PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
UNDERGRADUATE PROGRAM IN COMPUTER ENGINEERING

EXHAUSTIVE AND GENETIC APPROACHES TO TRANSISTOR SIZING OF CMOS DIGITAL NETWORKS

ALVIN LAURO BESKOW

Monograph presented as partial requirement for obtaining the degree of Bachelor in Computer Engineering at Pontifical Catholic University of Rio Grande do Sul.

Advisor: Prof. Dr. Ney Laert Vilar Calazans
Co-Advisor: MsC. Marcos Luiggi Lemos Sartori

Porto Alegre
2020

Eu dedico esse trabalho à minha família, principalmente à minha mãe que sem ela eu não teria conseguido chegar tão distante, ao meu pai que me ensinou muito sobre a vida e as pessoas, à minha irmã e ao meu cunhado que há pouco tempo se tornaram pais do Davi, garoto que só traz alegrias. Dedico esse trabalho ao Ney, que além de Orientador é um grande amigo e uma pessoa que admiro muito, ao Marcos que nunca mede esforços para auxiliar em qualquer dúvida ou problema. Também dedico esse trabalho aos meus amigos, especialmente ao Lucas, que sempre me lembram que existe vida fora da Universidade. À PUCRS e ao CNPq por me permitirem ter participado do grupo de pesquisa GAPH como bolsista. Agradeço também à UNISC que é minha universidade de origem, em especial ao Jacques, por ter me introduzido à IA e pela amizade que foi construída nesse processo.

I dedicate this work to my family, especially to my mother, who without her I would not have been able to reach so far, to my father who taught me a lot about life and people, to my sister and my brother-in-law who recently became David's parents, a boy who only brings joy. I dedicate this work to Ney, who in addition to being an Advisor is a great friend and a person I admire a lot, to Marcos who never goes to great lengths to help with any questions or problems. I also dedicate this work to my friends, especially Lucas, who always remind me that there is life outside the University. To PUCRS and CNPq for allowing me to have participated in the GAPH research group as a fellow. I also thank UNISC, which is my home university, especially Jacques, for introducing me to AI and for the friendship that was built in this process.

"Don't find fault, find a remedy; anybody can complain."
(Henry Ford)

ABORDAGENS EXAUSTIVA E GENÉTICA PARA DIMENSIONAR TRANSISTORES DE REDES DIGITAIS CMOS

RESUMO

O processo de projeto de sistemas digitais personalizados moderno usa sobretudo duas opções: dispositivos reconfiguráveis, ou Field Programmable Gate Arrays (e.g. FPGAs) e circuitos integrados dedicados ou Application Specific Integrated Circuits (ASICs). Ambas opções permitem, em princípio, o desenvolvimento de qualquer especificação de circuito digital, cada uma oferecendo ao projetista diferentes compromissos entre métricas tais como desempenho, consumo de energia, segurança etc. O projetista de ASICs frequentemente usa como recurso de implementação módulos pré-definidos e pré-caracterizados, disponibilizados em uma biblioteca. Tais bibliotecas consistem em conjuntos de circuitos de pequeno porte (e.g. portas lógicas) pré-projetados e pré-caracterizados, onde cada tal circuito é denominado uma célula. Células são tipicamente compostas como um rede de transistores interconectados que implementa uma função lógica bem definida. Assumindo o emprego de tecnologias planares (as mais comuns hoje), cada transistor desta rede é caracterizado pela sua largura (width ou W) e pelo seu comprimento do canal (length ou L). Projetistas de ASICs costumam empregar bibliotecas de células padronizadas, fornecidas por vendedores tais como ARM, Silvaco, Dolphin Integration e Faraday, e utilizam ferramentas de análise estática de temporização (static timing analysis ou STA) para selecionar a célula mais indicada para cada ponto de um circuito complexo (com milhares ou milhões de células). Como a vasta maioria dos projetos atuais emprega o método de projeto síncrono, bibliotecas de células comerciais com elementos dedicados a dar suporte a outros tipos de projeto digital são virtualmente inexistentes. Como exemplo de métodos de projeto sem o suporte citado, pode-se citar o projeto assíncrono quasi-delay insensitive (QDI), que embora podendo oferecer vantagens ao projeto baseado em células, não tem suporte de praticamente nenhuma biblioteca de células comercial. Isto faz com que circuitos gerados via projeto QDI tendam a possuir características sub-ótimas. Assim, o desenvolvimento de bibliotecas de células para circuitos QDI pode se beneficiar de um ferramenta de dimensionamento de redes de transistores. Esse trabalho descreve a proposta e implementação de uma ferramenta genérica voltada para o dimensionamento de redes de transistores, denominada TBSizer. TBSizer emprega extensivamente simulação elétrica para atender uma função custo pré-definida pelo usuário. TBSizer é uma ferramenta flexível para atender as necessidades de múltiplos conjuntos alternativos de requisitos de dimensionamento. Algumas características da ferramenta TBSizer são que ela: (i) facilita a seleção e configuração da tecnologia usada para descrever uma rede de transistores; (ii) delega a proposta de função custo ao usuário, que define a métrica ou combinação de métricas a usar durante o processo de dimensionamento; (iii) permite definir múltiplos grupos de transistores a dimensionar da mesma maneira, o que habilita modular o desempenho da execução do processo de dimensionamento; (iv) utiliza uma abordagem de dimensionamento voltada a estímulos dos arcos lógicos da rede, o que permite otimizar os caminhos críticos mais significativos do circuito. O software TBSizer habilita a escolha de uma entre duas estratégias de busca no espaço de soluções de dimensionamento: uma exaustiva e uma baseada em algoritmo genético. Resultados do uso experimental da ferramenta mostram-se promissores, incluindo: (1) a redução do atraso de propagação em comparação com resultados obtidos por outra ferramenta de dimensionamento disponível, o ROGen/CeS; (2) a melhoria das caracteríticas gerais de células, a partir do aumento de parâmetros considerados no processo de dimensionamento; (3) células de uma biblioteca comercial, a ARM TSMC 180nm Sage-X, tiveram seu dimensionamento reprojetado, obtendo redução no consumo energético em relação às células originais.

Palavras Chave: Dimensionamento de transistores, Arcos, Pré-vetores, Redes CMOS digitais, Algoritmo genético, Algoritmo exaustivo, Simulação elétrica, SPICE.

EXHAUSTIVE AND GENETIC APPROACHES TO TRANSISTOR SIZING OF CMOS DIGITAL NETWORKS

ABSTRACT

The design process of modern custom digital systems overwhelmingly employs one of two options: reconfigurable devices (e.g. FPGAs) and dedicated integrated circuits (ICs), also known as Application Specific Integrated Circuits (or ASICs). Both options in principle allow the development of any digital circuit specification, but with distinct trade-offs regarding design metrics such as performance, energy consumption, security etc. ASIC designs often employ pre-defined and pre-characterized modules available in libraries as a basic implementation resource. Such libraries consists in a set of small, pre-designed and pre-characterized circuits (e.g. logic gates), where each such circuit is called a cell. Cells are typically formed as a network of interconnected transistors that implement a well-defined logic function. Assuming the use of planar technologies (currently the most common), each transistor in this network is characterized by its channel width (W) and length (L). ASIC designers often employ standard cell libraries furnished by vendors such as ARM, Silvaco, Dolphin Integration and Faraday. In combination with these, designers use static timing analysis (STA) tools to select the most indicated cell for each point in a complex circuit (with thousands or even millions of cells). As the vast majority of current digital designs employ the synchronous design method, commercial cell libraries are accordingly virtually non-existent. As an example of design methods unsupported by commercial cell libraries there is the asynchronous quasi-delay insensitive (QDI) design. Although digital design could benefit from the use of QDI techniques, generating QDI circuits from ordinary commercial cell libraries tend to produce sub-optimal circuits. Thus, the development of libraries to support QDI design can benefit from a tool to dimension transistor networks. This work describes the proposal and implementation of a generic transistor network sizing tool, called TBSizer. TBSizer extensively employs electrical simulation to fulfill a user-defined cost function. TBSizer is a flexible, used to cover the needs of multiple alternative sizing requirements. Some characteristics of the TBSizer tool are that it: (i) facilitates the selection and configuration of specific technologies used to describe transistor networks; (ii) delegates the choice of a cost function to the tool user, which is best positioned to define the metric or metric combinations to employ during the sizing process; (iii) allows defining transistor groups to be sized in the same way, enabling to modulate the dimensioning process execution performance; (iv) employs a sizing approach based

Keywords: Transistor sizing, Logic arcs, Pre-vectors, CMOS digital networks, Exhaustive algorithm, Genetic algorithm, Electrical simulation, SPICE.

CONTENTS

1. INTRODUCTION AND MOTIVATION

Custom logic design has currently two major strands, Field Programmable Gate Arrays (FPGAs), and Application Specific Integrated Circuits (ASICs). FPGAs are reconfigurable logic hardware that is able to emulate the behavior of virtually any digital circuit specification. They are useful for prototyping, for small-scale deployment of large circuits in products, and are specially interesting for applications that require in-field reconfiguration. FPGAs have the advantage of low cost for small-scale production, but their cost increases quickly for large-scale deployment. Furthermore, FPGAs also present sub-par performance and power consumption figures. For applications where FPGAs are unsuitable and sales volume are expected to be large or very large, ASICs are normally the custom hardware target design choice.

A commonly used ASIC design style that drastically reduces development costs is *cell-based design*, a family of styles of which the most relevant representative technique is based on the use of standard cell libraries. Standard cell-based designs address complexity by relying on a pre-designed and pre-characterized set of self-contained logic blocks called *cells*. Each cell has limited, known dimensions, and is described by a transistor network that often implements a single logic function. Cells are designed for a specific technology, e.g. TSMC 180nm bulk CMOS, STMicroelectronics 28nm FDSOI etc. They are distributed in the form of libraries with a more or less pre-defined format, comprising, for each cell, a basic set of *views*. Electrical characterization, layout, abstract descriptions for use by automated synthesis tools, and behavioral are some of the often packaged views in any standard cell library. Together with these views, libraries come with extensive documentation and a set of rules for their use.

Designers often describe circuits at the behavioral level, using a Hardware Description Language (HDL). Synthesis tools read HDL descriptions and can then select cells from the library to implement the IC and/or its modules.

It is also possible to develop custom cells to incorporate in an ASIC design. These might be required when creating specialized modules or unconventional logic, e.g. to support asynchronous circuit design. The development of a custom standard cell requires designing the transistor network that implements the cell logic function. Each transistor is mainly characterized by its channel width (W) and channel length (L) parameters. Of course, there are other parameters that influence transistor behavior, such as drain and source areas and perimeters, transistor threshold voltage (V_t), which foundries often offer in a set of variations such as $LowV_t$, $StandardV_t$ and $HighV_t$, etc. However, the main parameter manipulated at the network design time is effectively W, and sometimes L. Other parameters may be the subject of a design specification and are already fixed at design time, such as the transistor V_t type choice. The values of W and L impact the electrical characteristics of the transistor. This, by consequence, impacts the timing and power characteristics of a cell or of any transistor network. The size of the different devices in a complex or even in simple transistor networks presents a major impact on the cell characteristics. The relationship between these sizes and the operational characteristics of the network are not always obvious, specially when considering different regimes like near-threshold or subthreshold. It is possible to use simplifying assumptions to predict the impact of a transistor size. However, on complex networks these assumptions might not always hold. It is also possible to size transistor networks by exploring different sizes for each transistor and using precise electrical simulations to collect and use the achieved results to pick the best size for each and every transistor. The computational cost of an exhaustive search approach based on electrical simulations is most often prohibitive for networks larger then a few transistors. However, it is also possible to search for the optimal solution using some set of heuristics.

This work describes TBSizer, a new tool to size transistor networks. It extensively employs electrical simulations of the transistor network with different size parameter combinations, to precisely evaluate the resulting network characteristics using adequate, adaptable cost functions. TBSizer aims mostly at being a flexible tool to fulfill the needs of several sizing requirements. In this sensem TBSizer has as main characteristics that it: (i) allows changing its target technology easily; (ii) delegates the sizing evaluation to user-defined metrics, enabling the use of simple as well as complex cost functions. These functions drive the sizing process by the transistor network designer specific needs and expectations; (iii) enables the definition of multiple sizing parameters search spaces, i.e. makes it possible to individually size different transistor groups or even single transistors inside an otherwise fully sized network; (iv) takes user-provided arc stimuli, allowing the designer to optimize application-dependent critical paths.

The aforementioned features produce a tool with flexibility above what can be found available today. Notably, TBSizer enables that designers control key aspects of the sizing process.

In its current version, TBSizer employs one of two approaches for sizing transistors. The first is the exhaustive approach. Given a set of constraints that defines the solution search space, selected by the software user, the TBSizer exhaustive approach evaluates every possible sizing configuration of the search space and returns the exact best solution, This approach is only feasible for a small set of parameters and for a limited range of possible values. The second is a Genetic Algorithm (GA) approach. The GA approach is suitable for dealing with large instances of the sizing problem, where the computational effort required by an exhaustive method is not feasible.

1.1 Justification for Proposing TBSizer

Although abundant literature on transistor sizing exists, there is no established tradition on proposing the automation of this design step by commercial Electronic Design Automation (EDA) vendors like Cadence or Synopsys. Even the availability of open-source tools for sizing is quite limited.

Designers often rely on the availability of e.g. commercial standard cell libraries by vendors like ARM, Synopsys, etc. and employ static timing analysis (STA) tools to select adequate cell driving strengths to achieve the design goals. While this is mainstream and works, there is still room for improvement in selected fields. For example, there is little support for designing asynchronous quasi-delay insensitive (QDI) circuits based on standard cells and using conventional libraries for this kind of design can lead to sub-optimal circuits. QDI cells can be quite large (several dozens of transistors). Developing cell libraries for QDI design is accordingly a task that can benefit a lot from the availability of a powerful sizing tool. Also, several designs might benefit from enhancing a standard cell library with specific design-oriented cells in critical parts of a module, for example by creating a large macrocell that implements some functionality in a specific transistor network. Adequate sizing of such a (large) transistor network can also benefit from a tool that systematically explores the network design space, looking for the best point in this space according to a user-selected set of optimization criteria.

The above discussion justifies the proposal of a flexible tool to perform the sizing of a transistor network describing a digital, combinational or sequential, functionality. This work proposes TBSizer, a tool fulfilling this proposal.

The rest of this document is divided into six Chapters. Chapter 2 presents some of the basic concepts covered in this work, from CMOS technologies to the characteristics of algorithms employed herein. Next, Chapter 3 contains a brief review of the literature on sizing, in breadth (sizing applied to domains different from that addressed in this work) and depth (multiple algorithmic approaches to sizing CMOS transistor networks, targeting varied cost functions). Follows Chapter 4, which depicts the TBSizer software organization, showing details of the existing software entities, in addition to discussing their behavior. Chapter 5 addresses the preparation of TBSizer input, detailing the preparation of the transistor network, the tool configurations and the elaboration of cost functions. Chapter 6 presents the results of three experiments using TBSizer: the first compares TBSizer with ROGen/Ces a previous approach to sizing transistor networks; the second presents trade-offs in the process of executing TBSizer based on the partitioning of the input transistor network in sizing classes; the last experiment shows TBSizer used to resize cells from a commercial standard cell library, comparing the original and resized cells. Chapter 7 includes a set of conclusions and directions for further work on the subject of transistor sizing.

2. BACKGROUND

This Chapter covers in five sections some basic material required to introduce the work. Section 2.1 gives a brief description and discussion of CMOS technologies, starting with the definition of pMOS and nMOS transistors and explaining structure and operation of some of the main current device types available in CMOS technologies: bulk, SOI and FinFET transistors. Section 2.2 addresses some logic circuit families, from the way transistors can be combined to form static and/or dynamic logic, to the differentiation between combinational and sequential circuits. Section 2.3 provides some characteristics of the Spice format, the fundamental transistor-level description language selected for use in this work. In Section 2.4, a typical standard cell development flow is explored, the ASCEnD-A Flow, proposed in the scope of the research group where this work is conducted. Standard cells are considered as one of the main targets where to apply the sizing techniques proposed in this work, although the software is in principle be applicable to any transistor network implementing any digitally specified behavior. Finally, Section 2.5 explores the basics of two space search techniques this work will employ, the exhaustive and the genetic-based techniques.

2.1 CMOS Technology

Most industrially employed semiconductors are formed from Silicon (Si) crystals processing. Silicon is classified as an element of Group IV in the Periodic Table of Elements. As a consequence, each atom of Si has a natural capacity to form four covalent bonds with neighboring atoms, as depicted in Figure 2.1[1]. Pure Si is not a good electrical conductor. However, by introducing selected impurities in Si crystals it is possible to improve its conductivity, in a process called *doping*. A Group V dopant, like Arsenic (As), has 5 valence electrons. Doping Silicon with this element, one electron remains free of connection, a free electron. If there is a free electron in a doped Si crystal, the material conduction becomes higher, and it is then called an n-type semiconductor, because free electrons are negative charges. Similarly, there are Group III dopants, such as Boron (B), which has 3 electrons in the valence layer. The use of this element as doping for Si crystals generates the lack of an electron, called *hole*, which can float along neighbor atoms. A hole acts as a positive conductor, and a Si crystal doped with a material that provides holes produces a p-type semiconductor.

[1]It should be noted that Si lattices are in fact tri-dimensional. The planar representations here are mere simplifications.

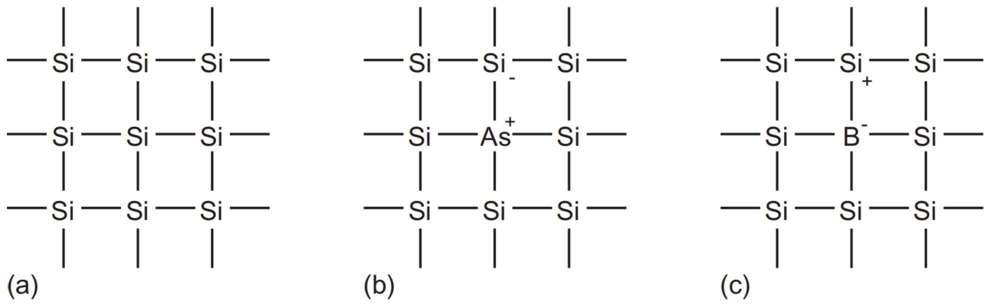


Figure 2.1: Pure Si lattice (a) and lattices with dopant atoms to form (b) n-type and (c) p-type semiconductors [WH11].

A Metal-Oxide-Semiconductor (MOS) structure is created with strict conduction and insulation rules. These rules involve specific chemical processes, such as oxidation of silicon, selective introduction of dopants, connections between wires, and contacts. A CMOS technology assumes the existence and use of two transistor types, the n-type (nMOS) and the p-type (pMOS) transistors. Operation of these devices relies on electric field effects. Each of these devices is accordingly called a Metal Oxide Semiconductor Field Effect Transistor (MOSFET). The following sections discuss some of the most used transistor types today.

2.1.1 The Bulk Transistor Type

A bulk transistor, illustrated in Figure 2.2 is composed by a conductive Gate built on top of an insulator that separates the Gate from a region of a Si crystal adequately doped, called the transistor *substrate* or *bulk*. To each side of this sandwiched MOS structure are regions doped with the reverse kind of dopants present in the bulk, respectively called the Source and the Drain of the transistor. The region below the Gate and between the Source and the Drain is called the transistor *channel*. The structure thus formed comprises a four-terminal electronic device. Often, the bulk region is assumed to be connected to a fixed voltage supply, while the other terminals can see their potential change along the device operation. Accordingly, at higher levels of abstraction, transistors are abstracted as three-terminal electronic devices. An nMOS transistor is built with n-type Source and Drain regions, while a p-type transistor is built with p-type Source and Drain regions. An important point is the behavior of transistors: nMOS transistors connect its Drain and Source when the Gate is at a positive potential (logically, G=1) allows current to flow between the Drain and the Source and disconnects Drain and Source otherwise. The pMOS transistor has a complementary behavior, a negative (or null) potential at the gate (logically, G=0) allows current to flow between the Drain and the Source.

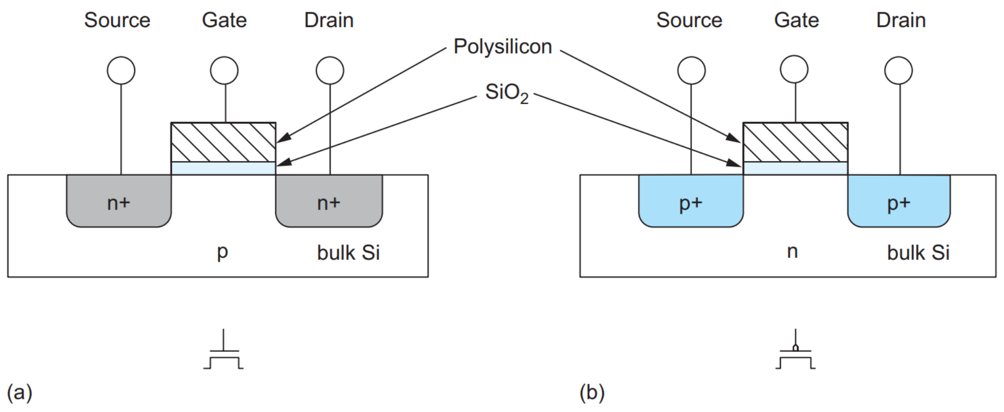


Figure 2.2: CMOS Bulk Transistors [WH11].

2.1.2 The Silicon on Insulator (SOI) Transistor Type

A variant of CMOS transistor are Silicon on Insulator (SOI) structures. Figure 2.3 depicts an example of a SOI transistor structure. This example shows transistor structures built on top of an insulator grown over a crystal of silicon, in a fully depleted silicon on insulator (FDSOI) technology. Besides using silicon crystals, it is also possible employ sapphire as insulator material, among others. One of the major advantages of using an insulator substrate is the elimination of capacitances in the Source and Drain regions and the substrate, leading to higher speed and less leaky transistors.

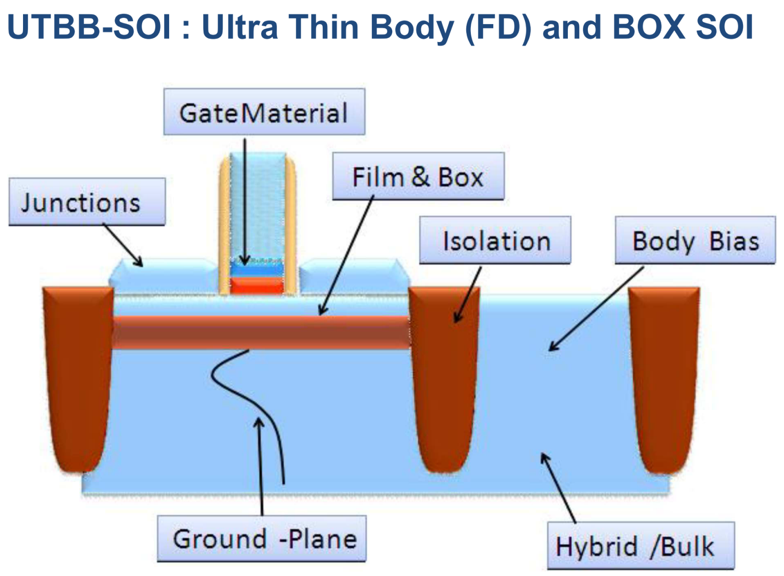


Figure 2.3: Example of a CMOS SOI transistor in an FDSOI technology [MFC13].

2.1.3 FinFET devices

The structure of FinFet transistors, schematically depicted in Figure 2.4, is based on the fact that the gate structure tri-dimensionally involves the transistor channel on three sides (top, left and right), which provides enhanced control of the flow of charge carriers along the channel [Ali10]. This structure is called a *fin*. FinFETs are accordingly also called tri-Gate transistors. Usually, a FinFET is constituted by several fins in parallel, sharing the diffusion area, which makes their sizing a discrete problem [PBMR14, ZMPR15].

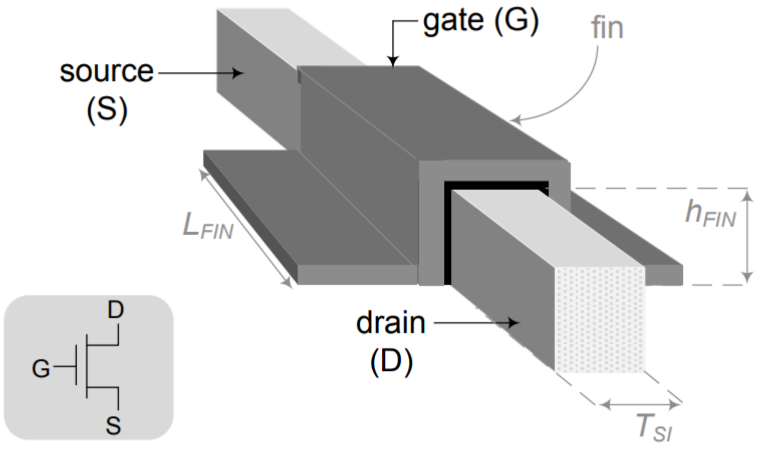


Figure 2.4: Three-terminal FinFET geometrical structure [Ali10].

2.2 Logic Circuit Families

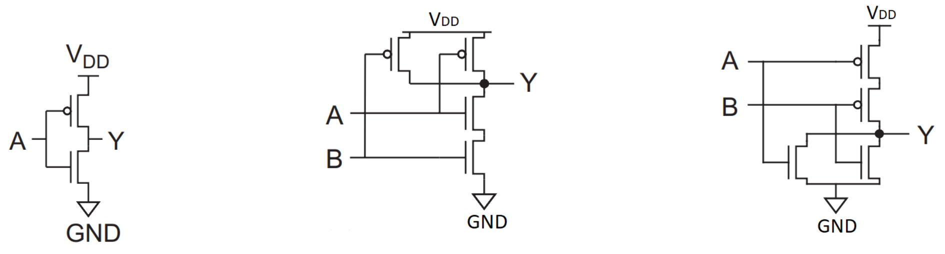
This Section explores some types of logic circuit topologies useful in general and specifically in the context of this document. This Section also explores some techniques for implementing circuits through the use of structures like pass transistors and transmission gates. This work only contemplates static logic design, as defined in Section 2.2.2. Dynamic logic topologies are discussed briefly, but the software proposed herein cannot deal with such circuits at all. Addressing the sizing problem in dynamic logic transistor networks is seen as a possible and interesting future work.

2.2.1 Totem Pole Logic Gate Implementations, Pass Transistors and Transmission Gates

The *strength* of a signal (or its capacity to drive loads at the output of a circuit or device) is directly related to its proximity and path to the power supply (here called VDD) and ground (called GND) nodes. These nodes correspond, respectively to the logic 1 and 0 *strong* values. A pMOS transistor is known to be efficient in passing a value 1 between its Drain and Source, and not that efficient in passing a 0 value between these terminals. An nMOS transistor has the opposite behavior, it passes the value 0 efficiently but not the value 1.

Given the transistor behavior just described, one electrically efficient form of building single output logic circuits is to employ the so-called *totem pole* or *stacked* transistor organization. In a totem pole, a pMOS transistor network with a single output connects to the VDD node and is controlled, by its transistor gates, if it passes or not a strong 1 to its output. Completing the totem pole there is an nMOS transistor network that connects to the GND node and is controlled, by its transistor gates, if it passes or not a strong 0 to its output. The single outputs of both pMOS and nMOS transistor networks are connected together and correspond to the transistor network (global) output.

Figure 2.5 shows three simple examples of totem pole transistor organizations that correspond respectively to CMOS implementations of well-known logic gates: an inverter, a 2-input NAND and a 2-input NOR.



(a) An inverter logic gate. (b) A 2-input NAND logic gate. (c) A 2-input NOR logic gate.

Figure 2.5: Example of simple totem pole transistor networks implementing some common logic gates [WH11].

It is worth noting that: (1) there is no direct electrical connection between the totem pole transistor network inputs and the gate output, all inputs influence the output value generation through electric field effects only, in practice insulating the influence of inputs on the outputs; (2) the pMOS and nMOS transistor networks have dual topologies, in the sense that two transistors connected in series and controlled by some two inputs at their Gates in the pMOS network correspond to a pair of transistors connected in parallel in the nMOS transistor network, controlled by the same inputs at the respective Gates; (3) any transistor network processing n inputs requires $2n$ transistors in a totem pole organization; (4) Even if transistors are passing 1s and 0s efficiently, attenuation does occur when carriers pass from a Drain to a Source; this imposes a limit on the number of stacked transistors allowed to be traversed from VDD or GND to a totem pole gate output. CMOS network transistor designers often agree that stacks with more than four transistors in series between the output and VDD/GND nodes are not an option.

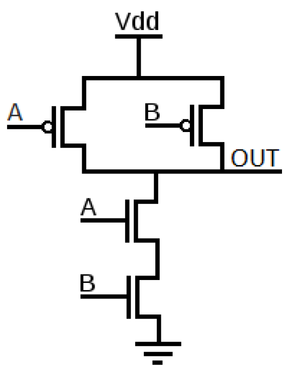
The reader can note from the above discussion that totem pole transistor networks can be expensive in terms of area and/or leakage current. Alternatives to totem pole transistor organizations can rely on the passing of logic values from inputs directly to the outputs under control of some chosen inputs. This can save area, at the cost of electrical efficiency. A *pass transistor* is a pMOS or nMOS transistor used to pass an input value from its Source to its Drain (or vice versa) under control of some other input attached to its Gate. This allows reducing (in some cases, drastically) the amount of required transistors to implement a logic function. Due to the electrical characteristics of single transistors, sometimes two transistors, one pMOS and one nMOS, with Drains and Sources respectively connected and inverted control signals attached to the gates is employed, in what is called a *transmission gate*. A transmission gate can pass 1 and 0 values efficiently from its input (a Drain-Source pair) to its output (the other Drain-Source pair). This is electrically better than using just an nMOS or a pMOS pass transistor, but adds costs. The associated additional costs of transmission gates are area and the need to make available the control values in direct and complemented form.

The software developed is capable to dimension width and length of transistors in totem pole organizations and pass transistor/transmission gate organizations as well.

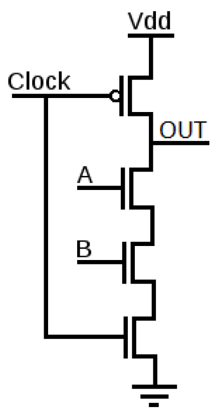
2.2.2 Static and Dynamic Logic Circuits

The development of cells and gates is usually accomplished with static logic such as the one described using totem pole transistor network organizations. Static logic comprises a pull-up network (the pMOS transistor network in a totem pole organization) and a pull-down transistor network (the nMOS transistor network in a totem pole organization), refer to Figure 2.6a.

An alternative gate development method is based on the use of dynamic logic organizations, refer to Figure 2.6b. The principle underlying dynamic transistor networks is to use a two-phase operation, where the network output is first pre-charged to a defined logic value (say, logic 1), followed by an evaluation phase where inputs are allowed to logically confirm or change the pre-charged output value in the output, according to the network expected behavior.



(a) A static NAND gate.



(b) A dynamic, clocked NAND gate.

Figure 2.6: Static (totem pole) and dynamic CMOS NAND gate implementations.

Figure 2.6b shows a 2-input NAND gate controlled by three inputs. The logic inputs (A and B) define the gate behavior, just as in the static implementation of Figure 2.6a. However, the gate output in the dynamic NAND can only be used in the evaluation phase, when the control Clock signal is at logic 1. The dynamic gate operation is as follows: (1) Clock is set to logic 0, defining the pre-charge phase, where the output node OUT goes to 1 through the only pMOS transistor in the network; (2) Clock is then set to logic 1, and the gate enters the evaluation phase, where the input values A and B determine if OUT is going to change from 1 to 0 or not, as defined by the 2-input NAND truth table.

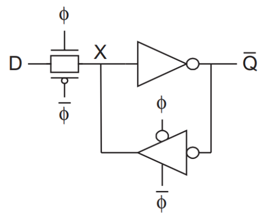
Circuits designed using dynamic logic styles can be considerably faster and more compact than their equivalent static implementation [DM04]. This design method is widely used in high performance circuits. However, dynamic gates have less noise resilience than static CMOS gates. The switching threshold voltage of a dynamic logic gate is usually the transistor threshold voltage. The switching voltage in static CMOS logic gates is about half the power supply voltage. For this reason, several research works [DM04, WS99] address dynamic logic noise issues.

2.2.3 Combinational and Sequential Circuits

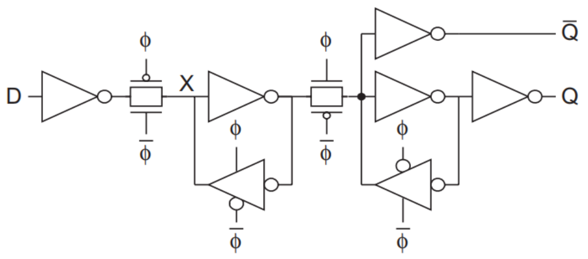
Digital logic consists of two strands, those of combinational logic and sequential logic. Combinational logic is one where the value of outputs is determined exclusively by the instantaneous value of the circuit inputs alone. Conventional logic gates such as inverters (NOT gates), NANDs and NORs are examples of combinational circuits, refer to Figure 2.5. Such circuits are called *combinational* because their output values result merely from a combination of its input values.

Sequential logic, by opposition to combinational logic, is defined as any logic where output values not only depend on the instantaneous values of its inputs, but also consider the sequence of values presented at the inputs along time. Often, the consideration of past inputs in sequential logic is captured by modeling the circuit's internal *state*. Some classical examples of sequential circuits are finite state machines, counters, etc. Sequential circuits require internal storage capability to *remember* part of its *history*. Storage of information inside logic circuits can be achieved using the feedback of outputs (of transistors, gates etc.) to the part of the circuit that generates new outputs.

The synchronous design paradigm is a simple and efficient way to organize sequential transistor networks to process information. A synchronous circuit relies on one or more special control signals, often called *clocks*, which are used to globally control the flow of information in the circuit. To support synchronous design it is usual to rely on the use of special transistor networks to store information. Examples of such networks are the latch and flip-flop devices, which are illustrated in Figure 2.7.



(a) A level-1 sensitive latch built with transmission gates.



(b) An edge-triggered master-slave CMOS flip-flop built with transmission gates.

Figure 2.7: Two examples of transistor networks to store one bit of information [WH11].

2.3 The Spice Format

The SPICE (Simulation Program with Integrated Circuit Emphasis) format and associated simulation tools were developed to solve the nonlinear differential equations that describe components such as transistors, resistors, capacitors and voltage sources. Spice is both a program to electrically simulate circuits, and a language that enables to electrically describe circuits and also generating stimulus and control the generation of simulation output results. The Spice language commands offer several ways to stimulate a circuit. A DC source or even a capacitive charge can easily be implemented in association with transistor networks. It is also possible to implement measurement techniques directly in the language, specifying initial conditions for the circuits and the input behavior, guide the simulation execution and retrieve the computed results.

In this work, Spice simulations have the fundamental function of generating the circuit simulation results. The software prepares the SPICE files and calls the Cadence Spectre program (a flavor of the Spice program) to perform circuit simulations.

2.4 A Standard Cell Library Design Flow and Cell Transistor Networks

The sizing of transistor networks is just one of several steps during the design of a complex integrated circuit (IC). The algorithms proposed in this work take on the task of dimensioning transistors of a network, a required step in the development of custom hardware.

The design of fully custom hardware for a complex IC is a very demanding and expensive process, allowed only to few companies and groups worldwide. A much more affordable way to design ICs is to employ what is known as a *cell-based design* technique. Cell-based design relies on the existence of a standard set of pre-characterized transistor networks (called *cells*) that can be logically interconnected to implement a given, maybe very complex, circuit behavior, such as a microprocessor or a USB controller. Designing standard cell libraries can be challenging, but once done, the development costs of the library can be amortized over dozens or even thousands of distinct designs.

One of the goals of the work proposed here is to support the design of transistor networks constituting cells of a standard cell library. Thus, it makes sense exploring the process of developing such a library and investigating the role of the transistor sizing step in such a flow. This is the objective of the next Section.

2.4.1 The ASCEnD-A Flow

An asynchronous standard cell library design flow was proposed and developed by the GAPH research group. This method is named the ASCEnD-A flow [Mor16]. The process of developing a standard cell library is divided in the ASCEnD-A flow into four macro steps: (1) Cell Library Template definition; (2) Cell Sizing; (3) Cell Layout generation; (4) Cell Characterization. Refer to Figure 2.8 for an illustration of the overall structure of the ASCEnD-A flow. Each of the macro steps is now briefly described.

1. The Cell Library Templates is the process responsible for generating the main models and parameters to use during the library development process. Libraries often have standardized features. For example, the height of all cells in a library is often the same (to facilitate place and routing of cells). The position and size of VDD and GND wires in a cell is pre-determined (to allow forming the IC VDD and GND distributions seamlessly). Most cell libraries define a certain number of *tracks*, pre-defined regions where wires can pass over the cell without interfering with the cell behavior. These and other parameters valid across the whole set of cells, and valid in the whole library define a Cell Library Template.

2. The Cell Sizing process is responsible for finding effective values for the W and possibly L to each cell transistor. It is this process which is addressed by the work proposed herein. It typically starts with a un-dimensioned or partially dimensioned transistor schematic described in Spice format that implements the cell functional behavior, added with target performance, power and other precise limits of operation for the cell at hand. The output of the Cell Sizing is the same transistor schematic where every transistor in the network is annotated with the specific sizes.

3. The Cell Layout is the process of physically generating the cell from the sized transistor schematic. The physical cell layout is produced based on some physical synthesis tools, following the technology settings. Cell layouts must of course respect the technology design rules and cell layouts must follow some standard description rules such as those contained in GDSII or CIF specifications. Also, abstract descriptions result as a sub-product of the Cell Layout generation process. An example is the LEF abstract cell description, used by physical synthesis tools in producing the IC layout. LEF descriptions inform just those cell features required to use the cell to build circuits with it, including the position and dimensions of I/O cell pins, possible locations for higher metal layers in the cell that may impose routing restrictions of wires passing over the cell, and so on.

4. The Cell Characterization process consists in the last stage of the cell production. Given a cell layout, electrical extraction obtains a detailed timing model for it, based on which an extensive set of electrical characteristics are defined. Some data are static, such as the cell input capacitances and the expected leakage current and power the cell drains. Other characteristics are dynamic, such as the switching delay of the cell, the rising and falling times for outputs and the dynamic power consumption. These are either specified as a function enabling to compute dynamic values, given the environment where the cell is operating, or as a set of tables from which the exact cell behavior can be interpolated or extrapolated by circuit synthesis tools.

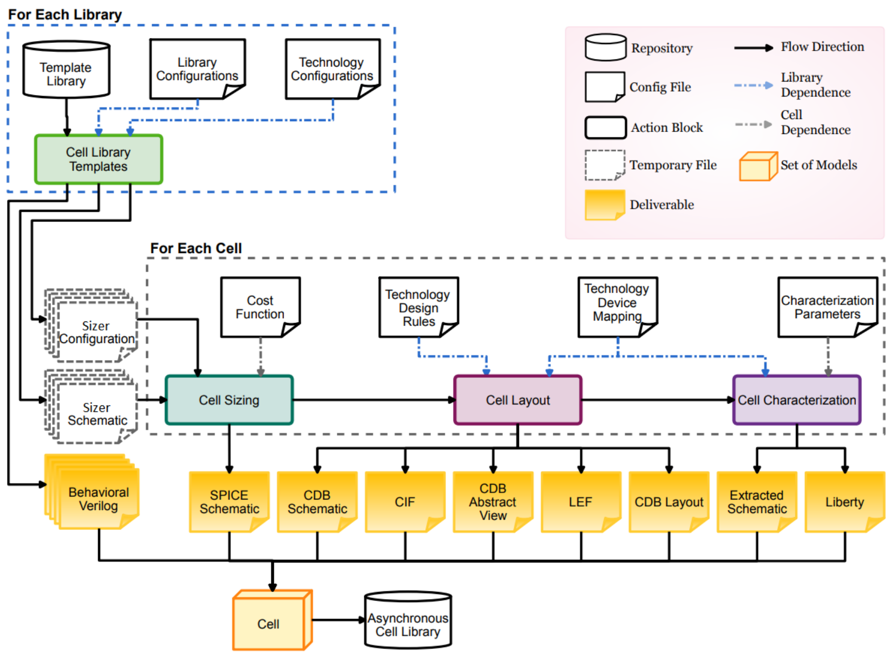


Figure 2.8: Overview of the ASCEnD-A flow [Mor16].

2.5 State Space Search Methods

The work developed focuses on dimensioning transistors in a transistor network. This process implies a very large solution state space, usually requiring great computational effort for all but very small transistor networks to find the absolute best solution to the sizing problem. To enable a flexible exploration of such large state space, the work herein proposes the use of two space search methods, an exhaustive search method, described in Section 2.5.1 and a heuristic search method, based on genetic algorithms, described in Section 2.5.2. The overall goal is to find the optimal combination of transistor sizings for a transistor network within an acceptable execution time.

2.5.1 Exhaustive Search Method

Exhaustive search has the characteristic of analyzing all possible combinations of sizing for every transistor to determine the absolute best combination of values optimizing a given cost function [AB09]. Small problems can be solved with such strategy and find the absolute optimum result is guaranteed. Depending on the size of the problem to solve, the time to finish the execution can be unfeasible. The problem of sizing transistors is classified as NP-Complete [AB09]. Assume that the number of possible combinations of sizes, is say m, and the number of transistors that undergo this variation is n. Having these two values, it can be seen that the order of magnitude of growth for an exhaustive sizing search algorithm is $\mathcal{O}(m^n)$.

2.5.2 Genetic Search Method

Genetic algorithms (GA) are probabilistic search algorithms designed to work with large state spaces [GH88]. These algorithms are parallel, using a set of individuals (a population of solutions) to generate a new set of solutions (offsprings). When processing a population, the genetic algorithm creates descendants. Then, it directs future populations to explore new solutions and this process characterizes the solution space. GAs rely on Darwin's theory on the origin of species [Sum92]. As Darwin pointed out, natural selection tends to make individuals well adapted to the environment, approaching perfect adaptation, or at least improving adaptation over other competing species in a same region.

A GA instance is implemented in five main steps Figure 2.9: (1) population initialization; (2) individuals selection; (3) crossover application; (4) mutation application; (5) evaluation. Figure 2.9 describes an example of how a GA implementation operates, by means of a sequential pseudo-code [Neu08].

```
$t := 0$;
initialise population $P(0)$;
**while** end of adaptation $\neq$ true **do**
    select $\alpha$-individual $b(t)$ as first parent;
    **for** $k = 0$ to $r - 1$ **do**
        select second parent $c(t)$ randomly;
        apply crossover $\chi_\Omega$ and mutation $\mu_\Omega$
            $a_k(t+1) := \mu_\Omega\left(\chi_\Omega\left(b(t), c(t)\right)\right)$;
        evaluate fitness $f\left(a_k(t+1)\right)$;
    **end**
    $t := t + 1$;
**end**
```

Figure 2.9: General structure of a GA with alpha-selection [Neu08].

1. The population initialization step consists in randomizing parameters of a defined set of individuals. Each individual receives a starting arbitrary value for each of its parameters that is within the limits of the problem to solve.

2. The step of selecting individuals consists of finding the best adapted individuals and storing their characteristics. Later, these are used to contemplate the crossover step.

3. The crossover stage consists in reproducing the characteristics of the previously selected individuals. Reproduction can be carried out with several variants, combining individuals parameters, such as the average of values, dominance by one parent features etc.

4. The mutation process consists in changing the values of the offspring parameters randomly, by wide variations. This process must be random, both for the selection of genes and for the variation of that gene.

5. Finally, the evaluation process consists in numerically determining how good an individual is. In this process, several cost functions can be used, making it easy to adapt the GA to the problem to solve.

One important feature of genetic algorithms is the unrolling of parallelism in its implementation. As mentioned in [Whi94], three types of parallelism area available for GAs.

1. The first type is global populations with parallelism. This model of parallelism creates candidates turns, so that in parallel, they fill the current population. Thus, it is possible to make the crossing and evaluation steps in parallel with various aspects. Having a population with several candidates, it is possible to parallelize several aspects, to more quickly fill the set of new generation offsprings.

2. The second model of parallelism is called island models. Where there are several processors, it is possible to separate the problem on several islands and let these run as if they were canonical genetic algorithms. This methodology brings genetic diversity to each island, each population shares genetic material with other islands in a different way. If too many connections occur between islands, miscegenation occurs and local differences are lost.

3. The third method of parallelism is the cellular genetic algorithms. This methodology can be explained as a matrix, where each candidate is in a position. From this, each candidate reproduces with the closest neighbors and as soon as an offspring manages to transcend over its predecessor, it assumes that position in the matrix. This model generates island behavior similar to the second method. Parallelism takes since each place in the matrix can be assigned to a processor.

Due to the characteristic of the transistor sizing problem, the methodology that best fits the objectives of this work is the first. This is because in the problem the longest delay during execution is expected to be the candidate cost function evaluation. Accordingly, this work implement a variation of the first parallelism method, in which the parallelism focuses mainly on system calls and tries to keep a multicore CPU fully loaded.

3. RELATED WORKS

This Chapter reviews a set of works published in the subject of transistor sizing and similar subjects. The objective is to provide an approach of the literature, covering part of the diversity of methods and applications targeted by sizing techniques. The review is in no way encompassing, and it is intended as historical and modern perspectives of how the transistor sizing problem has been treated. The Chapter starts with a Section on historical and modern perspectives of transistor and gating sizing works, loosely based on a recent paper by Singh and others [SJM$^+$18], which contains an encompassing state of the art review. Next, this perspective is enhanced with additional, more recent works, or relevant works not addressed by the Singh et al. article. The last Section of this Chapter brings a qualitative comparison of a relevant subset of the mentioned works.

3.1 Historical and Modern Perspectives on the Transistor Sizing Problem

One of the first, if not "the" first article to describe an approach to sizing transistors for CMOS technologies is that by Kang [Kan81]. This work sets as main objective to define the channel width of p- and n-type transistors of a standard cell transistor network in a 3.5μm technology[1]. The objective function is to minimize the area-delay product of the cell, ignoring power figures. The proposed analytical model is applied by hand and verification takes place using Spice simulations. All p-transistors are equally sized in the network, and the same is true for n-transistors, to simplify the problem (see Section 3.2 for more on this kind of technique). Besides area and delay, attention is given in this method to the cell noise margins, set as 25% of the supply voltage.

In the other extreme of time, the recent work by Singh et al. [SJM$^+$18] addresses power-delay and power-delay-area products (PDP and PDAP) as objective functions, on a 130nm CMOS technology. Authors employ an enhanced version of the logical effort (LE) method proposed by Sutherland et al. [SSSH99] in 1999. LE enhancement is based on the use of two heuristic algorithms: Interior Search (ISA) and Gravitational Search Algorithms (GSA). Results point to average enhancements of 35.1% and 63.8% respectively, for the PDP and PDAP figures.

Singh et al. [SJM$^+$18] provide an encompassing review of the literature in transistor sizing techniques, covering around 40 years from 1977 to 2018. Interested readers should address this reference for more details in the current state of the literature on transistor sizing methods. It is apparent in their review that analytical methods evolved slowly until 1999, when the LE method appeared, rapidly becoming the most used sizing method today. However, the pure LE method only optimizes transistor network delay, ignoring relevant figures like area, power and input slope variations, not to mention PVT variations and operation in near-threshold and subthreshold regimes. Also clear in this review is that the use of GAs stands out as the most used heuristic algorithm to perform sizing, either alone or in combination with other heuristics such as particle swarm optimiza-

[1]The Author calls standard cell *polycells*.

tion (PSO) and others. Probably the best way to perform transistor sizing today consists in using combinations of analytical, simulation and heuristic techniques. The work proposed here advocates using either an exhaustive technique and/or GA-based heuristics, both in combination with extensive electrical simulations executed in a highly parallel execution environment.

3.2 Geometric Programming for Transistor Sizing and Gate Sizing

Posser et al. [PFWR12] proposed and compared techniques to perform gate and transistor sizing for digital CMOS circuits. According to these authors, gate sizing is a restricted version of the sizing problem, where all transistors of the same type (pMOS or nMOS) in a network are assumed to have the same size. The article describes the use of the geometric programming technique, adopting a switch-level RC gate model and computing delays using the Elmore delay model. Analysis are conducted with Spice simulations. Comparisons are undertaken employing some of the ISCAS‘85 benchmarks and other in-house circuits. They show an average reduction of 21% in delay with the gate sizing technique, for iso-area and iso-energy values, when compared to a conventional 45nm predictive technology library. When applying transistor sizing, authors report an average of 52.5% reduction in delay compared to the same 45nm library. Authors mention the developed tool can minimize delay subject to area or minimize area subject to delay, but reported results are limited to the first option only.

The software proposed in this work will also support sizing using both transistor and gate approaches. In addition, the proposed software will be much more flexible. For example it will be able to deal with W sizing only, L sizing only or sizing of both, W and L. Also, the cost function scheme proposed herein is capable to deal with essentially any combination of cost parameters. The work of Posser et al. uses a methodology directed to minimize the network critical path. The system proposed here is fully agnostic, being able to employ a critical path approach, or any other function a designer can deem useful.

3.3 Simulated Annealing and Artificial Bee Colony Algorithm to Size Digital Cells

Gupta et al. [GMGA19] propose a method for sizing digital cells. This work employs and compares two algorithms, Simulated Annealing (SA) and Artificial Bee Colony (ABC). The objective was to attenuate the leakage current, keeping invariant parameters such as propagation times and the area of the circuit. The comparison of the results was made with cells of an existing digital CMOS library, achieving around 70% savings with their approach. A negative point found was the execution time of the algorithms, since using the SA algorithm took approximately 0.5 hours and the ABC algorithm took 1.5 hours.

The work proposed here aims to try to solve sizing problems much faster. A designer who develops circuits often needs results ready in minutes. Taking hours just to size a circuit can be impractical. In addition, cost functions need to be very flexible, enabling a large number of possible parameter combinations.

3.4 Sizing Analog Amplifiers using a Genetic Algorithm

In work [EBBZ18] authors proposed a transistor sizing technique targeting the sizing of transistors in an analog amplifier. The method used to design three-stage bipolar transistors was a GA implementation. The focus of the work was to improve various aspects of amplifiers, such as generated gain, input impedance, output impedance and cutoff frequency. Authors were able to show through SPICE simulations that a GA is capable of delivering the expected performance. The software proposed herein does not address analog circuits sizing, only sizing for digital CMOS circuits. The work of El Beqal et al. demonstrates how flexible GAs can be.

3.5 A Method for Standard Cell Optimization for the Subthreshold Regime

Developments presented by Grimminger et al. in [GFWK12] aim at optimizing transistor networks for operation in subthreshold regimes. It targets the symmetry between propagation times and the reduction of dynamic energy consumption. A ring oscillator structure is used to exhaustively determine the required gate delays. Authors manage to optimize cells for low energy consumption and reduce their propagation delay. The work under development here is expected to be useful to address transistor networks working in either subthreshold or superthreshold regimes. Unlike [GFWK12], an approach focused on simulation arcs is sought. Each switching possibility of the cell becomes an arc and arc simulations produce data to determine the network critical path more easily.

3.6 Transistor Sizing in Latch Comparator

The work developed by Yaqubi and Zahiri [YZ17] brings two approaches to the design of latch comparators, a relevant module of analog circuits such as analog to digital converters (ADCs). To perform a latch comparator sizing, three parameters are used to validate whether or not the system meets the needs, power consumption, delay, and total size figures of merit. As this is an approach where the objectives are multiple, a multi-objective particle swarm fuzzy algorithm, and a particle cluster swarm algorithm are employed. The run time for the methods were respectively 4.67 hours and 7.5 hours. In addition, there were no differences in quality between the results achieved by both approaches, only their execution times were distinct. Here, a software capable of processing digital cells of different types is the target. to deal with combinational and sequential transistor networks.

3.7 Comparison Between Revised Works

This Section briefly summarizes the revised works, comparing them according to four criteria: (1) type of circuits addressed in the approach; (2) employed sizing method; (3) target sizing objective; (4) processed experimental data. Table 3.1 collects the criteria information for each of the revised works. Singh et al. [SJM$^+$18] cite several more works. For example they analyze six other works that employ GAs, three that propose enhancements to the LE method and two papers addressing exact formulation of the transistor and gate sizing problems.

Table 3.1: A comparison of transistor/gate network sizing approaches [Author].

Work	Circuit Type	Sizing Method	Sizing Objective	Evaluation of Results
[SSSH99]	CMOS digital	LE	Delay propagation	Not Available
[SJM$^+$18]	CMOS digital	LE enhanced with ISA and GSA heuristics	PDP and/or PDAP	Boolean gates, Muxes, FFs
[PFWR12]	CMOS digital	Geometric Programming (+RC delay model)	Delay propagation	ISCAS'85 benchmark
[GMGA19]	CMOS digital	Simulated Annealing (SA) and Artificial Bee Colony (ABC)	Leakage power	Available CMOS library
[EBBZ18]	Analog amplifiers	GA	Voltage gain, Input impedance, Output impedance and Cutoff frequency	Industry case study
[GFWK12]	CMOS digital	Ring oscillator	Propagation delay symmetry, dynamic power	Industry case study
[YZ17]	Analog latch comparator	Multi-objective particle swarm intelligence and fuzzy systems	Power consumption, Delay and Total transistor width	Not available
This Work	CMOS digital	Exhaustive and/or GA enhanced with extensive electrical simulations	Anything (User Defined)	Asynchronous and Synchronous Gates

4. THE TBSIZER SOFTWARE

This work describes TBSizer, a software for sizing transistors on digital circuits. TBSizer applicability is limited to traditional bulk CMOS or Silicon on Insulator (SoI). The tool does not support sizing transistors of analog circuits either. In addition, FinFET transistor networks are not addressed as a target in this first version of the software. The conductivity of the channel in FinFET gates relies on using a set of parallel fins. Thus sizing is usually restricted to choose a discrete number of fins to use, not to change fin dimensions (see Section 2.1.3).

The main goal of TBSizer is: given a CMOS network of transistors that describes a static circuit behavior, identify the best combination of width (W) and eventually length (L) for each of the transistors in the network. Here *best* implies a precise definition of optimality, as parameterized by the software user using resources provided by the tool. A user-defined cost function drives the software to search for the best of all W and L allowed combinations, within a chosen range of sizes and given a valid increment step for each dimension, in accordance to the specific technology rules at hand.

The sizing of CMOS transistors in digital networks can affect many of the network characteristics. Measurable and controllable impacts include energy per operation, propagation time, circuit area, and combinations of these factors, among others. TBSizer enhances the possibility to address transistor networks employed in the design of asynchronous circuits. Standard cells supporting the design of asynchronous circuits often contain feedback loops (to enable sequential behavior), implying the existence of internal arcs. Examples of such cells are C-elements, NCL gates, arbiters and other cells with explicitly sequential behavior. Static sizing techniques, such as *logical effort* proposed by Sutherland et al. [SSSH99], may not always achieve the best dimensions for such cells. Other algorithms that do not analyze the internal states of the circuit may not provide the best solution for this target.

Figure 4.1 depicts an overview of the TBSizer internal flow. It takes a transistor network described in spice, a list of parameters to search and a configuration file. TBSizer has two approaches to generate possible sizing candidates: (i) the exhaustive approach, which yields the exact best possible result at the expanse of simulation time; and (ii) the genetic approach, which uses heuristics to reduce the simulation time at the expense of result quality. It employs consecutive electrical simulations to evaluate the impact of different transistor size combinations. The *Spice Handler* creates spice decks for stimulating each sizing candidate, invokes Cadence Spectre for simulation, and collects the results. The tool relies on user-defined a cost function in order to rank candidates and select the best result. A configuration file selects between the exhaustive or the genetic approach, and parameterizes the execution accordingly.

TBSizer enables sizing each any group of transistors or even individual transistors, thus allowing to explore the design space for unconventional solutions. The user-provided cost function enables defining the characteristics to optimize to best comply with design requirements. Further-

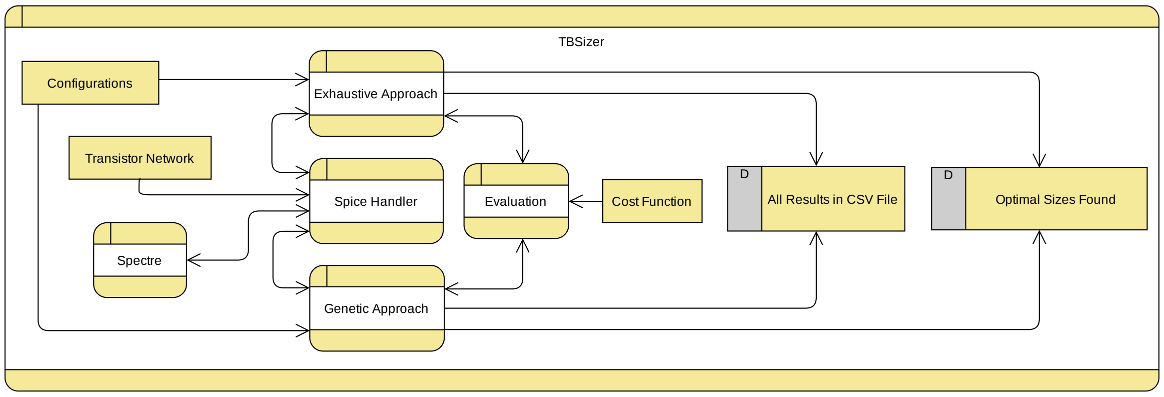


Figure 4.1: The TBSizer Flow [Author].

more, it is also possible to prioritize the influence of some arcs over others when computing the cost function.

This Chapter explores the features of TBSizer. Section 4.1 addresses the exhaustive (i.e. exact) approach, covering both the problems and advantages of using this. Section 4.2 explains the genetic (i.e. heuristic) approach much in the same way, covering advantages and liabilities of employing it. Finally, Section 4.3 covers the Spice handler software module of TBSizer. This module is responsible for most of the data processing in both, exact and heuristic approaches. The Spice handler tool prepares Spice files, calls an external electrical simulator and collect results, which are later analyzed in the decision taking process in either approach.

4.1 Exhaustive Approach

The exhaustive approach, depicted in Figure 4.2, tests every possible combination of sizing parameters in search of the best possible solution. It generates every possible solution candidate in a loop. Each candidate is simulated using Spectre and some metrics are extracted; e.g. transition time, slope, and power. An user provided cost-function evaluates these metrics and generate a cost value to rank the candidates. After all possible candidates are simulated and evaluated, the tool yields the candidate with the lowest cost as the final result. Since all possibilities are tested, it is guarantee that the final result is the best possible result.

However, the complexity of testing every possible sizing option is bounded by $\mathcal{O}(m^n)$, where m is the number of discreet values each parameter can assume, and n is the number of parameters. Normally, W or L of each transistor are parameters to be search. Increasing the number of transistors thus increases the number of parameters. Since the execution time grows exponentially with the number of parameters, the time required to run the exhaustive approach may become unfeasible rather quickly. Yet, the use of the exhaustive approach might be interesting since: (i) it is guarantee to yield the best possible result, as ranked by a given cost-function; (ii) if the design comprise a small enough transistor network, in this case the computational cost of the

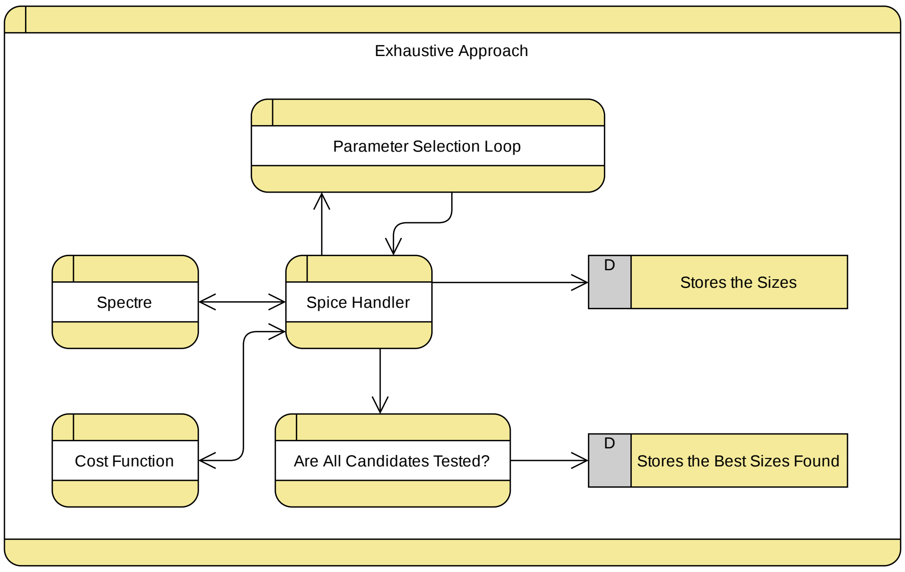


Figure 4.2: The Exhaustive Flow [Author].

exhaustive approach may not hinder its utilization; (iii) the results found by the exhaustive approach enables evaluating the quality of results yield by heuristic approaches, thus evaluating the heuristics.

4.2 Genetic Approach

As discussed in Section 4.1, the computational cost of sizing transistors using an exhaustive approach grows exponentially. This is not a problem for relatively small circuits, but as complexity of circuits grows larger, the execution time becomes prohibitive. In order to overcome this issue, an heuristic algorithm must be adopted to reduce the search-space. This enables finding a satisfactory solution under a feasible time. One such possible heuristic is the *genetic algorithm*, which is at the core of the genetic approach discussed here. The genetic algorithm, as any other heuristic, trades accuracy for performance. However, the result achieved are often good enough to satisfy the sizing problem.

Figure 4.3 depicts the genetic approach flow. The genetic algorithm is based on the natural selection principles as laid by Charles Darwin. Here a number of individuals (i.e. sizing candidates) are selected in each generation (i.e. iterations). The best individual of each generation are bread to generate the next generation. The individuals of each generation are also mutated to avoid local optimums. The user provides number of generations and number of individuals in the configuration file. The genetic approach comprises five main steps: (i) population initialization, where individuals are initialized with random values; (ii) evaluation of individuals, here the individuals are evaluated using spice simulation and the user-provided cost-function; (iii) individual selection, in this step candidates are ranked by their cost and the best candidates are selected for breading; (iv) crossover, here characteristics of the best individuals of the last generation are merged; and (v) Mutation, this step changes the transistor sizes of some individuals at random to introduce variability and broaden the search space, avoiding local minimums. After the individual selection (step iii), the program verifies whether the number of generations is reach. If so, TBSizer yields the best result found and quits. Otherwise, the algorithm continues to the crossover (step iv).

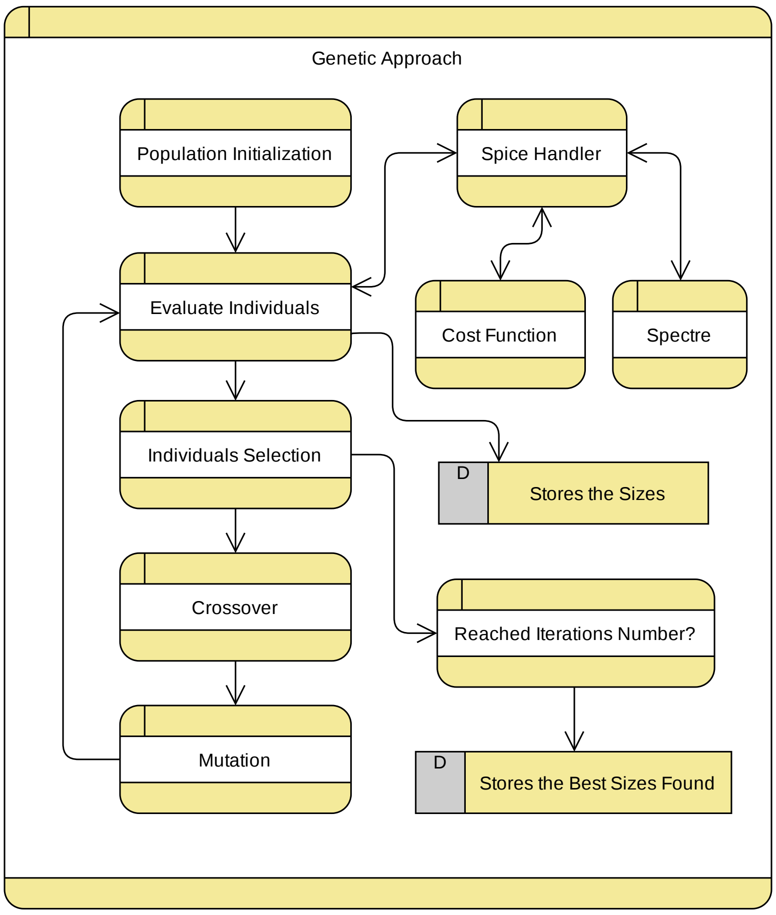


Figure 4.3: The Genetic Flow [Author].

The crossover breads the best individuals by combining their genes (i.e. sizing parameters) using three strategies chosen at random: (i) averaging; (ii) summing; and (iii) subtraction. Furthermore an individual's gene may exert dominance over other, in this case the dominant gene is propagated unchanged. The mutation step affects 10% of the gene pool by randomly bumping or knocking an gene value.

4.3 Spice Handler

The *spice handler* is depicted in Figure 4.4 It sits at the core of the TBSizer, and is used in both approaches. This entity is responsible for simulating and evaluating each candidate. It generates the spice decks, invokes the spice simulator, and collects the simulation results.

The spice handler simulates each candidate each arc[1] at a time. This simulation comprises 6 steps: (i) first an arc is selected to be stimulated, from a pull provided by the designer; (ii) after

[1]An arc is a stimulus applied at one input, which depending on the cell internal state may provoke a change on one or more outputs. An arc is classified as a transition arcs when the change on the input causes change on at least one output, or a hidden arc otherwise.

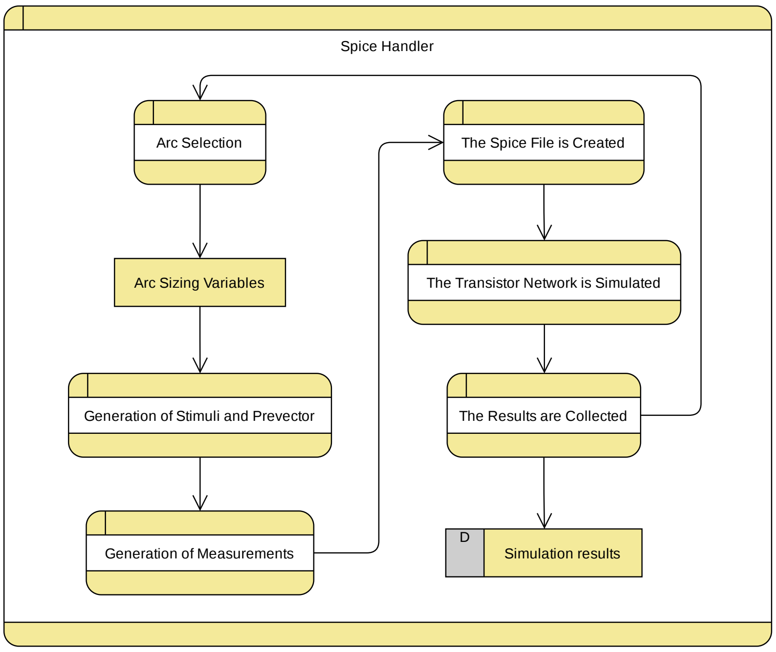

Spice Handler
Arc Selection
Arc Sizing Variables
Generation of Stimuli and Prevector
Generation of Measurements
The Spice File is Created
The Transistor Network is Simulated
The Results are Collected
D
Simulation results

Figure 4.4: The Spice Handler Flow.

selecting the arcs, the second step generates the stimuli defined by the user for that arc, which may include a pre-vector to prepare the internal state of sequential cells; (iii) this step generates the `.measure` commands, used to extract power and timing metrics from the simulation; (iv) after defining the stimuli and measure commands, the spice handler writes the spice deck to the disk; (v) the spice handler invokes Spectre to simulate the deck; (iv) after the simulation is complete, the results are collected.

5. PREPARATION OF THE INPUT DATA

This Chapter covers the flow of preparation of the input data of the software. For this, as can see in Figure 5.1, it is necessary for the hardware designer to supply three files for the software: (1) the transistor network that will be sized, with the parameters that TBSizer can understand, the transistor network, and the parameters that must be found; (2) provide the configurations, responsible for parameterizing the transistor technology that is used, besides that, it determines some global parameters of the software; (3) it is also necessary to prepare the cost function, which TBSizer use as an objective in the exhaustive and genetic search approaches.

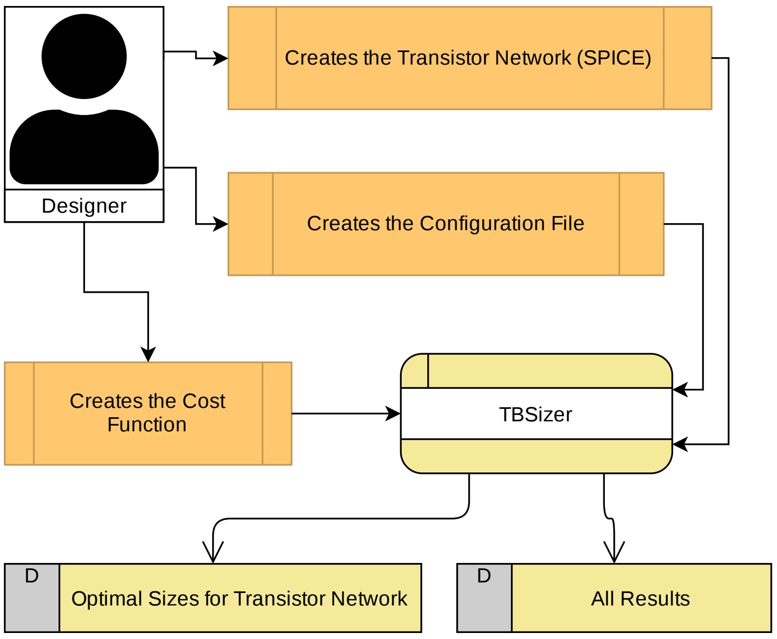

Designer
Creates the Transistor Network (SPICE)
Creates the Configuration File
Creates the Cost Function
TBSizer
D
Optimal Sizes for Transistor Network
D
All Results

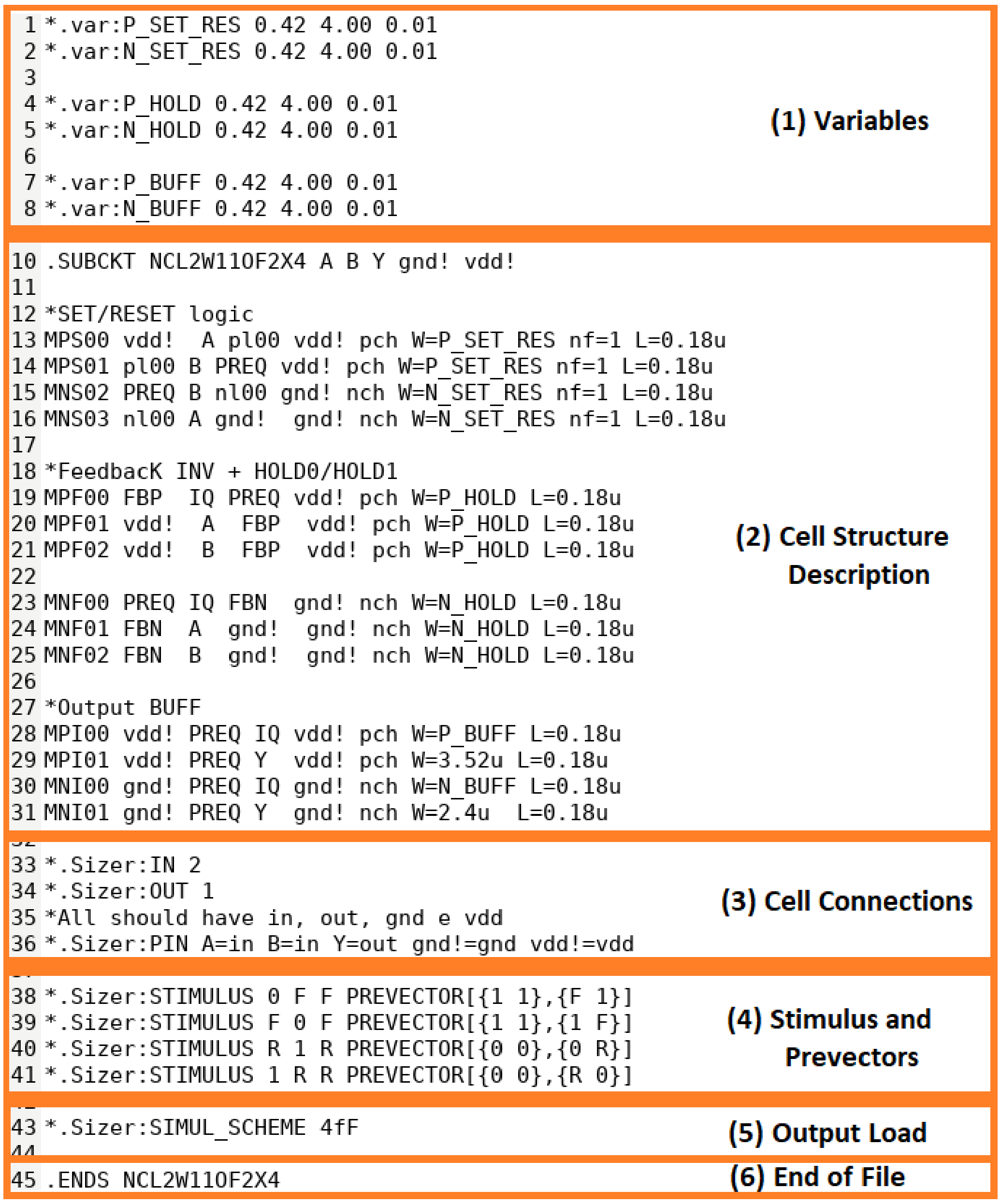
Figure 5.1: A Main Flow of Designer [Author].

In Figure 5.1, it can also be noted that after the creation of the three parameters of the system, it starts processing and generates two main results. (1) File in .CSV format with all sizes tested, results of simulations and results of the cost function. (2) The optimal size of the transistor network found among all those executed by the system.

In the next sections, the details of the files needed to run the system are shown. Section 5.1 details the preparation of the Spice transistor network. In Section 5.2, the preparation of the configuration file is shown in detail. Section 5.3 deals with the implementation of the cost function for TBSizer.

5.1 Preparing the Spice Transistor Network

The process of preparing the Spice transistor network is fundamental to the software. In this step, the transistor network behavior is described, from the variables that are exchanged, the behavior of the circuit, to the circuit itself. With this information ready, the software replicates for each simulation. In Figure 5.2, there is a demonstration of a spice file used by TBSizer. This file are separated into 6 parts, and each part is fundamental for the correct functioning of the system.



```
*.var:P_SET_RES 0.42 4.00 0.01
*.var:N_SET_RES 0.42 4.00 0.01

*.var:P_HOLD 0.42 4.00 0.01
*.var:N_HOLD 0.42 4.00 0.01

*.var:P_BUFF 0.42 4.00 0.01
*.var:N_BUFF 0.42 4.00 0.01

.SUBCKT NCL2W11OF2X4 A B Y gnd! vdd!

*SET/RESET logic
MPS00 vdd!  A pl00 vdd! pch W=P_SET_RES nf=1 L=0.18u
MPS01 pl00 B PREQ vdd! pch W=P_SET_RES nf=1 L=0.18u
MNS02 PREQ B nl00 gnd! nch W=N_SET_RES nf=1 L=0.18u
MNS03 nl00 A gnd!  gnd! nch W=N_SET_RES nf=1 L=0.18u

*FeedbacK INV + HOLD0/HOLD1
MPF00 FBP  IQ PREQ vdd! pch W=P_HOLD L=0.18u
MPF01 vdd!  A  FBP  vdd! pch W=P_HOLD L=0.18u
MPF02 vdd!  B  FBP  vdd! pch W=P_HOLD L=0.18u

MNF00 PREQ IQ FBN  gnd! nch W=N_HOLD L=0.18u
MNF01 FBN  A  gnd!  gnd! nch W=N_HOLD L=0.18u
MNF02 FBN  B  gnd!  gnd! nch W=N_HOLD L=0.18u

*Output BUFF
MPI00 vdd! PREQ IQ vdd! pch W=P_BUFF L=0.18u
MPI01 vdd! PREQ Y  vdd! pch W=3.52u L=0.18u
MNI00 gnd! PREQ IQ gnd! nch W=N_BUFF L=0.18u
MNI01 gnd! PREQ Y  gnd! nch W=2.4u  L=0.18u

*.Sizer:IN 2
*.Sizer:OUT 1
*All should have in, out, gnd e vdd
*.Sizer:PIN A=in B=in Y=out gnd!=gnd vdd!=vdd

*.Sizer:STIMULUS 0 F F PREVECTOR[{1 1},{F 1}]
*.Sizer:STIMULUS F 0 F PREVECTOR[{1 1},{1 F}]
*.Sizer:STIMULUS R 1 R PREVECTOR[{0 0},{0 R}]
*.Sizer:STIMULUS 1 R R PREVECTOR[{0 0},{R 0}]

*.Sizer:SIMUL_SCHEME 4fF

.ENDS NCL2W11OF2X4
```

Figure 5.2: Example Input Transistor Network for TBSizer [Author].

1. The first step addresses the environment variables used for the design. It can be noted that because they are variable, they can be used in any region of the circuit, as long as that region accepts the mapping performed by Spice technology. Numerous variables can also be created, with no limit to the system. Then it is possible to size the width of the transistor (W), the length of the channel of this transistor (L), in addition to other regions if necessary. Another characteristic that can be noticed is that each variable has 4 parameters, and all are used in the system: (1) the first parameter is the name that will be used for this variable, and the name that should be used when developing the transistor network; (2) the second parameter is the minimum size that this variable can assume, regardless of the method used; (3) the

third parameter is the maximum size that the variable can assume; (4) and finally, the last parameter is a minimum step that must be taken by the variable in each new iteration. In addition, parameters two, three and four are not mandatory, but having these parameters in the transistor network, simplifies the implementation and control of these values.

2. The second step is to develop the transistor network. It can be noted that it is a C-Element NCL, where the cell output have fixed sizes. Further, has 6 classes of variables being used to size the cell (i.e. P_SET_RES, N_SET_RES, P_HOLD, N_HOLD, P_BUFF, N_BUFF).

3. The third step is where the transistor network inputs and outputs are described, so that TBSizer can understand the type of connection used. The designer must declare the total number of inputs and outputs, in addition to informing which are the input pins, the output pins, ground (gnd) and voltage (vdd) pins. All of these connections are required by the software.

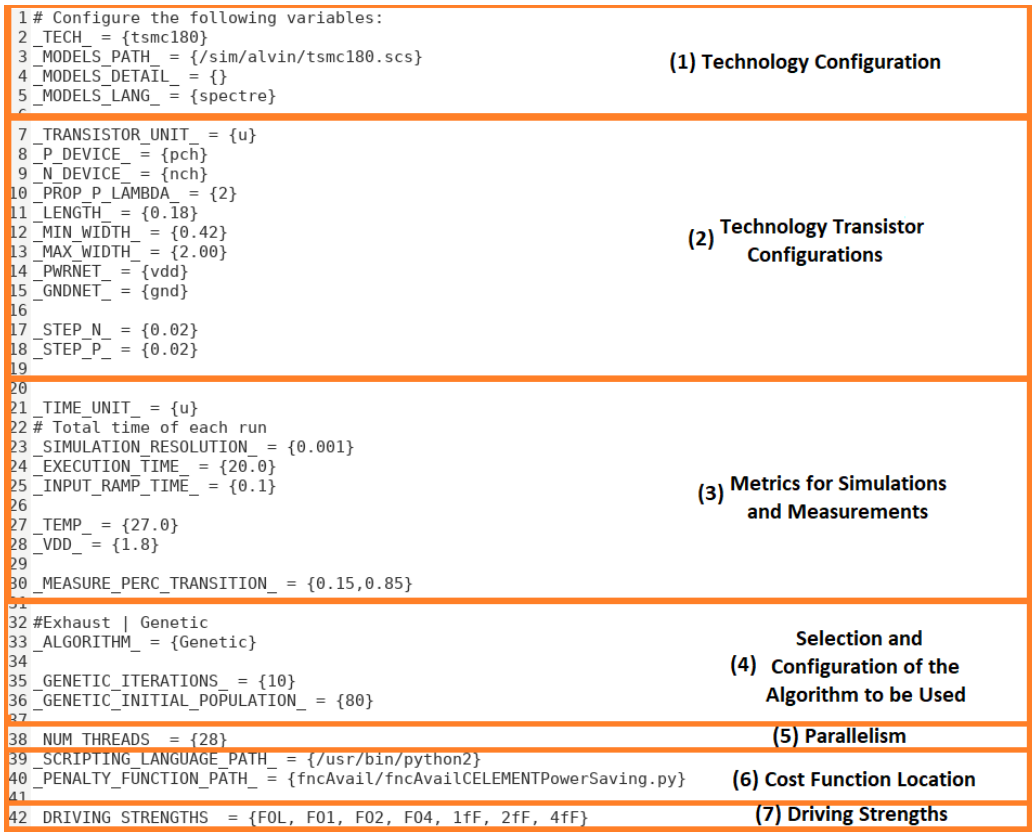
4. The fourth step is where the designer's knowledge is applied. The Designer needs to know the internal arcs of the transistor network very well, and to know how are the possibilities to reach this internal arc. This step brings the stimuli that are performed the measurements, and the pre-vector of the inputs that the software must have, before the measurements are made. Moreover, can noticed that the C-Element with two inputs and one output, has 4 arcs and each arc has two pre-vectors. That is, for the first arc to be tested in the system (line 38). The power supplies (i.e. input fields) should start at a high logic state. After that, the first entry falls. Finally, the fall stimulus on the second input is performed and together, measurements are made with the fall of the output.

5. The fifth step is where the load options for the transistor network are configured. The transistor network may be required for certain types of load, and in this step two types can be assumed, static capacitance, or Fanout-of-four (FO4). If static capacitance is selected, the size of the capacitor can be chosen. If FO4 is selected, the inverters are dimensioned with the minimum technology size.

6. Last but not least, it is necessary to finalize the file after all the registered settings.

5.2 Preparation of the Configuration File

The preparation of the technology to be used in the system is realized in the configuration file. As can be seen in Figure 5.3, the configuration file is divided into seven parts and is a fundamental file for the software to understand some parameters of the technology. This file requires that some global variables be completed, which are used in all simulations. In addition, some parameters are used for the load metrics and selection of the algorithm to be used. The steps are described below.


(1) Technology Configuration
(2) Technology Transistor Configurations
(3) Metrics for Simulations and Measurements
(4) Selection and Configuration of the Algorithm to be Used
(5) Parallelism
(6) Cost Function Location
(7) Driving Strengths

```
# Configure the following variables:
_TECH_ = {tsmc180}
_MODELS_PATH_ = {/sim/alvin/tsmc180.scs}
_MODELS_DETAIL_ = {}
_MODELS_LANG_ = {spectre}

_TRANSISTOR_UNIT_ = {u}
_P_DEVICE_ = {pch}
_N_DEVICE_ = {nch}
_PROP_P_LAMBDA_ = {2}
_LENGTH_ = {0.18}
_MIN_WIDTH_ = {0.42}
_MAX_WIDTH_ = {2.00}
_PWRNET_ = {vdd}
_GNDNET_ = {gnd}

_STEP_N_ = {0.02}
_STEP_P_ = {0.02}


_TIME_UNIT_ = {u}
# Total time of each run
_SIMULATION_RESOLUTION_ = {0.001}
_EXECUTION_TIME_ = {20.0}
_INPUT_RAMP_TIME_ = {0.1}

_TEMP_ = {27.0}
_VDD_ = {1.8}

_MEASURE_PERC_TRANSITION_ = {0.15,0.85}

#Exhaust | Genetic
_ALGORITHM_ = {Genetic}

_GENETIC_ITERATIONS_ = {10}
_GENETIC_INITIAL_POPULATION_ = {80}

 NUM THREADS  = {28}
_SCRIPTING_LANGUAGE_PATH_ = {/usr/bin/python2}
_PENALTY_FUNCTION_PATH_ = {fncAvail/fncAvailCELEMENTPowerSaving.py}

 DRIVING STRENGTHS  = {F0L, F01, F02, F04, 1fF, 2fF, 4fF}
```

Figure 5.3: Configuration file for TBSizer.

1. The first step is where the location of the technology corners file is mapped. It can be noted that four variables are used, they describe the technology used, the technology corners file, if it is necessary to use any section of the corners file, it can be described in the variable _MODELS_DETAIL_, and finally, the technology language is shown.

2. The second step is to configure the technology transistor parameters. It can be noted that the variables _P_DEVICE_ and _N_DEVICE_ must be filled in with the name of the PMOS and NMOS transistors respectively. In addition, there is the unit of measurement used for the parameters. The _PROP_P_LAMBDA_ variable is the size of the width that the PMOS transistor must have in relation to NMOS when generating FO4. In addition, in this step, standard design information is also filled in. This information is usually replaced with the information from the transistor network representation in section 5.1.

3. The third part of the settings is related to the simulations themselves, there is information about the simulation time, the resolution of the simulation, the times of the input ramp, the temperature to be used, the supply voltage, and the slew rate measurements.

4. The fourth part is the selection of the algorithm that is used to find the optimal transistor network sizes. It is possible to choose between two approaches, exhaustive and genetic. If the genetic approach is selected, it is also possible to configure two parameters, such as the number of iterations that are performed, and the number of individuals that are generated for each iteration.

5. In the fifth step, parallelism can be used, where it is possible to choose the number of threads that are used in the software. This selection is valid both for the exhaustive approach and for the genetic approach.

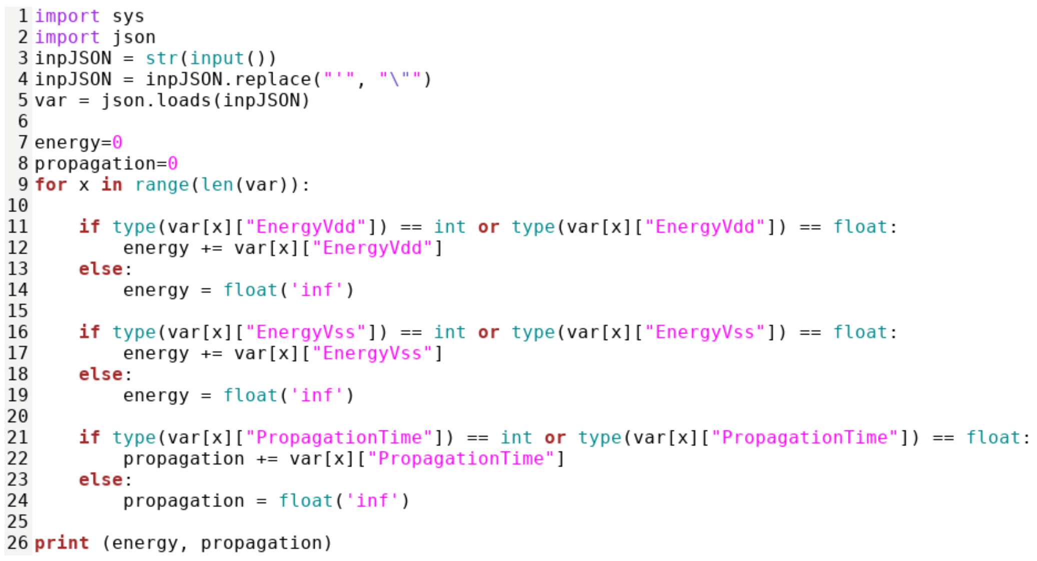
6. The sixth step makes it possible to choose the programming language that is used for the cost function, as well as the location of the cost function.

7. Finally, the last stage of the settings file is for information only, it contains the possibilities of driving strengths of the transistor network. It can be noted that this variable has a parameter with the name FOL, where it represents only one inverter at the transistor network output, and this one with PMOS and NMOS of minimum size of the technology. There are also FO1, FO2, FO4, which are quantities of inverters at the transistor network output. In addition, it is possible to use capacitors as a charge, and for that, it is necessary to specify only the size of the transistor in the transistor network file.

5.3 Cost Functions

The cost functions are fundamental for the correct functioning of the software. The software uses the results of the cost function as its objective, and for that, the problem that is being addressed must be analyzed and the results that should be generated correctly projected. In Figure 5.4, there is an example of a cost function previously provided by the software. This cost function is based on power saving, where it tries to reduce energy consumption while reducing propagation time.

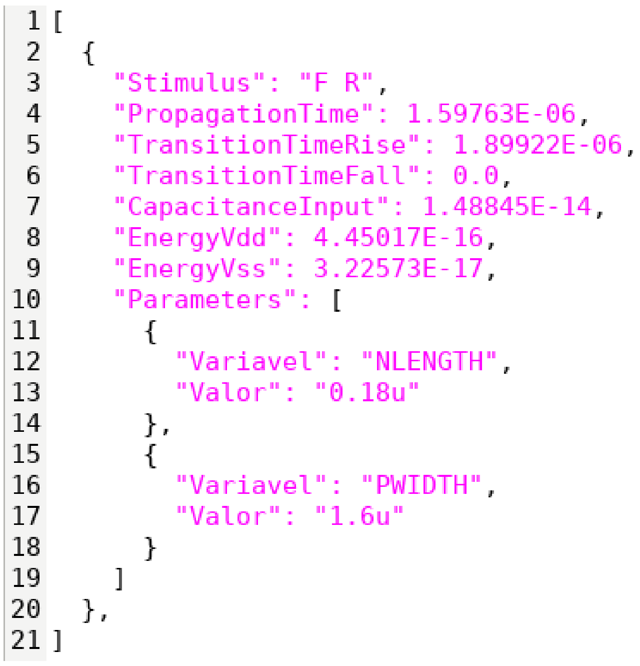
It can be seen in line 3, that standard input is used for passing parameters. After that, the conversion to object is performed. Starting from line 9, there are the sums needed to meet the power saving of the circuit.



```
 1 import sys
 2 import json
 3 inpJSON = str(input())
 4 inpJSON = inpJSON.replace("'", "\"")
 5 var = json.loads(inpJSON)
 6
 7 energy=0
 8 propagation=0
 9 for x in range(len(var)):
10
11     if type(var[x]["EnergyVdd"]) == int or type(var[x]["EnergyVdd"]) == float:
12         energy += var[x]["EnergyVdd"]
13     else:
14         energy = float('inf')
15
16     if type(var[x]["EnergyVss"]) == int or type(var[x]["EnergyVss"]) == float:
17         energy += var[x]["EnergyVss"]
18     else:
19         energy = float('inf')
20
21     if type(var[x]["PropagationTime"]) == int or type(var[x]["PropagationTime"]) == float:
22         propagation += var[x]["PropagationTime"]
23     else:
24         propagation = float('inf')
25
26 print (energy, propagation)
```

Figure 5.4: Cost Function for TBSizer.

In Figure 5.5, there is an example of the JSON parameters sent by TBSizer to the cost function. Where a list containing all parameters generated by the simulation of the arcs. The parameters sent are as follows: (1) *Stimulus*, responsible for showing which arc is being processed; (2) *PropagationTime*, this parameter has the propagation time that the transistor network needed to simulate this arc; (3) *TransitionTimeRise*, this field shows the rise time of the transistor network output; (4) *TransitionTimeFall*, this shows the fall time of the transistor network output; (5) *CapacitanceInput*, shows the input capacitance of the transistor network; (6) *EnergyVdd*, energy consumption between the source and the transistor network; (7) *EnergyVss*, energy consumption between ground and the transistor network; (8) *Parameters*, there are the sizes of the transistor network variables.



```
[
  {
    "Stimulus": "F R",
    "PropagationTime": 1.59763E-06,
    "TransitionTimeRise": 1.89922E-06,
    "TransitionTimeFall": 0.0,
    "CapacitanceInput": 1.48845E-14,
    "EnergyVdd": 4.45017E-16,
    "EnergyVss": 3.22573E-17,
    "Parameters": [
      {
        "Variavel": "NLENGTH",
        "Valor": "0.18u"
      },
      {
        "Variavel": "PWIDTH",
        "Valor": "1.6u"
      }
    ]
  },
]
```

Figure 5.5: An example of contents for the cost function specification file.

In addition, four standard cost functions are provided: (1) power saving, where it focuses on reducing energy consumption, while trying to reduce the total signal propagation time; (2) area reduction, where it projects a transistor network that can propagate the signal to the exit and with the smallest possible area; (3) less time difference between the propagation of a rise signal and a fall signal; (4) average function, try to find the average design among all characteristics. In some cases, such as the cost function three, it is necessary to specify the arcs in which the outputs rise and fall, as the software supports n outputs.

6. TBSIZER EXPERIMENTAL RESULTS AND COMPARISONS

This Chapter covers some of the capabilities of the TBSizer software. This Chapter is divided as three Sections each dedicated to one type of experiments conducted with TBSizer. Note that the type of experiments reported here are not the only ones possible, TBSizer capabilities are more encompassing than these simple experiments show. First, Section 6.1 addresses a comparison between TBSizer and the combination of the Ring Oscillator Generator and the Cell Specifier [Mor16] software tools. These tools constitute a sizer previously developed in the same research group of the Author. Second, Section 6.2 explores how TBSizer allows that the transistors in a network can be grouped into transistor classes that are sized in the same way, which allows users to trade the software runtime against the quality of the obtained sizing results. This can have deep impact on the dimensions of the treatable transistor networks. Finally, Section 6.3 demonstrates how TBSizer can be used to resize already sized transistor networks, eventually achieving improvements over a given sub-optimal implementation.

Due to technical reasons (facilities of the drawing tool) transistor networks schematics are showed here with symbols slightly different from those appearing in previous Chapters. Figure 6.1 contrasts previously used conventions (parts (a) and (c) in Figure 6.1) and the respective conventions adopted herein (parts (b) and (d) in Figure 6.1).

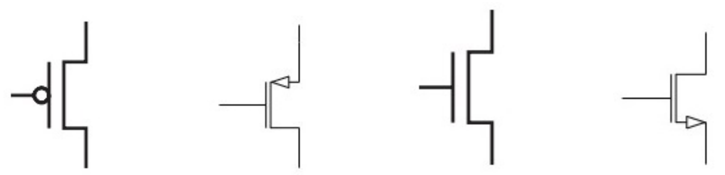

(a) PMOS. (b) PMOS. (c) NMOS. (d) NMOS.

Figure 6.1: Previous and new convention to represent CMOS transistors.

6.1 TBSizer versus ROGen/CeS

ROGen/CeS [Mor16] work as follows: (1) ROGen receives a cell description to size; (2) it iteratively produces standard ring oscillators (ROs) structures with a specified number of stages, using the input cell; at each iteration, the cell is sized differently, to cover a range of user-specified sizes; (3) ROGen next calls a Spice program (specifically the Cadence Spectre software, others can be used as well) to simulate the RO, collecting the simulation results. CeS employs a user-defined cost function to look for the sized cell that minimizes this function among all simulations fired by ROGen. This sized cell is the output of the sizer. ROGen uses an exhaustive process, where only two parameters (PPARAM and NPARAM, respectively the sizes of pMOS and nMOS transistors) are varied to find the best dimensions, according to a user-selected cost function. The ROGen/CeS combination captures figures like dynamic power, leakage power, transition delays, propagation delays and the ring propagation delay.

TBSizer is distinct from ROGen/CeS, but shares some features with the latter. As ROGen/CeS, it uses an circuit arc-oriented approach, that enables to explore not only transistor network output effects but is also aware of possible internal network states. The network analysis can find internal critical paths for circuits that have internal states, sizing it to meet specifications.

An advantage of TBSizer over ROGen/CeS is that it allows choosing between exhaustive and genetic sizing approaches. Another major advantage is that any set of transistors in a network can be chosen as target for sizing, while in ROGen/CeS there are just two sizes produced (one for pMOS transistors and another for nMOS transistors). ROGen/CeS require the user to choose which transistors will be sized (with PPARAM and NPARAM), all other transistors are ignored. More about this is explored along with the example cell to use in this Section.

Figure 6.2 shows the transistors of a 2-input C-element that employs the Sutherland transistor topology [Sut89].

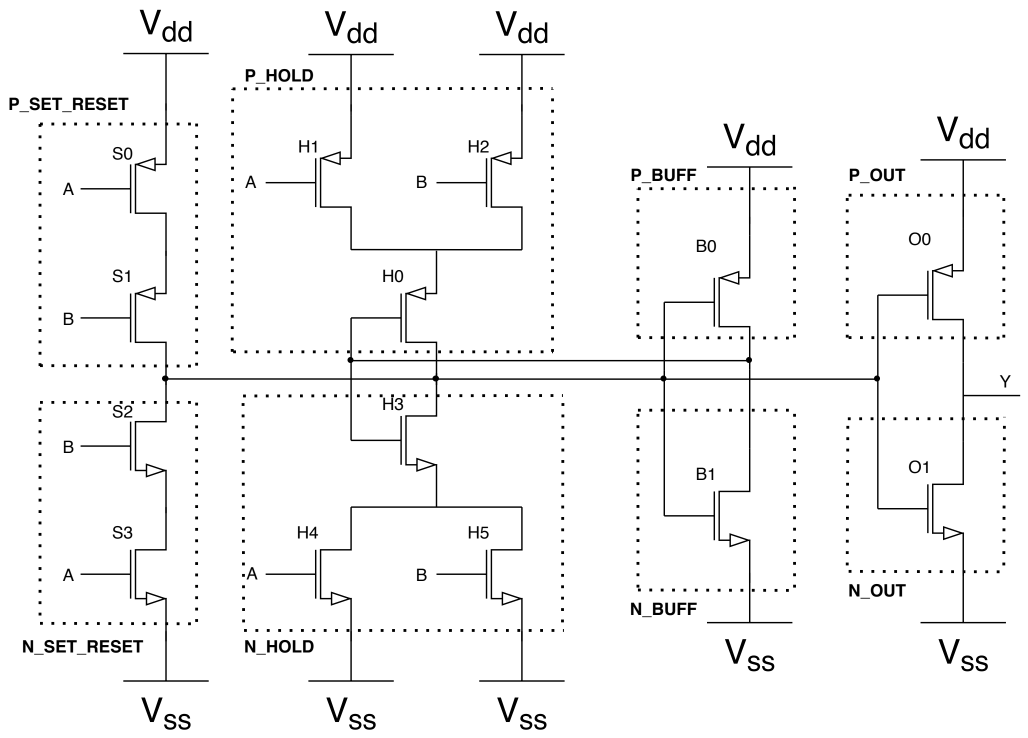


Figure 6.2: A transistor network implementing a 2-input C-element with Sutherland topology. Transistors are partitioned into 8 sizing classes.

The C-element network corresponds to one of the cells of the ASCEnD-TSMC180 library proposed within the scope of the GAPH research group [Sus19], The reader can verify that this network implements the behavior of a 2-input C-element: when inputs A and B are equal, output Y is forced to this same value. However, when inputs differ, the last value that appeared in the output is kept there by the controlled latch formed with the eight transistors in the central region of the cell. The four-transistor stack at the left of Figure 6.2 drive the output Y when A=B.

To facilitate the identification of transistors for sizing, the network is partitioned into eight transistor classes: P_SET_RESET, N_SET_RESET, P_HOLD, N_HOLD, P_BUFF, N_BUFF, P_OUT and N_OUT. The last two single-transistor classes (P_OUT and N_OUT) are the output buffer, with size fixed to guarantee the achievement of the correct network drive strength. In the implementation, the output buffer is in fact an XL (or X1, or X2, or X4) inverter taken from the commercial standard cell library with with which the ASCEnD-TSMC180 library was designed to be compatible. This is the Sage-X standard cell library for the TSMC 180nm bulk CMOS technology commercialized by ARM [ARM08].

In the experiments, TBSizer only sizes the other six transistor classes. For ROGen/CeS the strategy is more limited, it only sizes the P_SET_RESET and N_SET_RESET transistor classes. Other sizes are assigned by the designer using some heuristic.

The next Sections compare four 2-input C-elements sized with TBSizer and ROGen/CeS for the following driving strengths: XL, X1, X2 and X4. Sizing is performed on the mentioned transistors only, to find the lowest average energy consumption for the cell. Only the transistor width (W) is sized, all transistors have fixed length (L=180μm).

Performance results are obtained after sizing using a the Liberate cell characterization tool from Cadence. This tool measures results such as capacitance, leakage current, cell propagation delay and power. The approach here is to find a combination os sizes leading to the lowest average energy per transition of the cell.

6.1.1 A Comparison of Sizing for the C-element with Drive XL

The transistors P_OUT and N_OUT are kept with the sizes defined by the already mentioned ARM library, which are P_OUT=0.64μm and N_OUT=0.42μm. The other variables are sized by TBSizer. As for ROGen/CeS, transistors P_BUFF and N_BUFF are fixed as an XL ARM inverter and transistors in classes P_HOLD and N_HOLD are all minimum size (W=0.42μm, L=180μm).

Figure 6.3 shows the best transistor sizes (W) for the C-element as suggested by ROGen/CeS and TBSizer. The biggest difference is that TBSizer drastically increases the size of the P_SET_RESET and N_SET_RESET transistors, increases the N_BUFF transistor size and decreases the P_BUFF transistor size.

Figure 6.4 compare the performance of the cells after sizing. The sizing generated by TBSizer increases the input capacitance of the cell by approximately 23%, from 3.0fF to 3.7fF. Although the absolute values are not significant, the change is noticeable. Leakage power increases by about 3.9%, from 0.0051nW to 0.0053nw, which is not quite relevant. The Delay Rising figure is in fact the cell propagation delay for the logic 1 value, which is 10.24% smaller for the TBSizer cell, going from 0.1045ns to 0.0938ns, while the Delay Falling (the cell propagation delay for the logic 0 value) reduces 18.83% in the TBSizer cell, from 0.2299ns to 0.1866ns, corresponding to an average propagation delay reduction of 16.14%. Also, TBSizer cells increase power consumption by

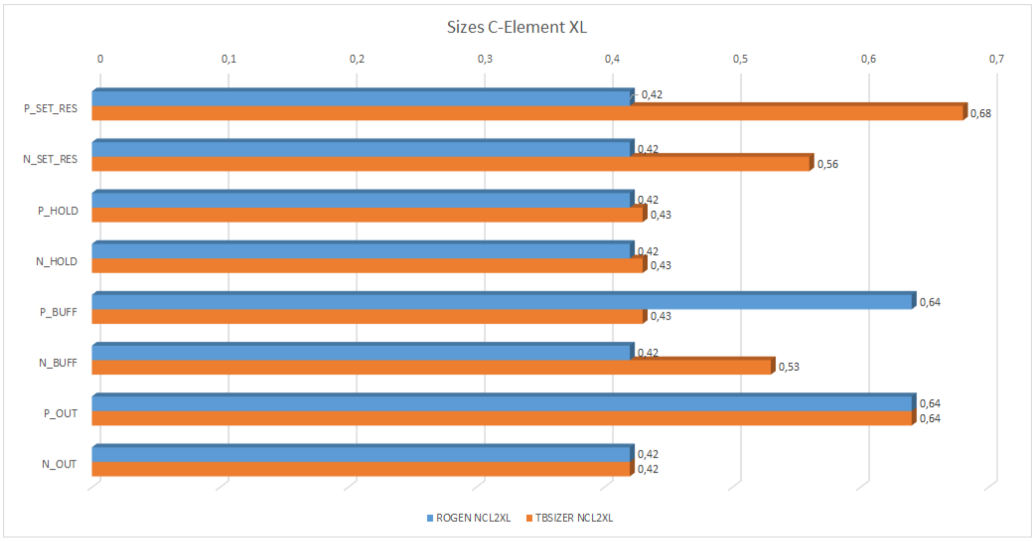


Figure 6.3: ROGen and TBSizer sizes (W) comparison for the XL C-element. Units are in μm.

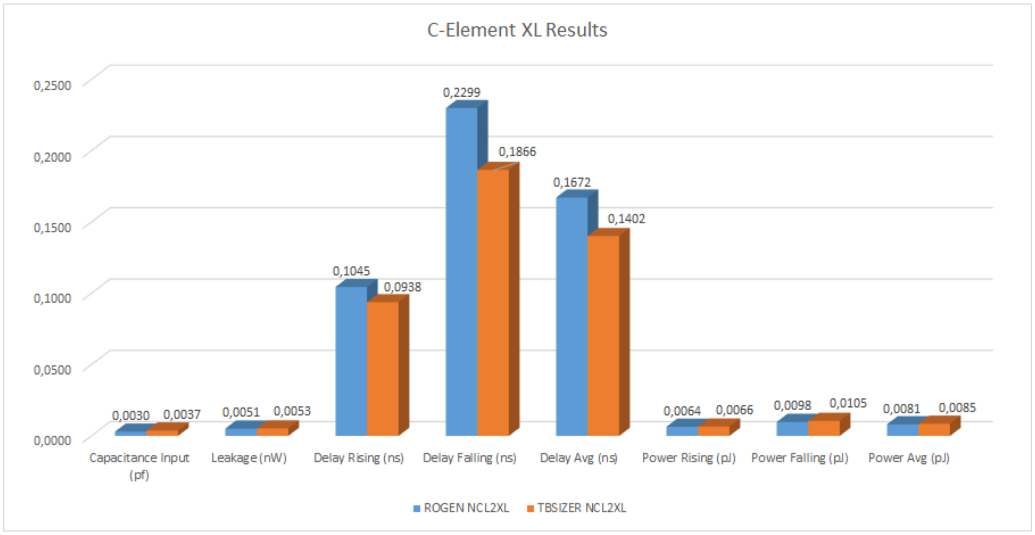


Figure 6.4: ROGen and TBSizer performance comparison for the XL C-element.

4.9% in average. The sizing provided by TBSizer increases energy consumption while it reduces the propagation time of the cell. Cell performance is thus increased using TBSizer. Another advantage of using TBSizer is a 30% reduction in the difference between the times to propagate logic 1s and 0s.

6.1.2 A Comparison of Sizing for the C-element with Drive X1

The same comparison method developed in Section 6.1.1 is applied here to the X1 C-element sizing. Figure 6.5 brings the data on the new sizing. The transistor classes P_SET_RESET and N_SET_RESET show large differences among their respective sizes. TBSizer assigns to the class P_SET_RESET a W=0.73μm, while ROGen computes W=0.42μm for the same class. TBSizer also assigns to the class N_SET_RESET W=0.55μm, but ROGen chooses W=0.42μm. Given the heuristic choice to size the buffer (P_BUFF and N_BUFF transistors) as an XL ARM inverter, the buffer is quite large, while TBSizer keeps P_BUFF at W=0.45μm, even smaller than the N_BUFF transistor, which has W=0.47μm. The transistor groups P_OUT and N_OUT are fixed with the standard sizes for a drive X1 inverter from the ARM library: P_OUT has W=0.9μm and N_OUT has W=0.6μm.

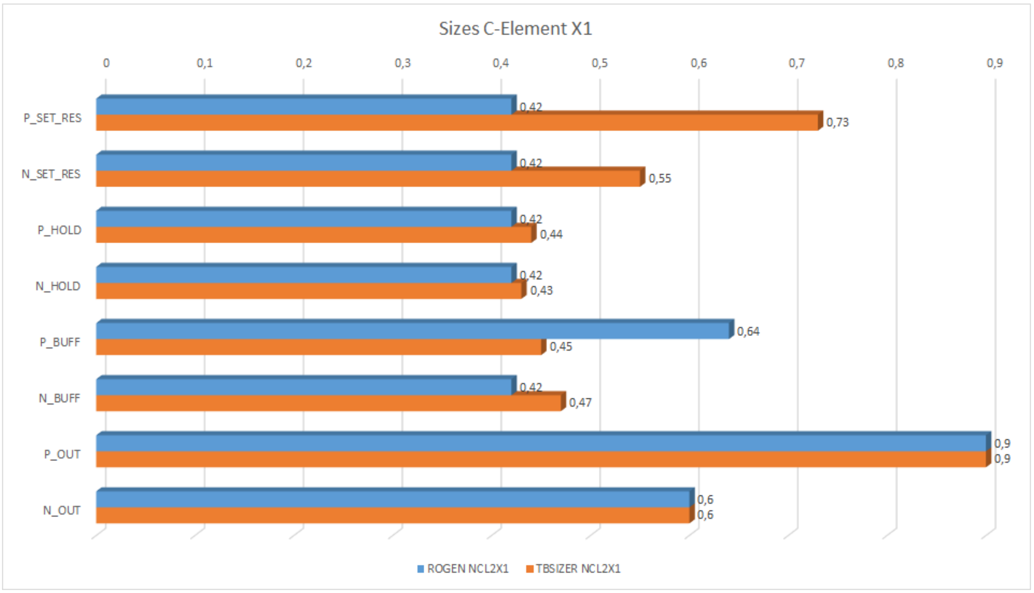


Figure 6.5: ROGen and TBSizer sizes (W) comparison for the X1 C-element.

TBSizer generated quite interesting results here, see Figure 6.6. The input capacitance increased by 26.7%, from 3.0fF generated by ROGen/CeS, to 3.8fF. Leakage Power increased slightly, about 1.8%, from 0.0055nW generated by ROGen/CeS sizing, to 0.0056nW generated by the TB-Sizer sizing.

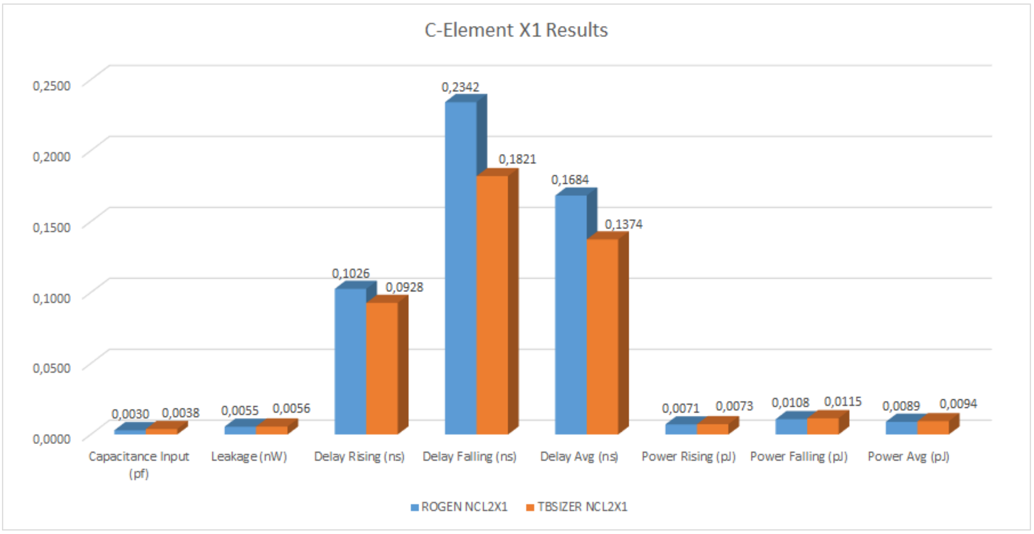


Figure 6.6: ROGen and TBSizer performance comparison for the X1 C-element.

The propagation delay for logic 1s reduces by 9.6%, from 0.1026ns to 0.0928ns, while propagation delay for logic 0s reduces by 22.25%, going from 0.2342ns to 0.1821ns. The average propagation delay reduces by 18.41%, a significant value, going from 0.1684ns to 0.1374ns, in addition to the reduction on the difference between logic 0s and 1s propagation delay. The average power increases about 5.6%, from 0.0089pJ to 0.0094pJ. The cells sized by TBSizer obtained advantages with regard to cell speed, but with a small effect on energy consumption.

6.1.3 A Comparison of Sizing for the C-element with Drive X2

Again, transistors P_OUT and N_OUT are fixed with the sizes defined for an X2 inverter in the ARM library, P_OUT has W= 1.76μm and N_OUT has W=1.2μm. The other six classes are sized as usual by TBSizer, while ROGen/CeS take the same use of heuristics as in the previous sections.

Figure 6.7 shows that the dimensions generated by TBSizer are somehow intriguing. The class P_SET_RESET has a W value defined as 1.2μm by TBSizer, while ROGen assigns to it a W=0.52μm, a difference of 126%. The class N_SET_RESET has its W value defined as 0.67μm by TBSizer, while ROGen sets it to W=0.46μm, corresponding to a 45.6% difference. The N_BUFF transistor size also increases considerably, from 0.42μm to 0.54μm, i.e. 28.57%. But there is also a reduction, the transistor P_BUFF, which ROGen sets W=0.64μm, TBSizer defines W as 0.46μm, reducing it by 28.13%. The other values remained pretty much similar.

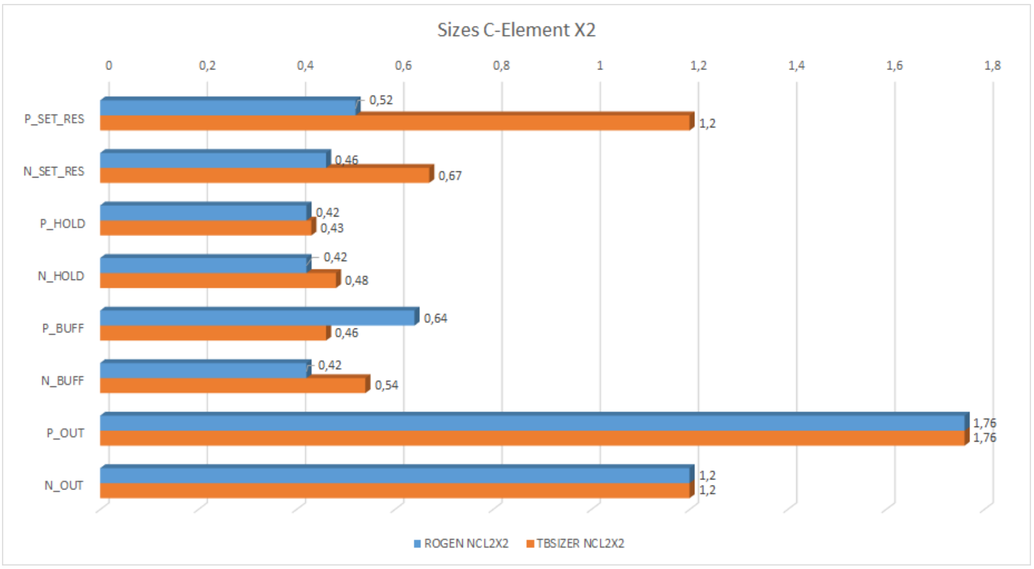


Figure 6.7: ROGen and TBSizer sizes (W) comparison for the X2 C-element.

TBSizer generates quite interesting results for the X2 C-element, see Figure 6.8. The input capacitance increased by 56.25%, from 3.2fF generated by ROGen, to 5.0fF in the cell sized by TBSizer. Leakage Power increases by 10.29%, from 0.0068nW generated by the ROGen sizing, to 0.0075nW produced by the TBSizer sizing. The propagation delay of 1s reduces by 15.05%, from 0.1050ns to 0.0892ns, while the propagation delay of 0s reduces by 33.36%, from 0.2317ns to 0.1544ns.

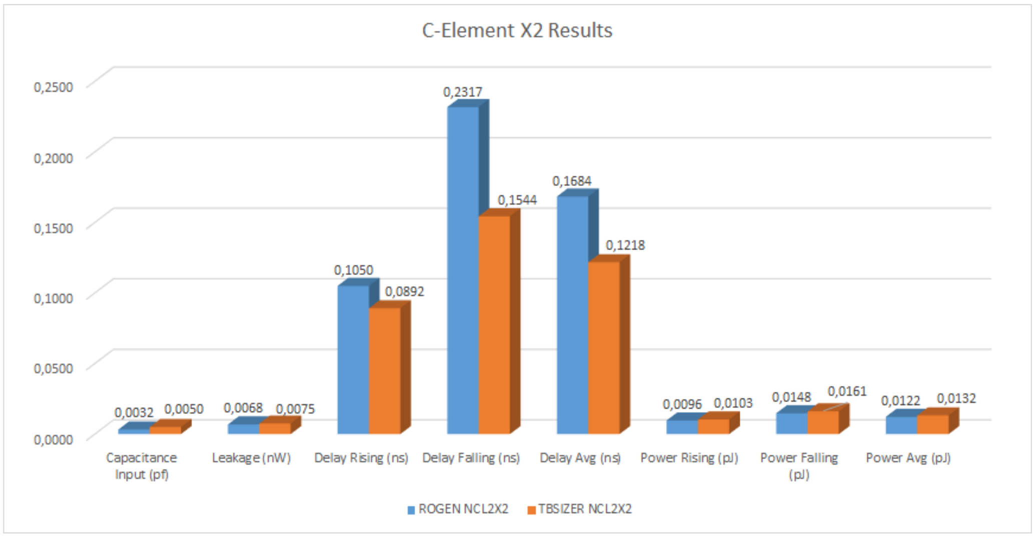


Figure 6.8: ROGen and TBSizer performance comparison for the X2 C-element.

The average propagation delay is reduced by 27.67%, a very expressive value, going from 0.1684ns to 0.1218ns, in addition to the reduction in the difference between the time to propagate 0s and 1s. The average power increases about 8.2%, going from 0.0122pJ to 0.0132pJ. In conclusion, the cells sized by TBSizer for the X2 drive obtained significant advantage with regard to speed and were not significantly impaired in terms of energy consumption. The main advantage of this sizing is that the critical path is about 33.36% smaller, which reduces the average propagation delay.

6.1.4 A Comparison of Sizing for the C-element with Drive X4.

The P_OUT and N_OUT transistors have fixed sizes, as defined by the X4 inverter of the ARM library: P_OUT has W=3.52μm and N_OUT has W=2.4μm. Figure 6.9 shows the dimensions generated by TBSizer and by ROGen do not display much difference, but TBSizer keeps the input transistor sizes larger than those computed by ROGen. The class P_SET_RESET has a value of W defined as 1.2μm by TBSizer, while ROGen sets W=0.72μm, a difference of 66.67%. The class N_SET_RESET has a value of W defined as 0.99μm by TBSizer, while ROGen sets W=0.78μm, a difference of 26.92The N_BUFF transistor also increases considerably in TBSizer computations, from 0.42μm to 0.53μm, i.e. 26.19%. The reduction occurs the size of P_BUFF that ROGen defines as W=0.64μm, while TBSizer defines it as W=0.46μm, a reduction of 28.13%. The remaining sizes are similar.

Figure 6.10 shows that TBSizer generates interesting results for the X4 C-element. For example, the input capacitance increases by 30.95%, from 4.2fF generated by ROGen, to 5.5fF by TBSizer. Leakage power increases by about 4%, from 0.0100nW as generated by the ROGen sizing, to 0.0104nW as generated by the TBSizer sizing. The average propagation delay is reduced by 21.33%, from 0.1613ns to 0.1269ns, which is quite interesting, in addition to the reduction in the difference between the delay for propagating 0s and 1s through the cell. The average Power reduces

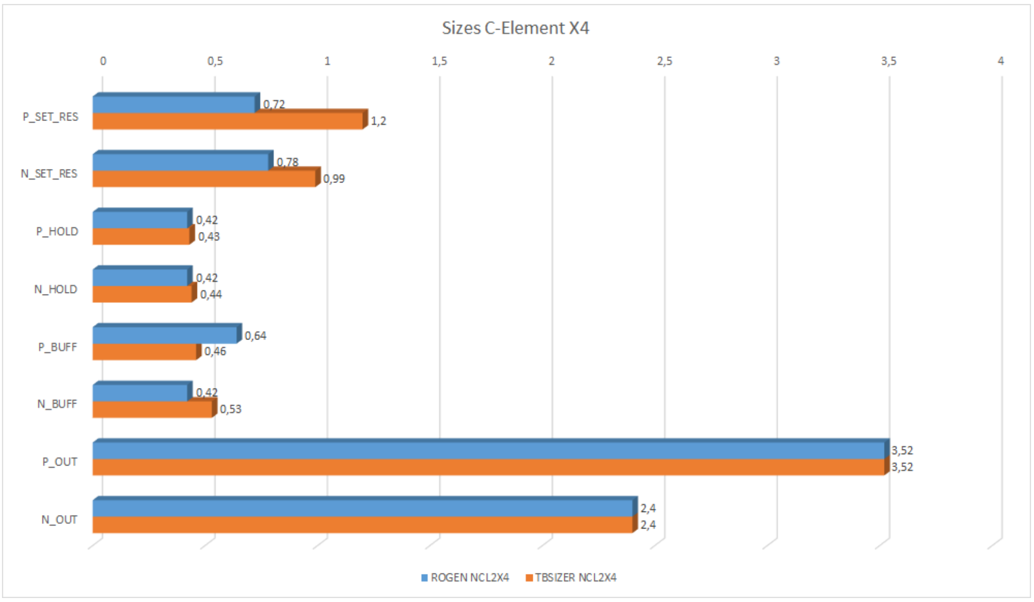


Figure 6.9: ROGen and TBSizer sizes (W) comparison for the X4 C-element.

by about 2.55%, from 0.0196pJ to 0.0191pJ. In conclusion, the sizing of the X4 C-element achieved speed advantage of about 21.33% and an energy reduction of about 2.55%.

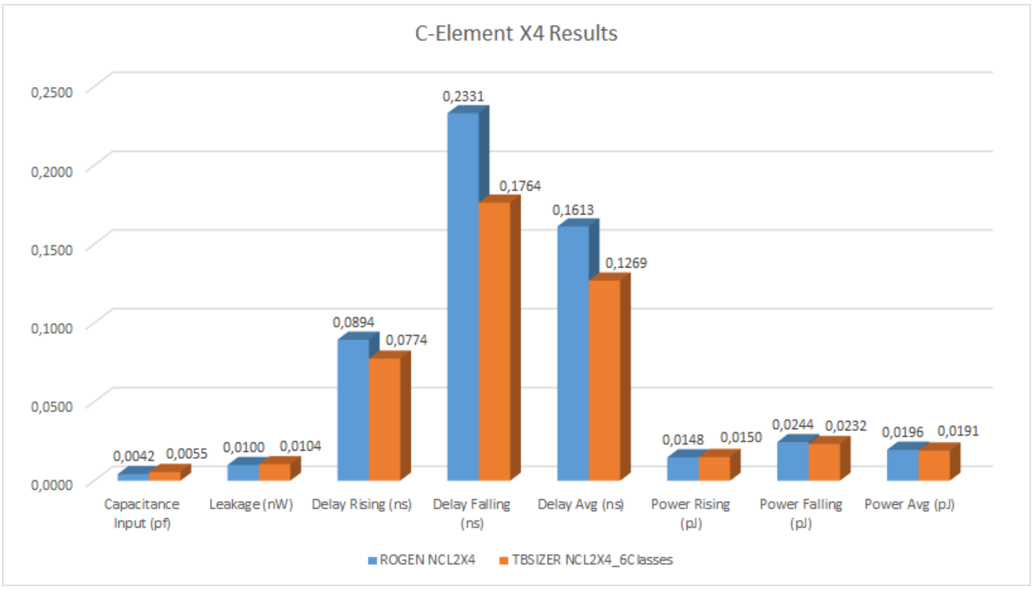


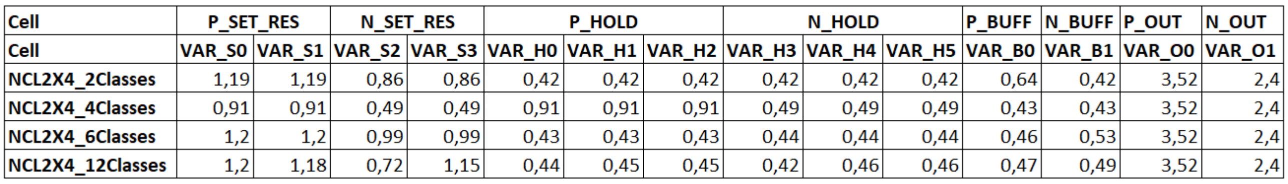
Figure 6.10: ROGen and TBSizer performance comparison for the X4 C-element.

6.2 The Transistor Grouping Feature for Sizing with TBSizer

TBSizer is software capable of sizing transistors of a given network. Often, several transistors in a network share some topological characteristic, such that it can be beneficial to size that in the same way. This gives rise to the concept of transistor classes to size together. The concept has been informally explored in Section 6.1 and will be further investigated here. The way TBSizer can be informed of transistor classes is using the so-called *variables* in the cell structural description file, as described in Section 5.1. Also, TBSizer can use more than one transistor sizing approach, choosing between the exhaustive or the genetic approaches. The exhaustive method is constrained in terms of execution time. Thus, in almost all tests the genetic approach is used. This Section explores four tests using TBSizer, mainly differing in the used transistor grouping strategy. The case study used in this Section is the same used in the previous Section, the 14-transistor, 2-input, Sutherland topology C-element.

The method for sizing the C-element of Figure 6.2 using different transistor groupings is as follows: (1) TBSizer sizing two classes of transistors, where only the P_SET_RESET and N_SET_RESET are sized and the other values are kept as defined by the ROGen/CeS heuristic treatment, explored in Section 6.1; (2) TBSizer sizing four classes of transistors, where the classes P_SET_RESET and P_HOLD are unified, the classes N_SET_RESET and N_HOLD are also unified, and the classes P_BUFF and N_BUFF are also used, totaling four sizing classes; (3) TBSizer sizing six classes of transistors, as discussed in the previous section; (4) TBSizer sizing twelve classes of transistors, where each class is exactly one of the C-element transistors (except of course for the output buffer, P_OUT and N_OUT, which are fixed for each drive strength). All experiments in this Section employ just one drive strength, X4.

Figure 6.11 shows the different transistor sizing achieved after running TBSizer with the four transistor grouping strategies. One interesting pattern that stands out in 12Classes C-element, is (refer to Figure 6.2 for the transistor positioning) transistors closest to the power supply, or to ground, have sizes larger than those transistor more distant. Analyzing the series between the transistors VAR_S3 which is with one of the terminals on the ground and VAR_S2, which is not, there is a difference of 37.39% on size. This is result that reduces input capacitance, decreases energy consumption, while not significantly affecting the network propagation time. This type of sizing is only feasible with the genetic approach, due to the execution time limitations imposed by the exhaustive approach, when dealing with 3 or more distinct transistor classes. Even if a global optimum sizing result is not guaranteed to be found, the benefits might be clear.



Cell	P_SET_RES		N_SET_RES		P_HOLD			N_HOLD			P_BUFF	N_BUFF	P_OUT	N_OUT
Cell	VAR_S0	VAR_S1	VAR_S2	VAR_S3	VAR_H0	VAR_H1	VAR_H2	VAR_H3	VAR_H4	VAR_H5	VAR_B0	VAR_B1	VAR_O0	VAR_O1
NCL2X4_2Classes	1,19	1,19	0,86	0,86	0,42	0,42	0,42	0,42	0,42	0,42	0,64	0,42	3,52	2,4
NCL2X4_4Classes	0,91	0,91	0,49	0,49	0,91	0,91	0,91	0,49	0,49	0,49	0,43	0,43	3,52	2,4
NCL2X4_6Classes	1,2	1,2	0,99	0,99	0,43	0,43	0,43	0,44	0,44	0,44	0,46	0,53	3,52	2,4
NCL2X4_12Classes	1,2	1,18	0,72	1,15	0,44	0,45	0,45	0,42	0,46	0,46	0,47	0,49	3,52	2,4

Figure 6.11: TBSizer sizes (W) comparison for the X4 C-element using different transistor groups for sizing. Units are in μm.

Refer to Figure 6.12. Analyzing the input capacitance, the average difference is around 12%, maintaining values between 4.9fF generated by the 4Classes approach up to 5.5fF, for the 6Classes approach. These values are very small, and the increase in capacitance is also small. Leakage Power varies about 3%, going from 10.1pW, generated by the 4Classes design, to 10.4pW, generated by the 6Classes design. The propagation delay of 1s shows a big difference for the

4Classes approach sizing, about 87.6%. The other results of 1s propagation delay vary around 12%. Propagation delay of 0s also increased for the 4Classes design, but this time around 17.67%. The difference between rising and falling propagation delays reduces significantly for the 4Classes design.

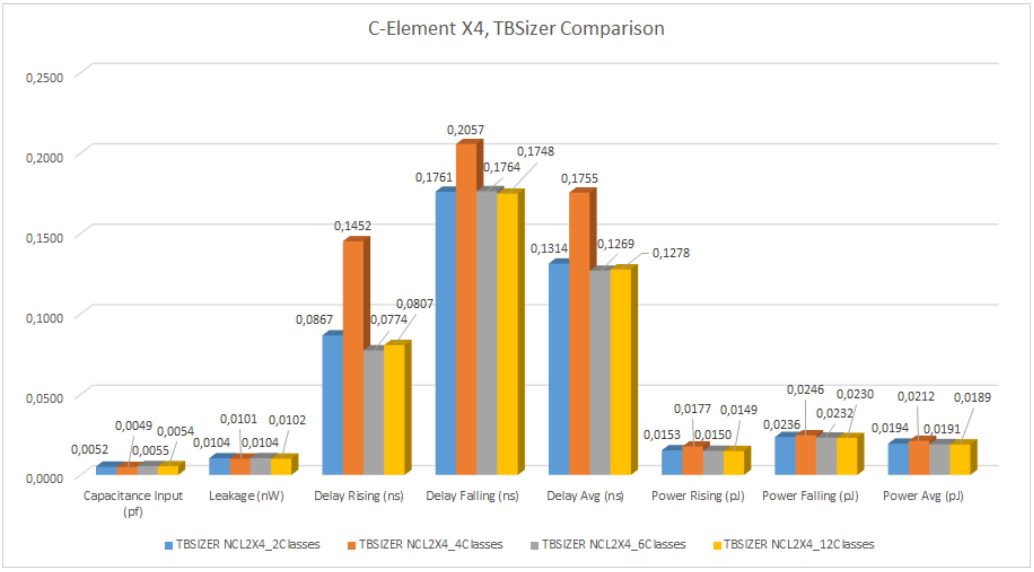


Figure 6.12: TBSizer performance results for transistor classes comparison over the X4 C-element.

The average propagation delay varied by 38.29%, a quite large value, but much of this increase is due to the significant increase in the propagation delay of 1s in the 4Classes approach. The power rising varied by 18.79%, and the power falling varied by 6.96%, which gives an average power variation around 12.17%. The cells sized with the 4Classes approach have a very distinct behavior with regard to the other transistor grouping strategies.

The most similar results were the sizing with 2Classes and the sizing with 6Classes. However, the best results are provided by the sizing with 12Classes, where each transistor is individually sized. In the 12Classes design there is a reduction in leakage, a reduction in the critical path delay, and a reduction in energy consumption.

6.3 Resizing Transistors of a Sub-Optimally Sized Transistor Network with TBSizer

The methodology adopted for the comparison between cells already sized by TSMC 180 with TBSizer is based on the lowest average energy consumption. The TSMC 180 provides several cells already sized to be used in projects, so four combinational cells are selected, and the sizing transistor is used, where each transistor is a sizing variable. The combinational cells chosen are: (1) an AND2X4, which contains two inputs and one output with the drive X4, and only the internal transistors of the cell are sized; (2) an OR2X4, which contains two inputs and one output with the drive X4, the internal transistors of the cell are resized; (3) a NAND2X4, which contains two inputs and one output with the drive X4 (note that NAND is a special case, where all transistors are sized: as there is no inverter at the output with the X4 drive sizes, the drive is given by the number of fingers in the cell); (4) an XOR2X4, which is sized in ten variables, keeping the output transistors with their X4 drive.

6.3.1 The AND X4 Cell Case

Figure 6.13 shows the schematic representation of the AND cell approached in this Section.

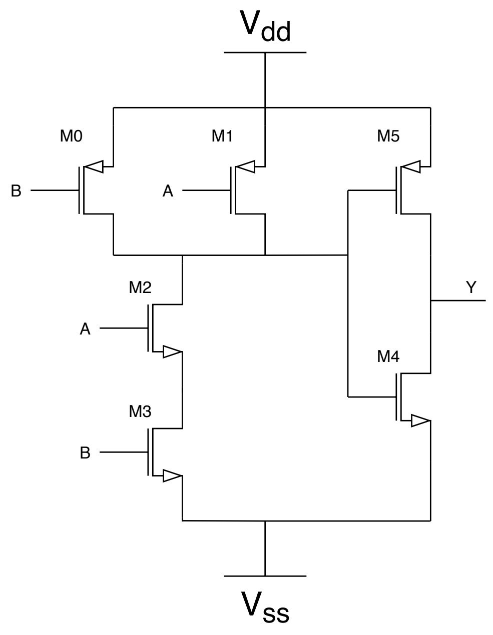


Figure 6.13: The AND cell schematic.

The circuit has six transistors, three PMOS transistors and three NMOS transistors. Sized transistors are M0, M1, M2 and M3. The other transistors, M5 and M6, are kept in their sizes to provide the X4 drive.

Figure 6.14 shows the default sizes provided by TSMC 180 in blue, and the sizes sized by TBSizer in orange.

Note that the total cell area is reduced by the TBSizer sizing. Transistors M0 and M1 decreased 21.67%, while transistor M2 reduced 67.94% and transistor M3 reduced 32.82% in width. M4 and M5 transistors did not change their size, they remained at 2.36u and 3.62u respectively, to respect the X4 drive.

Results provided by Liberate can be seen in Figure 6.15, where in blue are the results for the standard design of the TSMC 180, and in orange are the design results of TBSizer.

With the TBSizer sizing, the input capacitance reduces by 60%, from 5fF to 3fF. Because they are very small values, the difference is not that impressive. Leakage power is also reduced by 12.5%, from 0.008nW to 0.007nW. The rise delay increased by 43.94%, but note however that the critical path is the fall delay, which reduced 2.08%. Thus, using TBSizer can reduce the difference

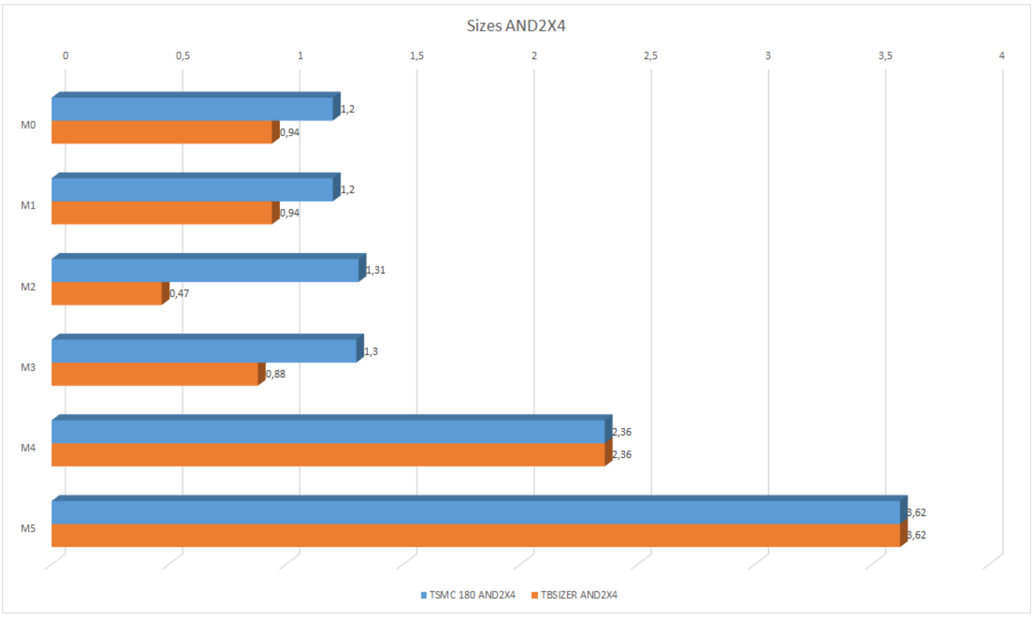


Figure 6.14: The AND2X4 sizes.

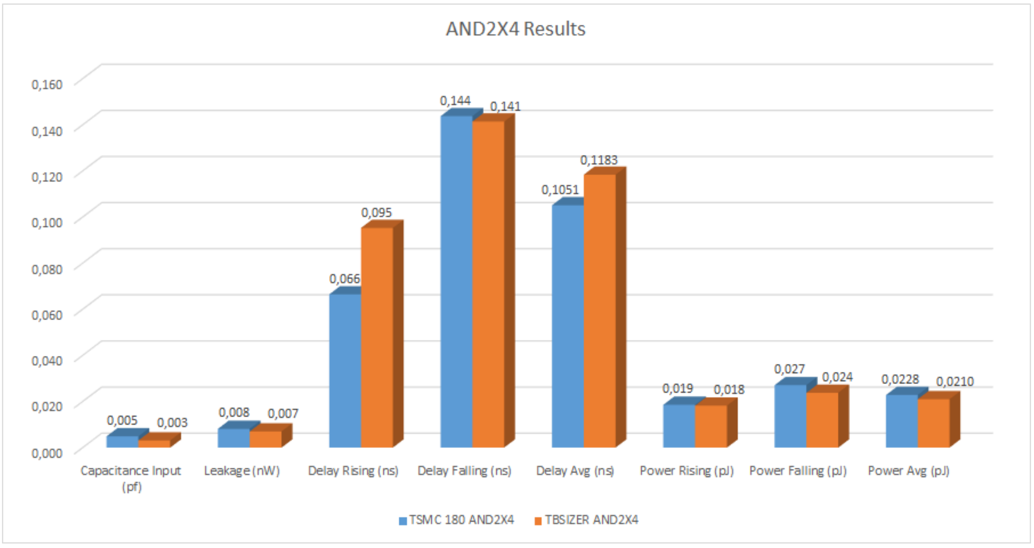


Figure 6.15: The AND2X4 results.

between rising and falling propagation times. But there are also problems, such as the average delay that increased by 12.56%, due to the increase in the delay rising. In addition, rise power decreased by 5.26%, going from 0.019pJ to 0.018pJ and fall power also decreased, this time by 11.11% leading to an average power reduction of 7.89%. Moreover, it is noticeable that the results found with TBSizer reduced the energy consumption of the cell, reduced its input capacitance and also its critical delay, losing only in the average delay of the cell. TBSizer can thus be a good choice to use in projects that require similarity for rise and fall times.

6.3.2 The OR X4 Cell Case

Figure 6.16 shows the schematic representation of the OR cell covered in this Section.

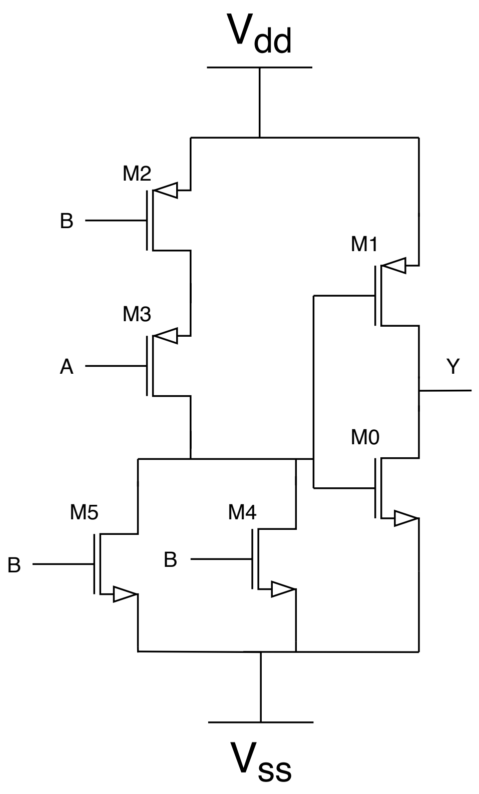


Figure 6.16: The OR cell schematic.

The circuit has six transistors, three PMOS and three NMOS. Sized transistors are M2, M3, M4 and M5. The other two transistors, M0 and M1, are kept in their sizes to supply the required X4 drive.

Figure 6.17 depicts the default sizes provided by TSMC 180 in blue, and the TBSizer computed sizes in orange.

The total cell area decreases with the TBSizer sizing. As expected, transistors M0 and M1 did not change their size, they remained with 2.4u and 3.6u respectively. Transistors M4 and M5 experienced an average decrease of 50%, while transistor M3 decreased by 35.29% and transistor M2 increased by 15.88% in width.

Results provided by Liberate appear in Figure 6.18, where in blue are the results for the standard design of the TSMC 180, and in orange there are the results achieved by TBSizer.

With the TBSizer sizing, the input capacitance reduces by 20%, from 5fF to 4fF. Again, this difference is not quite significant. Leakage power also reduces by 10%, from 0.010nW to 0.009nW. The rise delay increased by 49.33%. Nonetheless, the critical path is the fall delay, which reduced 5.84%. TBSizer can thus, again, reduce the difference between rising and falling propagation times. The average delay increases by 13.99%, due to the increase in the rise delay. In addition, rise power

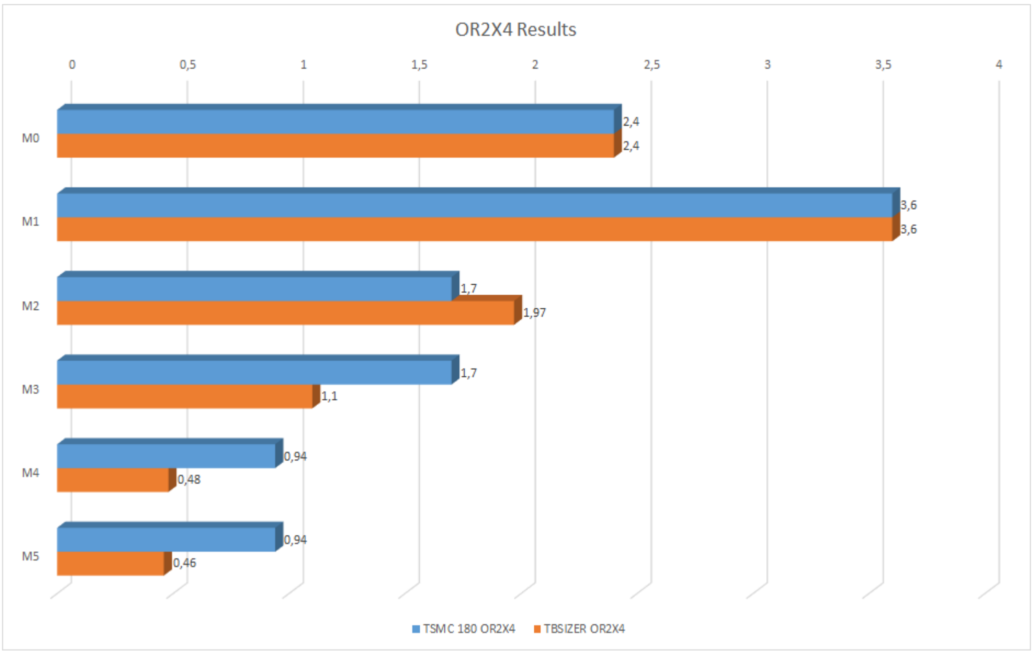


Figure 6.17: The OR2X4 sizes.

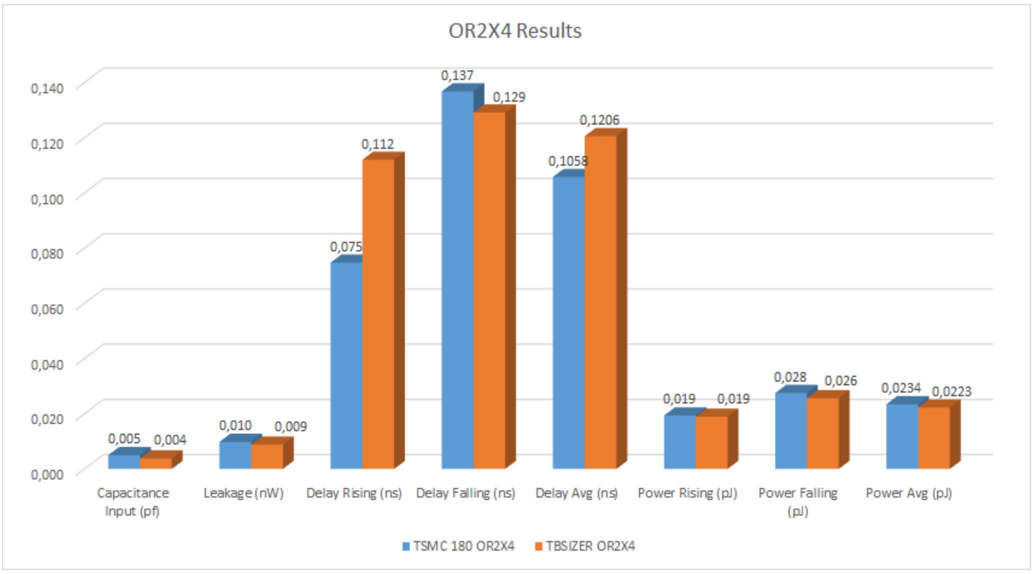


Figure 6.18: The OR2X4 results.

is unchanged, and fall power decreased by 7.14%. The average power accordingly reduces by 4.7%. Results got with TBSizer reduce the cell energy consumption, the input capacitance and also the critical delay, with some loss in the cell average delay. TBSizer shows again to be a good choice for cells that require similar rise and fall times, in addition to being adequate for low energy consumption cells.

6.3.3 The NAND X4 Cell Case

A NAND Cell has characteristics that distinguishes it from the previously approached cells. There is no inverter at the output to determine alone the cell output drive, all cell transistors have to be sized. Drive X4 of the NAND cell is given by the number of fingers it has. Figure 6.19 shows the schematic representation of the NAND cell treated here.

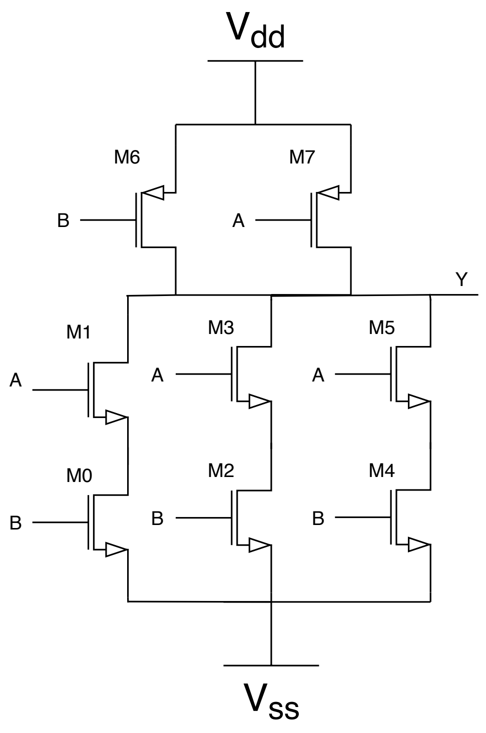


Figure 6.19: The NAND cell schematic.

The circuit has eight transistors, two PMOS and six NMOS. The PMOS transistors are M6 and M7, while the NMOS transistors are M0, M1, M2, M3, M4 and M5.

Figure 6.20 shows the default sizes provided by TSMC 180 in blue, and the sizes obtained by TBSizer in orange.

The total cell area reduces a lot with TBSizer sizing. M6 and M7 transistors shows the largest difference, reducing their size to 15.53% with regard to the sizes in TSMC 180 cells. A pattern appears where transistors closest to the source or ground have are larger than transistors closer to the output node. The same occurs with transistor pairs M0/M1, M2/M3 and M4/M5.

Results provided by Liberate appear in Figure 6.21, blue represents the results for the TSMC 180 original design, while orange represents results from TBSizer.

The TBSizer sizing led to an input capacitance reduction by 61.54%, from 13fF to 5fF. Leakage power also reduces, by 50%, from 0.008nW to 0.004nW. The rise delay increased dramatically, about 108.24%, and the fall delay reduced 48.48%. The fall delay acquired a negative value, managing to switch before the source ended the ramp. However, this sizing is not interesting in terms of time matching, since it worsens the characteristics for the specific cell. The average delay

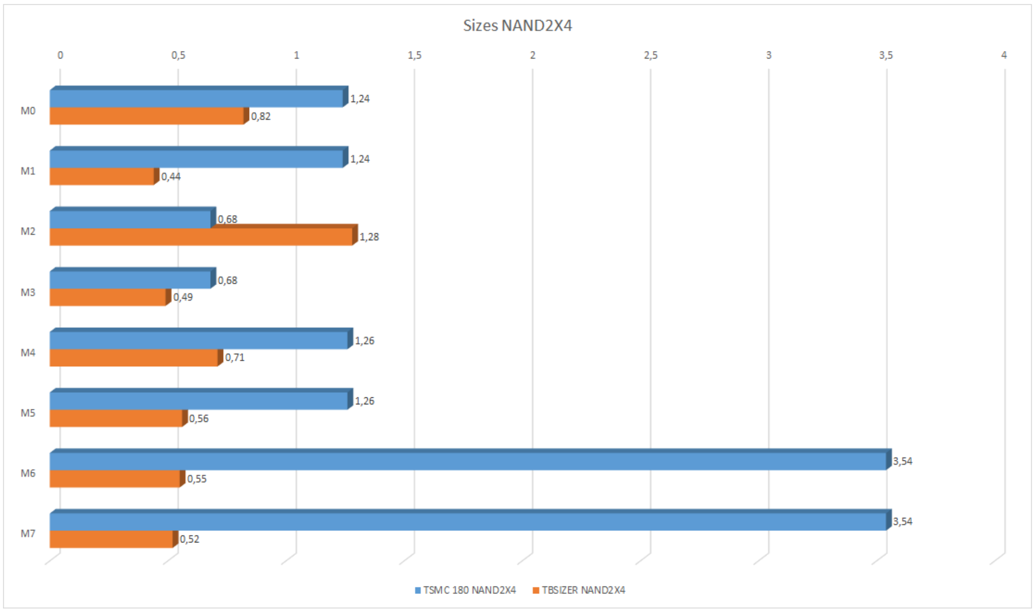


Figure 6.20: The NAND2X4 sizes.

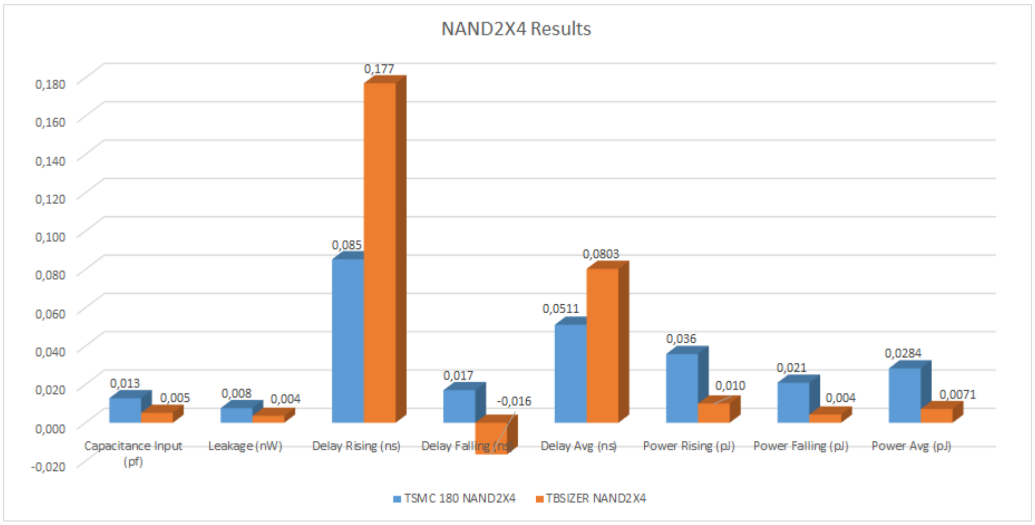


Figure 6.21: The NAND2X4 results.

increased by 57.14%, due to the increase in the rise delay. In addition, the circuit achieved good results with regard to energy consumption. The rise power decreases by 72.22%, and fall power reduces by 80.95%. Accordingly, the average power reduces by 75% to deliver the same signal at the cell output. Moreover, results found with TBSizer greatly reduced the cell energy consumption and the input capacitance, but the cell delay increased.

6.3.4 The XOR X4 Cell Case

The XOR cell is the most complex cell for sizing in this set of experiments. Even with an inverter at the output to determine the cell drive, the sum of all the transistors that are sized is large, 10 transistors. Figure 6.22 shows the schematic representation of the XOR cell studied here.

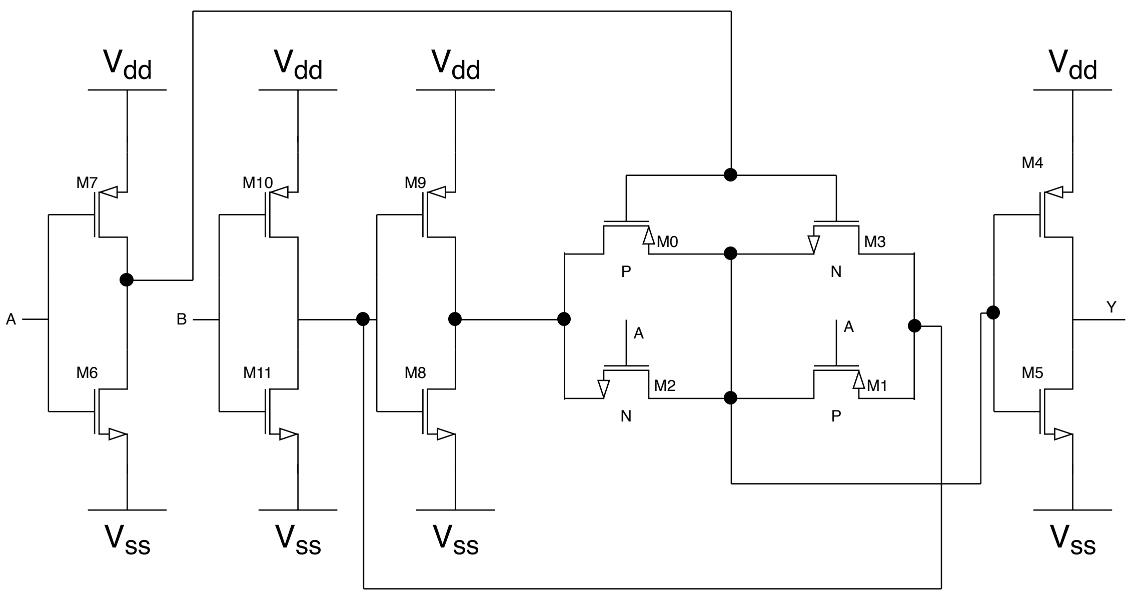


Figure 6.22: The XOR schematic.

The circuit has twelve transistors, six PMOS and six NMOS. Transistors M4 and M5 are used to produce the circuit output, and their size is 2.6u and 3.32u, respectively.

Figure 6.23 shows the sizes from the TSMC 180 cell in blue, and the sizes achieved by TBSizer in orange.

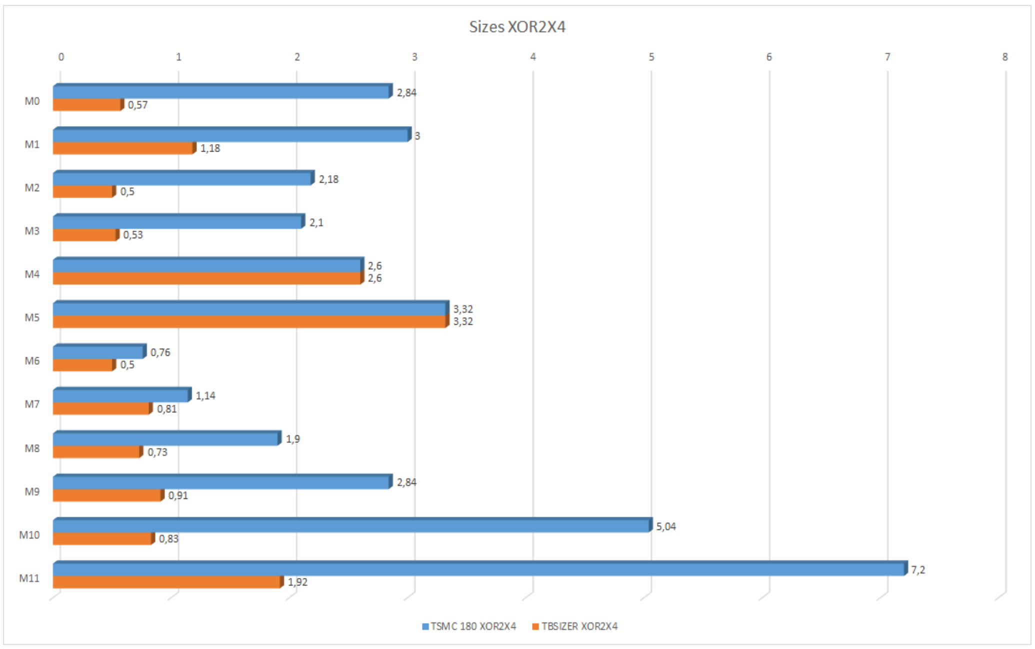


Figure 6.23: The XOR2X4 sizes.

The total cell area reduces significantly in the TBSizer sized cell. All transistors had their size reduced significantly. Four transistors stand out as reducing the width the most: M0 reduces 79.92%, M9 reduces 67.95%, M10 reduces 83.53% and M11 reduces 73.33%.

Results provided by Liberate appear in Figure 6.24, where blue represents results for the standard design of the TSMC 180 library, while orange depicts TBSizer cell results.

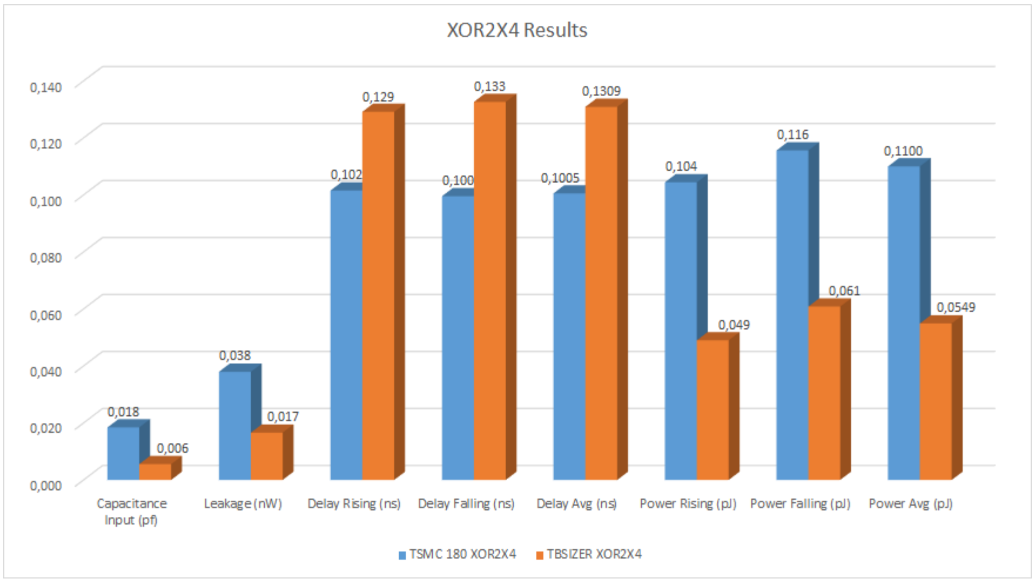


Figure 6.24: The XOR2X4 results.

With the TBSizer sizing, input capacitance reduces by 66.67%, from 18fF to 6fF. Leakage power also reduces, by 55.26%, from 0.038nW to 0.017nW. The rise delay increases by 26.47%, and the fall delay increases by 33%. The average delay accordingly increased by 30.25%. The circuit obtained good results regarding energy consumption. The rise power decreases 52.88%, and fall power reduces 47.41%. Also, the average power is accordingly reduced by 50.09% to deliver the same signal at the cell output. Results found with TBSizer largely reduced the cell energy consumption, the input capacitance and the cell area, at the cost of a slight cell delay increase. TBSizer has thus shows to be an interesting choice for projects with low energy consumption in mind or for those with relevant cell area restrictions.

7. CONCLUSIONS AND FUTURE WORK

This work was essential for the author to be able to apply the knowledge acquired during his undergraduate course. The content of the disciplines such as Microelectronics, VLSI, Digital Circuits and Analysis of Algorithms were widely used during this work development. Moreover, extracurricular content from research projects with a focus on hardware design were essential for the completion of this work.

The development of custom cells to incorporate in an ASIC design has as main objective to achieve the best possible behavior of cells with regard to figures like energy, area and delay.

The work developed compiled the knowledge acquired in the development of transistor networks in a software, TBSizer. TBSizer explores one of the most important issues during cell design, which is the cell sizing step. The developed software uses two approaches for sizing transistors: (1) an exhaustive approach, used for small transistor networks and few transistor selected for sizing into a set of transistor classes (all transistors ina class are sized identically), and (2) the genetic approach, which is capable of sizing much larger networks partitioned into a possibly large number of transistor classes. Each transistor class can contain a single transistor or an arbitrary number of transistors, allowing a wider design space for the designer to explore. Using the genetic approach with several transistor classes, it is possible to compute a sized network that optimizes the relevant figures of merit for the problem at hand.

An initial set of experiments conducted with TBSizer shows the potential of this software. As detailed in Chapter 6, several comparisons were made, with different circuit types. Patterns can be seen as in the C-Element example, where the propagation time is always reduced compared to the ROGen/Ces sizer, with an insignificant increase in power. In fact, power was seen to be significantly reduced in case of X4 C-Element. Another pattern is that by increasing the number of transistor classes, there is a potential for further optimizing the transistor network sizing, but this comes at additional costs in the computational effort conducted by the software. If the number of iterations in the genetic approach is small, this can even make sizing results worse instead of better. Increasing the number of variables, transistors closest to the source/ground terminals are sized bigger than those transistors further from these terminals. Finally, the comparison with the TSMC 180nm ARM standard cell library Sage-X showed that this library is sub-optimal in some aspects, since it was possible to use TBSizer and enhance energy and area for selected cells in this library.

The possibilities for future work are extensive. First, in the process of creating transistor networks, all arcs that are analyzed must be described by the designer. Automatically identifying arcs from the transistor network topology simplifies the transistor sizing process with TBSizer. In addition, the time the designer is required to dedicate to parameterize the transistor network in TBSizer can be reduced.

Another possible future work is the implementation of new space search approaches. The exhaustive approach delivers the optimal overall result, but its execution time is, as expected, very long. The genetic approach is not always able to stay close to the global optimum. Therefore, implementing an approach that runs in polynomial time and achieves optimal results for a significant amount of transistor classes, the sizing problem execution time can be mitigated.

BIBLIOGRAPHY

[AB09] Arora, S.; Barak, B. “Computational Complexity: A Modern Approach”. Cambridge University Press, 2009, 594p.

[Ali10] Alioto, M. “Analysis of Layout Density in FinFET Standard Cells and Impact of Fin Technology”. In: IEEE International Symposium on Circuits and Systems (ISCAS), 2010, pp. 3204–3207.

[ARM08] ARM. “TSMC 180nm Process 1.8V Sage-X Standard Cell Library Databook”, Technical Report, Revision 1.0, 2008, 794p.

[DM04] Ding, L.; Mazumder, P. “A Novel Technique to Improve Noise Immunity of CMOS Dynamic Logic Circuits”. In: ACM/IEEE Design Automation Conference (DAC), 2004, pp. 900–903.

[EBBZ18] El Beqal, A.; Benhala, B.; Zorkani, I. “Sizing of Three-stage Bipolar Transistor Amplifier by the Genetic Algorithm”. In: International Conference on Optimization and Applications (ICOA), 2018, pp. 59–63.

[GFWK12] Grimminger, F.; Fischer, G.; Weigel, R.; Kissinger, D. “Optimum Transistor Sizing for Low-Power Subthreshold Standard Cell Designs”. In: International Semiconductor Conference Dresden-Grenoble (ISCDG), 2012, pp. 151–153.

[GH88] Goldberg, D. E.; Holland, J. H. “Genetic Algorithms and Machine Learning - Guest Editorial”, *Machine Learning*, vol. 3, 1988, pp. 95–99.

[GMGA19] Gupta, P.; Mandadapu, H.; Gourishetty, S.; Abbas, Z. “Robust Transistor Sizing for Improved Performances in Digital Circuits using Optimization Algorithms”. In: International Symposium on Quality Electronic Design (ISQED), 2019, pp. 85–91.

[Kan81] Kang, S. M. “A Design of CMOS Polycells for LSI Circuits”, *IEEE Transactions on Circuits and Systems*, vol. CAS-28–8, 1981, pp. 838–843.

[MFC13] Magarshack, P.; Flatresse, P.; Cesana, G. “UTBB FD-SOI: a Process/Design symbiosis for breakthrough energy-efficiency”. In: Design, Automation and Test in Europe Conference (DATE), 2013, pp. 952–957.

[Mor16] Moreira, M. T. “Asynchronous Circuits: Innovations in Components, Cell Libraries and Design Templates”, Ph.D. Thesis, Pontifícia Universidade Católica do Rio Grande do Sul, FACIN-PPGCC, 2016, 276p.

[Neu08] Neubauer, A. “Theory of the Simple Genetic Algorithm with α-Selection”. In: Annual Conference on Genetic and Evolutionary Computation (GECCO), 2008, pp. 1009–1016.

[PBMR14] Posser, G.; Belomo, J.; Meinhardt, C.; Reis, R. "Perfomance Improvement with Dedicated Transistor Sizing for MOSFET and FinFET Devices". In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2014, pp. 418–423.

[PFWR12] Posser, G.; Flach, G.; Wilke, G.; Reis, R. "Transistor Sizing and Gate Sizing Using Geometric Programming Considering Delay Minimization". In: IEEE International NEWCAS Conference (NEWCAS), 2012, pp. 85–88.

[SJM$^+$18] Singh, K.; Jain, A.; Mittal, A.; Yadav, V.; Singh, A. A.; Jain, A. K.; Gupta, M. "Optimum transistor sizing of CMOS logic circuits using logical effort theory and evolutionary algorithms", *Integration the VLSI Journal*, vol. 60, Jan 2018, pp. 25–38.

[SSSH99] Sutherland, I.; Sproull, R. F.; Sproull, B.; Harris, D. "Logical effort: designing fast CMOS circuits". Morgan Kaufmann, 1999, 256p.

[Sum92] Sumida, B. "Genetics for Genetic Algorithms", *ACM SIGBIO Newsletter*, vol. 12–2, 1992, pp. 44–46.

[Sus19] Susin, G. M. "ASCEND-TSMC180-NCL: A Simple NCL Cell Library to Support Manufacturable QDI Designs", Technical Report, Pontifícia Universidade Católica do Rio Grande do Sul. End of term work, Computer Engineering Course, 2019, 108p.

[Sut89] Sutherlad, I. E. "Micropipelines", *Communications of the ACM*, vol. 32–6, 1989, pp. 720–738.

[WH11] Weste, N. H. E.; Harris, D. M. "CMOS VLSI Design: A Circuits and Systems Perspective". Addison-Wesley, 2011, 4th ed., 867p.

[Whi94] Whitley, D. "A Genetic Algorithm Tutorial", *Statistics and computing*, vol. 4, 1994, pp. 65–85.

[WS99] Wang, L.; Shanbhag, N. R. "Noise-tolerant dynamic circuit design". In: 1999 IEEE International Symposium on Circuits and Systems (ISCAS), 1999, pp. 549–552.

[YZ17] Yaqubi, E.; Zahiri, S. H. "Transistor sizing in latch comparators to achieve optimum specifications". In: Iranian Conference on Electrical Engineering (ICEE), 2017, pp. 222–226.

[ZMPR15] Zimpeck, A. L.; Meinhardt, C.; Posser, G.; Reis, R. "Process Variability in FinFET Standard Cells with Different Transistor Sizing Techniques". In: International Conference on Electronics, Circuits and Systems (ICECS), 2015, pp. 121–124.