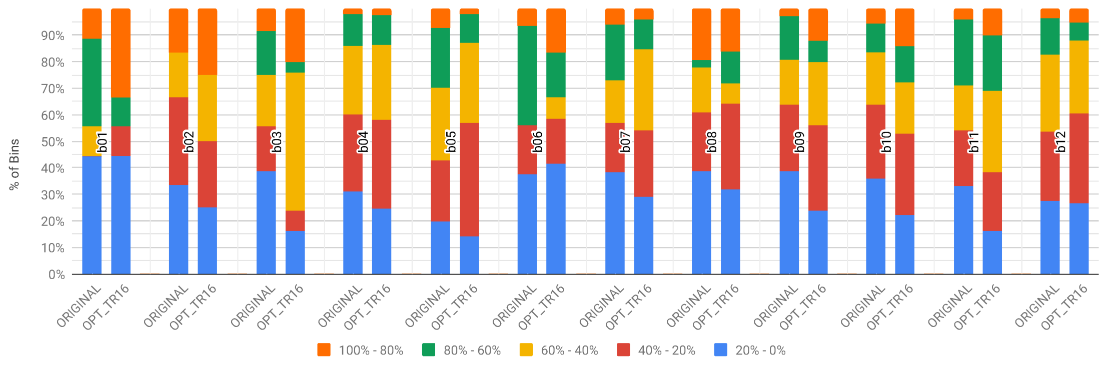
Ckt	DRAGON						CAPO					
	before	8T	10T	12T	14T	16T	measure	8T	10T	12T	14T	16T
b01	120,2	-7,1	1,4	-1,6	-7,8	-8,1	108,0	2,4	-6,8	-3,9	-21,3	-21,3
b02	67,5	5,9	-6,1	-13,4	-18,4	-14,9	59,9	2,7	0,6	6,5	2,2	2,2
b03	613,8	-7,0	-4,1	-3,8	-5,3	-8,9	464,5	4,8	16,9	4,1	21,6	17,1
b04	2058,0	13,3	10,8	14,3	15,1	11,5	1944,8	2,8	9,3	26,4	40,3	11,4
b05	1973,4	0,6	3,4	-2,5	0,9	-0,3	1990,3	1,2	3,0	-1,5	17,2	-2,2
b06	168,9	-2,2	1,7	1,4	-10,8	-7,1	134,4	10,9	14,2	3,8	-1,8	11,7
b07	1833,5	-0,6	-1,6	-1,5	0,2	-1,7	1457,1	4,7	4,8	17,3	22,2	29,1
b08	612,2	-3,0	-3,7	2,2	0,5	-0,3	535,6	6,1	10,0	12,6	-0,7	0,0
b09	594,4	1,3	3,3	-0,3	2,0	3,4	510,8	1,8	4,8	20,6	7,2	3,3
b10	690,2	0,3	-0,4	-0,3	-0,1	1,1	595,1	0,5	15,5	12,2	10,7	2,8
b11	2376,3	-1,1	-0,1	-3,4	0,4	0,6	2104,5	1,1	-0,7	5,9	-0,7	7,2
b12	4878,5	1,3	0,7	9,6	5,0	4,9	4439,9	3,6	6,0	26,3	47,5	10,6
AVG.	1332,2	0,1	0,4	0,1	-1,5	-1,7	1195,4	3,6	6,5	10,9	12,0	6,0

A possible explanation for this increase in total wirelength is the fact that we avoid to include new inverters by looking in the circuit for the inverted inputs the new gate demands. It may be getting a signal from a gate with fanout bigger than one, which is more likely to be kept by the placer far apart from the new gate when the circuit is placed again – thus demanding a longer wire. This longer wire shall be bigger than the sum of the length of the wires dropped during the merging process. Another clever strategy would be to consider the position of cells and merge then only if they are closer each other. This is another way to improve the proposed technique when applying it in physical synthesis.

Congestion estimation

We go a bit further to analyze wire congestion estimation in the circuit. We use the program CongestionMaps-Lnx64[5] to report congestion data. Congestion analysis works by dividing the circuit into smaller areas called bins, and defining a number of routing tracks each area supports. A given bin is considered congested when its available number of tracks is smaller than the estimated number of tracks that routing requires. In Figure 5.20, we show a chart with the distribution of bins congestion of each circuit before and after we apply our merging technique. By looking at Figure 5.21, which exhibit median, mean, and peak congestion values of each circuit, we can see that the congestion increases for the most circuits.

Figure 5.20: Wire congestion distribution in circuits before and after applying OPTIMIZED algorithm set to generate new cells with up to 16T.



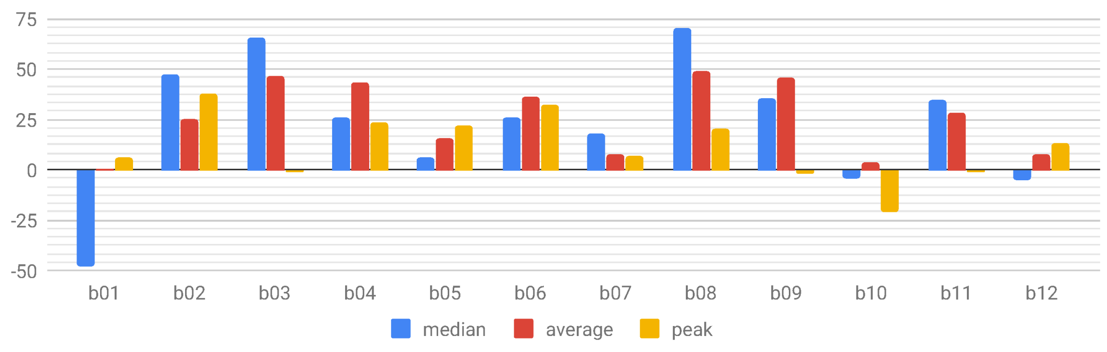
Source: own authorship

Wirelength distribution

On the other hand, although increasing congestion, our technique tends to eliminate shorter connections, as we can see in Figure 5.22. However, by observing the varia-

[5]Available: http://vlsicad.eecs.umich.edu/BK/PlaceUtils/

Figure 5.21: Variations of median, average and peak on congestion map of the circuits after applying the merging technique.

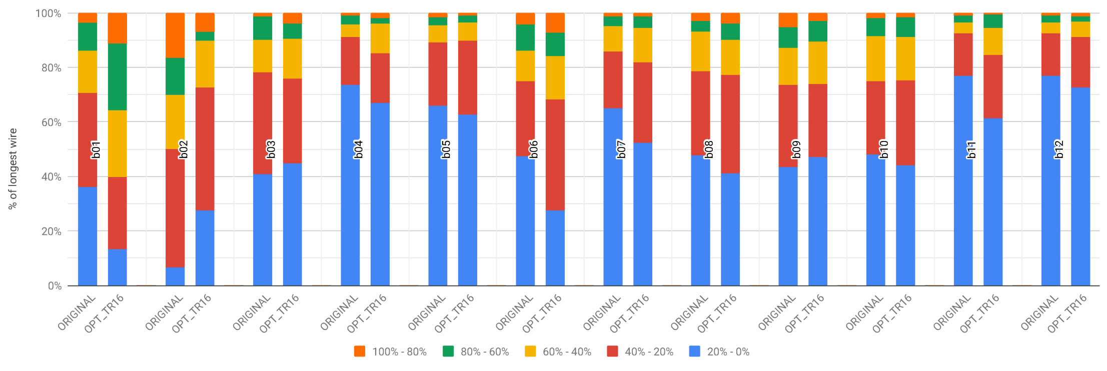


Source: own authorship

tion on median, average, and peak values of estimated wirelength in each circuit presented in Figure 5.23, we can notice that the longest wire reduces in most circuits. Median and average wire length increases, as a direct effect of eliminating shorter wires, which was predicted in (SEO et al., 2008).

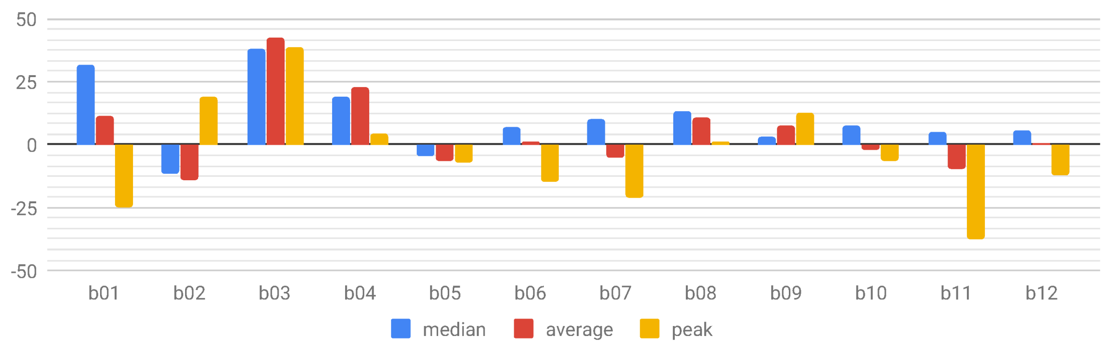
However, we also observe an increase in congestion as a side effect result mostly due to area reduction we achieve with our merging technique, which implies that cells are now closer to each other. The increasing on the demanded buffer insertion, the main argument sustained in (SEO et al., 2008) against using large cells, may be an overestimated problem. Anyway, it has to be measured in future works when performing this technique until physical synthesis.

Figure 5.22: Distribution of HPWL wirelength as quotas of the longest wire



Source: own authorship

Figure 5.23: Variations of median, average and peak on connections in the circuit after applying the merging technique.



Source: own authorship

5.6 Comparing to a commercial tool

We also performed tests to evaluate our technique when applied to a netlist generated with a commercial tool, targeting the Nangate Open Cell Library 45nm (NOCL45nm). After the netlist is generated, we replace every oversized cell by its unitary size version, in order to do a fair transistor count. Also, all cells in the original library were resynthesized using Astran in order to make a fair comparison in area and wirelength. We use this resulting netlist as input to the OPTIMIZED algorithm set to do the exhaustive search for groups with up to 10 connected unitary fanout gates, and allowing new SCCGs with up to 16T.

The results regarding transistor, instance, pin, nets, and wire segments count, as well as cell area and total wirelength estimations, are reported in Table 5.13. We can see that our technique can considerably reduce all the parameters, with a drawback in total wirelength estimation for some circuits. It demonstrates that there are significant optimization opportunities in the traditional digital design flow, and approaches not limited to the set of cells available in a cell library can do the job.

Table 5.13: Compared transistor, instance, pin, net, and wire segment count, and estimated cell area (in μm^2) and total wirelength (in μm) of netlists generated with a commercial tool (#) and the variation after applying our merging technique with OPTIMIZED algorithm, set to generate cells with up to 16 transistors

Ckt	transistor		instances		pins		wire segments		cell area		Total Wirelength	
	#	%	#	%	#	%	#	%	#	%	#	%
b01	344	0,87	42	0,69	137	0,79	86	0,81	72,6	0,84	177,5	0,87
b02	246	0,87	30	0,67	97	0,76	60	0,78	50,6	0,85	119,7	0,80
b03	1.460	0,95	106	0,63	422	0,81	280	0,85	283,3	0,84	577,3	0,84
b04	3.920	0,96	307	0,84	1.185	0,90	799	0,92	798,4	0,86	1.880,2	0,95
b05	3.088	0,92	416	0,78	1.386	0,84	933	0,86	677,4	0,86	2.025,1	1,04
b06	404	0,96	36	0,89	123	0,91	75	0,91	80,9	0,94	166,5	1,02
b07	2.916	0,93	276	0,79	986	0,86	666	0,88	624,9	0,84	1.667,8	0,94
b08	1.138	0,93	90	0,73	352	0,85	230	0,87	223,4	0,85	524,5	1,03
b09	1.448	0,91	149	0,60	511	0,75	331	0,80	287,9	0,85	775,6	0,83
b10	1.256	0,89	145	0,71	509	0,79	334	0,81	262,0	0,83	827,4	0,91
b11	2.716	0,92	364	0,80	1.207	0,85	804	0,86	603,0	0,86	1.953,6	1,06
b12	7.482	0,94	726	0,79	2.652	0,86	1.800	0,89	1.537,5	0,86	4.547,0	1,05
AVG.	2.202	0,92	224	0,74	797	0,83	533	0,85	458,5	0,86	1.270,2	0,94

6 CONCLUSIONS AND FUTURE WORKS

We have developed a systematic approach to minimize the transistor count, wires, and contacts of a circuit based on merging interconnected cells of unitary fanout present in the input netlist. Also, by adopting the proposed alternative flow towards a library free design of digital circuits, we can keep using some of the sophisticated tools designed to handle the standard cell-based design. The proposed method is library independent and can be adopted as part of a full-custom automated design methodology.

The experiments show the effectiveness of the proposed methodology as it is capable of reducing the transistor count by up to 13%, 8% on average when compared to original netlist generated with a traditional commercial logic synthesis tool. The technique also showed to be active on reducing the number of instances(up to 39%), wire segments (up to 21%), and contact count (up to 24%) in all test cases, even when compared to netlists generated with ABC targeting big virtual libraries. Fewer contacts means fewer vias, which also impacts on the overall area of the circuits.

The evaluation showed that the proposed merging technique is useful to reduce the number of transistors in a transistor network of a circuit no matter the cells used in the input netlist. Our OPTIMIZED algorithm showed the best results, always leading to incrementally better results when a higher degree of freedom is allowed. However, it still needs improvements regarding performance, as the algorithm is based on the exploration of all possible coverings of a group of connected cells and can take time to process all of them.

We also estimate the area impact of the technique by generating cells using ASTRAN and comparing the cell area of the netlists before and after our proposed transistor count optimization method. We observed an area reduction of 5.3% on average when compared to ABC results, and an average reduction of 14% when compared to a leading commercial logic synthesis tool. Such reductions tend to be even more significant, since we are reporting only the total cell area, without considering the area occupied by vias, also reduced with our technique. These results can be significant in recent technologies nodes where the delay of interconnections and vias has become dominant.

On the other hand, estimations made in the physical design domain reveals that our proposed technique may increase the wirelength as a payoff for some circuits. We impute this result to the decision of avoiding to insert new inverters by reusing signals existing in the circuit. Even though we reduce a considerable number of connections, in some cases, the sum of their lengths is smaller then the length of the new interconnections added to the new gate. The routing tool will probably need to insert buffers in the circuit to break the long wires in smaller ones, but this extra area shall not be larger than the area budget we get by applying our technique. Besides that, an increase in wirelength not necessarily makes unfeasible the circuit routing, as there is a reduction in the number of connections.

To summarize, the main contributions of this thesis are:

- A study about transistor count as a quality parameter of both academic and commercial tools;
- A novel method to achieve a library free design methodology of a digital circuits, that still can use the traditional standard cell framework of tools and formats;
- A set of applied algorithms to find all possible covers in a root-tree-like structure in a circuit;
- Three different algorithms to minimize transistor count in a netlist using the proposed merging technique;
- A systematic procedure to reduce transistor count in a circuit, also capable of reducing wires and vias, and potentially useful to reduce area and power consumption; and
- A new tool to compose the TRANCA methodology of our research group, to work with the layout automation tool ASTRAN or any new one.

6.1 Future Works

The developed tool needs minor improvements on the user interface. It also has to be improved seeking performance, especially in the implementation of the OPTIMIZED algorithm to avoid process again repeated groups of cells. A possible way to achieve this goal is to use data structures like lookup tables to store intermediary results. It is also possible to re-implement the exhaustive algorithm to compute data of each cover in parallel, as each cover is independent.

It is still necessary to improve the method to take into consideration the placement information of the cells on deciding to insert new inverting cells. Further, one can improve the proposed technique on physical domain by adding the relative position of cells in the criteria to merge cells, merging only cells that are closer to each other. Also, the reasons for the unwanted results regarding total wirelength in some circuits and the actual impact of it in circuit routability must be investigated in future works.

A possible unfolding of this thesis is to integrate a continuous sizing tool to attend specified timing goals. It is also in our plans to perform a more in-depth validation of the proposed method in the physical domain, especially considering more recent technology nodes.

Another possible investigation derived from this thesis is to study how the proposed library free design flow can help to improve the security of circuits, as the unlimited number of cell layout favors camouflage of the circuit, thus making reverse engineering difficult.

REFERENCES

ALBRECHT, C. Iwls 2005 benchmarks. In: **Proc. Int. Workshop Logic Synthesis (IWLS 2005)**. [S.l.: s.n.], 2005. Available in: iwls.org/iwls2005/benchmark_presentation.pdf.

AMARÚ, L.; GAILLARDON, P. E.; MICHELI, G. D. Mixsyn: An efficient logic synthesis methodology for mixed xor-and/or dominated circuits. In: **2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2013. p. 133–138. ISSN 2153-6961. DOI:10.1109/ASPDAC.2013.6509585.

BAMPI, S.; REIS, R. Challenges and emerging technologies for system integration beyond the end of the roadmap of nano-cmos. **VLSI-SoC: Technologies for Systems Integration**, Springer, p. 21–33, 2011. DOI:10.1007/978-3-642-23120-9_2.

BERKELAAR, M.; JESS, J. Technology mapping for standard-cell generators. In: **IEEE International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 1988. p. 470–473. DOI:10.1109/ICCAD.1988.122551.

Cardoso, M. S. et al. Libra: An automatic design methodology for cmos complex gates. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 65, n. 10, p. 1345–1349, Oct 2018. ISSN 1549-7747. DOI: 10.1109/TCSII.2018.2866231.

CONCEIÇÃO, C. et al. A cell clustering technique to reduce transistor count. In: **IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. [S.l.: s.n.], 2017. p. 186–189. DOI:10.1109/ICECS.2017.8291996.

CONCEIÇÃO, C.; POSSER, G.; REIS, R. Reducing the number of transistors with gate clustering. In: **IEEE Latin American Symposium on Circuits Systems (LASCAS)**. [S.l.: s.n.], 2016. p. 163–166. DOI:10.1109/LASCAS.2016.7451035.

CONCEIÇÃO, C. M. O.; REIS, R. A. L. Transistor count reduction by gate merging. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 66, n. 6, p. 2175–2187, June 2019. ISSN 1549-8328. DOI: 10.1109/TCSI.2019.2907722.

CONG, J.; DING, Y. On area/depth trade-off in lut-based fpga technology mapping. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 2, n. 2, p. 137–148, June 1994. ISSN 1063-8210. DOI:10.1109/92.285741.

DETJENS, E. et al. Technology Mapping in MIS. In: IEEE. **Proc. of the Int. Conf. on Computer Aided Design**. [S.l.], 1987. p. 116–119.

FLACH, G. et al. Effective method for simultaneous gate sizing and$v$th assignment using lagrangian relaxation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 33, n. 4, p. 546–557, April 2014. ISSN 0278-0070. DOI:10.1109/TCAD.2014.2305847.

GAVRILOV, S. et al. Library-less synthesis for static cmos combinational logic circuits. In: **IEEE/ACM International Conference on Computer-aided Design**. Washington, DC, USA: IEEE Computer Society, 1997. (ICCAD '97), p. 658–662. ISBN 0-8186-8200-0. Available from Internet: <https://dl.acm.org/citation.cfm?id=266388.266570>.

GHANE, M.; ZARANDI, H. R. Gate merging: An nbti mitigation method to eliminate critical internal nodes in digital circuits. In: **Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)**. [S.l.: s.n.], 2016. p. 786–791. ISSN 2377-5750. DOI:10.1109/PDP.2016.90.

GUIMARãES, D. S.; PUGET, J.; REIS, R. A. L. A mixed cells physical design approach. In: **2015 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2015. p. 1446–1449. ISSN 0271-4302. DOI:10.1109/ISCAS.2015.7168916.

JIANG, Y.; SAPATNEKAR, S. S. An integrated algorithm for combined placement and libraryless technology mapping. In: **IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 1999. p. 102–105. ISSN 1092-3152. DOI: 10.1109/ICCAD.1999.810630.

KAGARIS, D. Moto-x: A multiple-output transistor-level synthesis cad tool. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 35, n. 1, p. 114–127, Jan 2016. ISSN 0278-0070. DOI:10.1109/TCAD.2015.2448675.

KAHNG, A. B. et al. Timing closure. In: ____. **VLSI Physical Design: From Graph Partitioning to Timing Closure**. Dordrecht: Springer Netherlands, 2011. p. 219–264. ISBN 978-90-481-9591-6. DOI:10.1007/978-90-481-9591-6_8.

KNUTH, D. E. **The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms**. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997. 373 p. ISBN 0-201-89683-4.

MARKOV, I. L.; HU, J.; KIM, M. Progress and challenges in vlsi placement research. **Proceedings of the IEEE**, v. 103, n. 11, p. 1985–2003, Nov 2015. ISSN 0018-9219. DOI: 10.1109/JPROC.2015.2478963.

MARQUES, F. S. et al. Dag based library-free technology mapping. In: **ACM Great Lakes Symposium on VLSI**. New York, NY, USA: ACM, 2007. (GLSVLSI ’07), p. 293–298. ISBN 978-1-59593-605-9. DOI: 10.1145/1228784.1228857.

MARTINS, M. G. A. et al. Boolean factoring with multi-objective goals. In: **IEEE International Conference on Computer Design (ICCD)**. [S.l.: s.n.], 2010. p. 229–234. ISSN 1063-6404. DOI:10.1109/ICCD.2010.5647772.

MATOS, J. M. et al. Deriving reduced transistor count circuits from aigs. In: **IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.: s.n.], 2014. p. 1–7. DOI:10.1145/2660540.2661008.

MICHELI, G. D. **Synthesis and optimization of digital circuits**. [S.l.]: McGraw-Hill Higher Education, 1994.

MORAES, F. G. et al. A Physical Synthesis Design Flow based on Virtual Components. In: **DCIS’00: XV Design of Circuits and Integrated Systems Conference**. Montpellier, France: [s.n.], 2000. p. 740–745. Available from Internet: <https://hal-lirmm.ccsd.cnrs.fr/lirmm-00239439>.

MURGAI, R. Technology-dependent logic optimization. **Proceedings of the IEEE**, v. 103, n. 11, p. 2004–2020, Nov 2015. ISSN 0018-9219. DOI:10.1109/JPROC.2015.2484299.

NEWTON, A. R. A survey of computer aids for vlsi layout. In: **IEEE Symposium on VLSI Technology**. [S.l.: s.n.], 1982. p. 72–75.

ONODERA, H.; HASHIMOTO, M.; HASHIMOTO, T. Asic design methodology with on-demand library generation. In: **IEEE Symposium on VLSI Circuits (VLSIC)**. [S.l.: s.n.], 2001. p. 57–60. DOI: 10.1109/VLSIC.2001.934194.

PILATO, C.; FERRANDI, F.; PANDINI, D. A design methodology for the automatic sizing of standard-cell libraries. In: **ACM Great Lakes Symposium on VLSI**. New York, NY, USA: ACM, 2011. (GLSVLSI '11), p. 151–156. ISBN 978-1-4503-0667-6. DOI: 10.1145/1973009.1973040.

POSSANI, V. N. et al. Graph-based transistor network generation method for supergate design. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 24, n. 2, p. 692–705, Feb 2016. ISSN 1063-8210. DOI:10.1109/TVLSI.2015.2410764.

POSSER, G. et al. Transistor sizing and gate sizing using geometric programming considering delay minimization. In: **10th IEEE International NEWCAS Conference**. [S.l.: s.n.], 2012. p. 85–88. DOI:10.1109/NEWCAS.2012.6328962.

POSSER, G. et al. A study on layout quality of automatic generated cells. In: **2010 17th IEEE International Conference on Electronics, Circuits and Systems**. [S.l.: s.n.], 2010. p. 651–654. DOI: 10.1109/ICECS.2010.5724596.

RABAEY, J. M.; CHANDRAKASAN, A. P.; NIKOLIC, B. **Digital Integrated Circuits**. [S.l.]: Prentice hall Englewood Cliffs, 2002. ISBN 1-40207-075-6.

REIMANN, T.; SZE, C. C.; REIS, R. Challenges of cell selection algorithms in industrial high performance microprocessor designs. **Integration**, v. 52, p. 347 – 354, 2016. ISSN 0167-9260. DOI:10.1016/j.vlsi.2015.09.001.

REIS, A. I. Covering strategies for library free technology mapping. In: IEEE COMPUTER SOCIETY. **Proc. of the IEEE Computer Society Symp. on Integrated Circuit Design and System Design**. [S.l.], 1999. p. 0180–0180. DOI:10.1109/SBCCI.1999.803115.

REIS, A. I. et al. Library free technology mapping. In: **VLSI: Integrated Systems on Silicon**. [S.l.]: Springer, 1997. p. 303–314. DOI:10.1007/978-0-387-35311-1_25.

REIS, R. Physical design automation at transistor level. In: **2008 NORCHIP**. [S.l.: s.n.], 2008. p. 241–245. DOI:10.1109/NORCHP.2008.4738270.

REIS, R. Design automation of transistor networks, a new challenge. In: **IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2011. p. 2485–2488. ISSN 0271-4302. DOI:10.1109/ISCAS.2011.5938108.

REIS, R. Power consumption & reliability in nanocmos. In: **IEEE Conference on Nanotechnology (IEEE-NANO)**. [S.l.: s.n.], 2011. p. 711–714. ISSN 1944-9399. DOI:10.1109/NANO.2011.6144656.

ROY, R.; BHATTACHARYA, D.; BOPPANA, V. Transistor-level optimization of digital designs with flex cells. **Computer**, v. 38, n. 2, p. 53–61, Feb 2005. ISSN 0018-9162. DOI:10.1109/MC.2005.74.

SCARTEZZINI, G.; REIS, R. Power consumption in transistor networks versus in standard cells. In: **IEEE International Conference on Electronics, Circuits, and Systems (ICECS)**. [S.l.: s.n.], 2011. p. 740–743. DOI:10.1109/ICECS.2011.6122380.

SCHNEIDER, F. R. **Building transistor-level networks following the lower bound on the number of stacked switches**. Thesis (MSc in Computer Science) — Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS - Brazil, 2007. Available in: http://hdl.handle.net/10183/55446.

SEO, J. s. et al. On the decreasing significance of large standard cells in technology mapping. In: **2008 IEEE/ACM International Conference on Computer-Aided Design**. [S.l.: s.n.], 2008. p. 116–121. ISSN 1092-3152. DOI:10.1109/ICCAD.2008.4681561.

SILVA, L. M. da. **Minimização Lógica por Fusão de Portas**. Thesis (MSc in Microlectronics) — Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS - Brazil, 2017. Available in: http://hdl.handle.net/.

SMANIOTTO, G. et al. A post-processing methodology to improve the automatic design of cmos gates at layout-level. In: **IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. [S.l.: s.n.], 2017. p. 42–45. DOI:10.1109/ICECS.2017.8292073.

TONFAT, J.; FLACH, G.; REIS, R. Leakage current analysis in static cmos logic gates for a transistor network design approach. In: **IEEE International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)**. [S.l.: s.n.], 2016. p. 107–113. DOI:10.1109/PATMOS.2016.7833673.

WEBER, O. Fdsoi vs finfet: differentiating device features for ultra low power amp;amp; iot applications. In: **IEEE International Conference on IC Design and Technology (ICICDT)**. [S.l.: s.n.], 2017. p. 1–3. DOI:10.1109/ICICDT.2017.7993513.

XUE, J.; AL-KHALILI, D.; ROZON, C. N. Tree-based transistor topology extraction algorithm for library-free logic synthesis. In: **IEEE International Conference on Semiconductor Electronics**. [S.l.: s.n.], 2004. p. 5 pp.–. DOI:10.1109/SMELEC.2004.1620879.

ZIESEMER, A.; REIS, R. Simultaneous two-dimensional cell layout compaction using milp with astran. In: **IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2014. p. 350–355. DOI:10.1109/ISVLSI.2014.79.

ZIESEMER, A. et al. Automatic layout synthesis with astran applied to asynchronous cells. In: **IEEE Latin American Symposium on Circuits and Systems (LASCAS)**. [S.l.: s.n.], 2014. p. 1–4. DOI:10.1109/LASCAS.2014.6820314.

ZIESEMER, A. M.; REIS, R. A. L. Simultaneous two-dimensional cell layout compaction using milp with astran. In: **IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2014. p. 350–355. ISSN 2159-3469. DOI:10.1109/ISVLSI.2014.79.

ZIESEMER JR., A. M. **Síntese Automática do Leiaute de Redes de Transistores**. Thesis (PhD in Microlectronics) — Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS - Brazil, 2014. Available in: http://hdl.handle.net/10183/97852.

APPENDIX A

Here we present an example of the configuration file as used in the current version of the tool. It is a plain text file where commented lines start with #, and valid lines are composed of a pair parameter-value, separated by white space.

config_45nm

```
######################################################
# UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  - UFRGS #
# INSTITUTO DE INFORMATICA                   - INF   #
# PROGRAMA DE POS GRADUACAO EM COMPUTACAO    - PPGC  #
# Autor: Calebe Micael de Oliveira Conceicao         #
# Orientador: Ricardo Augusto da Luz Reis            #
######################################################
# path of the input Verilog file
netlist     b12.v
# path of the .lib file
library     NOCL45nm_functional_with_astran_area.lib
# name of the weakest inverter available in the library
invgate     INV_X1
# name of the output pin of the inverter
output      ZN
# name of the input pin of the inverter
input       A
# default flipflop
defaultff   DFFR_X1
# name of the inverted output pin of the FF gate
invffout      QN
# length of the transistor in the technology
l_size      50
# prefered width of the transistor P
wpsize      630
# prefered width of the transistor N
wnsize      415
# name of the power pin
vddpin      VDD
# name of the ground pin
gndpin      VSS
# default unit of length. It will be used in spice
unit        n
```

```
# name of the NMOS transistor
nmos_t      nmos
# name of the PMOS transistor
pmos_t      pmos
# defining how to merge.
#    "unbounded" -> without any limitation.
#    "bounded" -> limited by serial transistor.
#    "optimized" -> bounded, using exhaustive search method
method     optimized
# maximum number of serial transistors.
serial      4
# maximum number of nodes to perform exhaustive search
maxnodeopt   10
# maximum number of transistors the new SCCG can have.
#    valid only in bounded and optimized methods
maxnumtrs   16
# enable adding inverters when an inverted input is needed
#    assign false only when estimating using unbounded method
addinvs     true
```

THESIS SUMMARY IN PORTUGUESE

Esta tese se concentra na otimização da tecnologia de circuitos integrados visando a diminuição do número de transistores necessários para implementá-lo. Essa demanda fica patente ao se observar o número reduzido de funções lógicas que uma biblioteca de células tradicional usualmente fornece, o que representa uma limitação inerente à otimização do número de transistores no circuito, algo que também, conforme estudos anteriores, tem influência direta sobre as métricas usuais de desempenho do circuito como área, dissipação de energia e atraso. Nesse contexto, uma abordagem de projeto livre de bibliotecas se faz necessária para obter circuitos otimizados, usando ferramentas para permitir a síntese de layout de qualquer rede de transistores. Observando a taxonomia do método adotado nessa tese, ele se encaixa como um processo customizado usando células compiladas, conforme ilustrado na Figura 1.3 do texto principal.

O objetivo desta tese é avaliar a efetividade do método proposto e desenvolver algoritmos para otimizar a netlist lógica de um circuito que usa como estratagema a realização de substituições de grupos de células simples por células complexas, visando reduzir o número de transistores. Tomamos como entrada para nossos algorítimos a netlist gerada no fluxo de síntese lógica tradicional para células padrão, a mesma que serve como entrada para as ferramentas de projeto físico do circuito, bem como informações da biblioteca, e parâmetros definidos pelo usuário da ferramenta em que os algoritmos estão implementados. A netlist é então processada para sistematicamente substituírmos agrupamentos de células combinacionais conexas de fanout unitário por uma nova porta complexa com lógica equivalente, gerada sob demanda para reduzir o número de transistores. A nova porta complexa normalmente não está disponível na biblioteca de células tradicional, e por ser gerada automaticamente para a demanda pontual – usando para isso uma ferramenta de geração automática de leiaute tal qual o ASTRAN – ela possui uma rede de transistores personalizada, que pode ser adequadamente organizada e dimensionada para atender aos requisitos específicos da porção do circuito onde se insere.

As abordagens livres de biblioteca disponíveis na literatura usualmente provêem um número extenso de funções lógicas habilitadas pela adoção de biliotecas virtuais como as de formato *.genlib*, que são descrições muito mais simplificadas das características da células básicas que as caracterizações encontradas nos formatos *liberty*, de modo a permitir lidar com a complexidade que um número grande de funções confere ao processo de mapeamento tecnológico. Por sua vez, a metodologia adotada nesta tese consiste em uma forma alternativa de ampliar o número de funções lógicas utilizadas para implementar um circuito lógico, e apresenta resultados similares aos alcançados com ferramentas conhecidas no meio acadêmico e industrial. Como resultado, a ferramenta desenvida para esta tese suporta a realização paralela da substituição de agrupamento de células no circuito, enquanto apresenta resultados similares às ferramentas desenvolvidas anteriormente independente da complexidade do conjunto de funções lógicas usadas para mapear a netlist de entrada para nosso método. Isso significa que a técnica aqui proposta pode partir de mapeamentos iniciais tão simples quanto os que fazem uso apenas de células básicas como NANDs e NORs, e ainda assim é capaz de reduzir o número de transistores a quantidades equivalentes às alcançadas em abordagens livres de biblioteca anteriores que fazem uso de bibliotecas virtuais com milhares de funções lógicas.

Três algoritmos principais foram desenvolvidos no arcabouço dessa tese para realizar a identificação, agrupamento e substituição de células em uma netlist, a qual é representada internamente como um grafo. Os Algoritmos 3, 4 e 8 apresentados no Capítulo 3 dessa tese são complementares entre si, adequados para avaliar características ou realizar otimizações distintas dentro do método proposto. Cada algoritmo e os respectivos resultados são tema das três principais publicações realizadas no curso de desenvolvimento desta tese (CONCEIÇÃO; POSSER; REIS, 2016; CONCEIÇÃO et al., 2017; CONCEIÇÃO; REIS, 2019). O Capítulo II desta tese contém um detalhado passo a passo da metodologia proposta, e do funcionamento geral dos algoritmos para um circuito básico de exemplo.

As principais contribuições dessa tese compreendem os algoritmos propostos e os estudos e discussões realizados, que buscam responder às questões que naturalmente emergem quanto às consequências da aplicação do método, a exemplo do real impacto sobre congestionamento e comprimento dos fios, e quanto ao impacto potencial de otimização da técnica aplicada a uma dada netlist. Além disso, foram feitas estimativas sobre as métricas usuais de desempenho do circuito, como o impacto sobre a área e sobre o consumo do circuito, apesar da tese estar concentrada nas otimizações possíveis de serem feitas no escopo da síntese lógica.

Os experimentos realizados mostram que a abordagem proposta é capaz de reduzir o número de transistores em todo o circuito em até 13 % em comparação com netlists geradas usando outras ferramentas de minimização, independentemente do tamanho da biblioteca de células padrão usada inicialmente para sintetizar a netlist original. Também reduzimos o número de instâncias (consequentemente contatos e conexões) nos experimentos realizados em 14%, (11% e 10% em média, respectivamente), quando comparados com a netlist gerada usando uma ferramenta acadêmica líder em síntese lógica. Investigamos também o impacto da otimização proposta na área e comprimento de fio, alcançando estimativas de redução média de 5 % na área e de até 14 % no comprimento total de fio. Por outro lado, conjectura-se que a aplicação da técnica aqui proposta será mais eficaz se realimentada com dados extraídos do processo de síntese física, como informações do posicionamento (real ou estimado) das células consideradas nas substituições, como forma de aprimorar os resultados obtidos quanto ao congestionamento do circuito. Para além disso, restam evidenciadas as oportunidades de otimização possíveis de serem alcançadas ao romper com o paradigma de concepção de circuito digitais usando células básicas como blocos construtivos para direcionar à uma concepção livre de biblioteca, que enxerga o circuito de forma transparente como um agrupamento de redes de transistores em um fluxo full custom automatizado, que permite maiores graus de customização.